Time Limits in Reinforcement Learning

Fabio Pardo [1] **Arash Tavakoli** [1] **Vitaly Levdik** [1] **Petar Kormushev** [1]

Abstract

In reinforcement learning, it is common to let an agent interact with its environment for a fixed amount of time before resetting the environment and repeating the process in a series of episodes. The task that the agent has to learn can either be to maximize its performance over (i) that fixed period, or (ii) an indefinite period where time limits are only used during training to diversify experience. In this paper, we provide a formal account for how time limits could effectively be handled in each of the two cases and explain why not doing so can cause state-aliasing leading to suboptimal policies, instability, slower convergence, and invalidation of experience replay. In case (i), we argue that the terminations due to time limits are in fact part of the environment, and thus a notion of the remaining time should be included as part of the agent's input to avoid violation of the Markov property. In case (ii), the time limits are not part of the environment and are only used to facilitate learning. We argue that this insight should be incorporated into the value-based algorithms by continuing to bootstrap at the end of each partial episode. For both cases, we illustrate, empirically, the significance of our considerations in improving the performance and stability of existing reinforcement learning algorithms, showing state-of-the-art results on several control tasks.

1. Introduction

The reinforcement learning framework (Sutton & Barto, 1998; Szepesvari, 2010; Kaelbling et al., 1996) considers a sequential interaction between an agent and its environment. At every time step t, the agent receives a representation S_t of the environment's state, selects an action A_t that is executed in the environment which in turn provides a representation S_{t+1} of the successor state and a reward signal R_{t+1}. An individual reward received by the agent does not directly indicate the quality of its latest action as some rewards may indeed be the consequence of a series of actions taken far in advance. Thus, the goal of the agent is to learn a good policy by maximizing the discounted sum of future rewards, also known as the *return*:

$$G_t^\gamma \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (1)$$

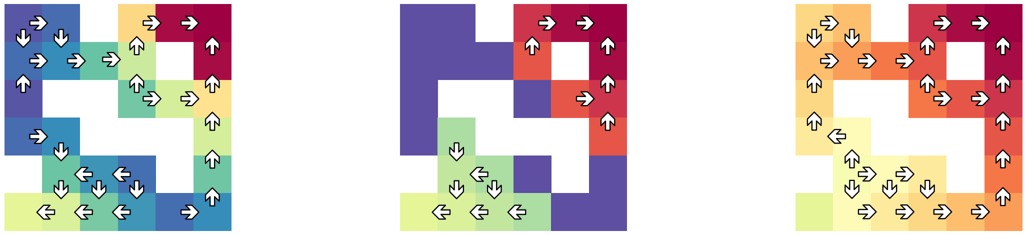
A discount factor $0 \leq \gamma < 1$ is necessary to exponentially decay the future rewards ensuring bounded returns. While the series is infinite, it is common to use this expression even in the case of possible terminations. Indeed, episode terminations can be considered to be the entering of an absorbing state that transitions only to itself and generates zero rewards thereafter. However, when the maximum length of an episode is fixed, it is easier to rewrite the expression above by explicitly including the time limit T:

$$G_{t:T}^\gamma \doteq R_{t+1} + ... + \gamma^{T-t-1} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \quad (2)$$

Optimizing for the expectation of the return specified in Equation 2 is suitable for naturally *time-limited tasks* where the agent has to maximize its expected return $G_{0:T}$ over a fixed episode length only. In this case, since the return is bounded, a discount factor of $\gamma = 1$ can be used. However, in practice it is still common to keep γ smaller than 1 in order to give more priority to short-term rewards. Under this optimality model, the objective of the agent does not go beyond the time limit. Therefore, an agent optimizing under this model should ideally learn to take more risky actions that allow for higher short-term rewards as approaching the time limit. In Section 2, we study this case and illustrate that due to the presence of the time limit, the remaining time is present in the environment's state and is essential to its *Markov property* (Sutton & Barto, 1998). Therefore, we argue for the inclusion of a notion of the remaining time in the agent's input, an approach that we refer to as *time-awareness* (TA). We describe various general scenarios where lacking a notion of the remaining time can lead to suboptimal policies and instability, and demonstrate significant performance improvements for agents with time-awareness.

Optimizing for the expectation of the return specified by Equation 1 is relevant for *time-unlimited tasks* where the

[1]Imperial College London, UK. Correspondence to:
Fabio Pardo <f.pardo@imperial.ac.uk>,
Arash Tavakoli <a.tavakoli@imperial.ac.uk>,
Vitaly Levdik <v.levdik@imperial.ac.uk>,
Petar Kormushev <p.kormushev@imperial.ac.uk>.



(a) Standard (b) Time-awareness (c) Partial-episode bootstrapping

Figure 1: Illustrations of color-coded state-values and policies overlaid on our Two-Goal Gridworld task with two rewarding terminal states (50 for reaching the top-right and 20 for the bottom-left), a penalty of -1 for moving, and a time limit $T = 3$. (a) A standard agent without time-awareness which cannot distinguish timeout terminations from environmental ones. (b) An agent with time-awareness that learns to stay in place when there is not enough time to reach a goal. (c) An agent with partial-episode bootstrapping that bootstraps from any early terminations to maximize its return over an indefinite period.

interaction is not limited in time by nature. In this case, the agent has to maximize its expected return over an indefinite (e.g. infinite) period. However, it is desirable to use time limits in order to diversify the agent's experience. For example, starting from highly diverse states can avoid converging to suboptimal policies that are limited to a fraction of the state space. In Section 3, we show that in order to learn good policies that continue beyond the time limit, it is important to differentiate between the terminations that are due to time limits and those from the environment. Specifically, for value-based algorithms, we argue to continue bootstrapping at states where termination is due to time limits, or more generally any other causes than the environmental ones. We refer to this approach as *partial-episode bootstrapping* (PEB). We describe various scenarios where having a time limit can facilitate learning, but where the aim is to learn optimal policies for indefinite periods, and demonstrate that this approach can significantly improve the performance.

We evaluate the impact of these considerations on a range of novel and popular benchmark domains using a deep reinforcement learning (Arulkumaran et al., 2017; Henderson et al., 2017) algorithm called the Proximal Policy Optimization (PPO), one which has recently been used to achieve state-of-the-art performance in many domains (Schulman et al., 2017; Heess et al., 2017). We use the OpenAI Baselines (Hesse et al., 2017) implementation of PPO with the hyperparameters reported by Schulman et al. (2017), unless stated otherwise. The time-aware version of PPO concatenates the observations provided by the environment and the remaining time represented by a scalar. The partial-episode bootstrapping version of PPO makes a distinction between environment resets and terminations by using the value of the last state in the evaluation of the advantages if no termination is encountered. All novel tasks are implemented using the OpenAI Gym (Brockman et al., 2016) and the standard benchmarks are from the MuJoCo (Todorov et al., 2012) Gym collection. We removed the TimeLimit wrappers around all environments in the case of partial-episode bootstrapping. For each task involving PPO, to achieve perfect reproducibility, we used the same 10 seeds $(0, 1000, 2000, ..., 9000)$ to initialize the pseudo-random number generators for the agents and environments.

We empirically show that time-awareness significantly improves the performance of PPO for the time-limited tasks and can sometimes result in interesting behaviors. For example, in the Hopper-v1 domain, our agent learns to efficiently jump forward and fall towards the end of its time in order to maximize its travelled distance, performing a "photo finish". For the time-unlimited tasks, we show that bootstrapping at the end of partial episodes allows to significantly outperform the standard PPO. In particular, on Hopper-v1, even if trained with episodes of only 200 steps, the agent with partial-episode bootstrapping manages to learn to hop for at least 10^6 time steps (two hours). Finally, we demonstrate that the negative impact of large experience replay buffers shown by Zhang & Sutton (2017) can often be vastly reduced if timeout terminations are properly handled. The source code and videos can be found at: `sites.google.com/view/time-limits-in-rl`.

While the importance of time-awareness for optimizing a time-limited objective (finite horizon) is well-established in the dynamic programming and optimal control literature (Bertsekas, 1995; Bertsekas & Tsitsiklis, 1996) (e.g. model-based backward induction), we observed that it has been largely overlooked in the reinforcement learning literature. This oversight has affected the design of the popular benchmarks, thereby resulting in misleading performance evaluations of the reinforcement learning algorithms. In the view of the above, this paper may serve as an introduction to this concept, and as the first attempt to bring it to bear on the problems and practices of reinforcement learning. Lastly, the thorough analysis of the specific issues which can be caused by the lack of time-awareness in time-limited tasks, the formalization of the partial-episode bootstrapping method, and the extensive empirical evaluations demonstrating improved performance and stability of an existing algorithm using the ideas described in this paper remain novel contributions.

2. Time-awareness for time-limited tasks

In tasks that are time-limited by nature, the learning objective is to optimize the expectation of the return $G^{\gamma}_{0:T}$ from Equation 2. Interactions are systematically terminated at a predetermined time step T if no environmental termination occurs earlier. This time-wise termination can be seen as transitioning to a terminal state whenever the time limit is reached. The states of the agent's environment, formally a *Markov decision process* (MDP) (Puterman, 2014), thus contain a notion of the remaining time used by its transition function. This time-dependent MDP can be thought of as a stack of T time-independent MDPs followed by one that only transitions to a terminal state. Thus, at each time step $t \in \{0, ..., T-1\}$, actions result in transitioning to the next MDP in the stack.

Therefore, a time-unaware agent effectively has to act in a *partially observable Markov decision process* (POMDP) (Lovejoy, 1991) where states that only differ by their remaining time appear identical. This phenomenon is a form of *state-aliasing* (Whitehead & Ballard, 1991) that is known to lead to suboptimal policies and instability due to the infeasibility of correct *credit assignment*. In this case, the terminations due to time limits can only be interpreted as part of the environment's stochasticity where the time-unaware agent perceives a chance of transitioning to a terminal state from any given state. In fact, this perceived stochasticity depends on the current agent's behavioral policy. For example, an agent could choose to stay in a fixed initial state during the entire course of an episode and perceive the probability of termination from that state to be $1/T$, whereas it could choose to always move away from it in which case this probability would be perceived as zero.

In the view of the above, we consider time-awareness for reinforcement learning agents in time-limited domains by including directly the remaining time $T - t$ in the agent's representation of the environment's state or by providing a way to infer it. The importance of the inclusion of a notion of time in time-limited problems was first demonstrated in the reinforcement learning literature by Harada (1997), yet seems to have been largely overlooked. A major difference between the approach of Harada (1997) (i.e. the $\mathrm{Q_T}$-learning algorithm) and that described in this paper, however, is that we consider a more general class of time-dependent MDPs where the reward distribution and the transitions can also be time-dependent, preventing the possibility to consider multiple time instances at once.

Here, we illustrate the issues faced by time-unaware agents via exemplifying the case for value-based methods. The state-value function at time t for a time-aware agent in an environment with time limit T is:

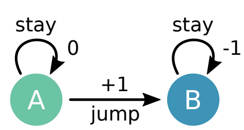
$$v_\pi(s, \boldsymbol{T-t}) \doteq \mathbb{E}_\pi \left[G^{\gamma}_{t:T} | S_t = s \right] \tag{3}$$

By denoting an estimate of the state-value function by $\hat{v}_\pi$, the targets for the *temporal-difference* (TD) updates (Sutton, 1988), after transitioning from a state s to a state s' and receiving a reward r as a result of an action a, are:

$$\begin{cases} r & \text{at all terminations} \\ r + \gamma \hat{v}_\pi(s', \boldsymbol{T-t-1}) & \text{otherwise} \end{cases} \tag{4}$$

The added notion of the remaining time is indicated in bold and blue. A time-unaware agent would be deprived of this information and thus would update $\hat{v}_\pi(s)$ with or without bootstrapping from the estimated value of s' depending on whether the time limit is reached. Confused by the conflicting updates for estimating the value of the same state, instead of learning an accurate value function, this time-unaware agent learns an approximate average of these inconsistent updates. It is worth noting that, for time-aware agents, if the time limit is never varied, the inclusion of the elapsed time t would be sufficient. This time could be measured by the agent itself from the beginning of the current episode without adding anything to the observations from the environment. For more generality, however, we chose to always represent the remaining time. In practice, we normalized the remaining time from 1 to -1 and concatenated it to the observations from the Gym environments.

2.1. The Last Moment problem



To give a simple example of the learning of an optimal time-dependent policy, we consider an MDP containing two states A and B. The agent always starts in A and has the possibility to choose an action to "stay" in place with no rewards or a "jump" action that transitions it to state B with a reward of $+1$. However, state B is a trap with no exit where the only possible action leads to a penalty of -1. The episodes terminate after a fixed number of steps T. The goal of the game is thus to jump at the last moment. For a time-unaware agent, the task is impossible to master for $T > 1$ and the best feasible policy would be to stay in place, resulting in an overall return of 0. In contrast, a time-aware agent can learn to stay in place for $T - 1$ steps and then jump, scoring an undiscounted sum of rewards of $+1$.

2.2. The Two-Goal Gridworld problem

To further illustrate the impact of state-aliasing for time-unaware agents, we consider a deterministic gridworld environment (see Figure 1) with two possible goals rewarding

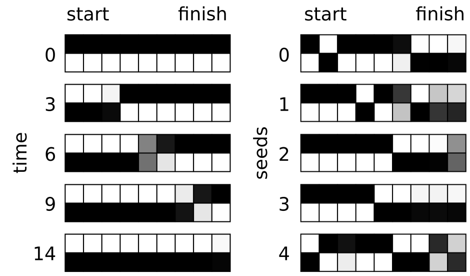
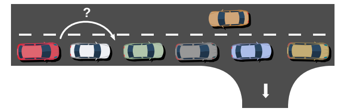


Figure 2: Heat map of the learned action probabilities overlaid on our Queue of Cars problem (black and white indicate 0 and 1, respectively). For each block, the top row represents the dangerous action and the bottom row the safe one. The 9 non-terminal states are represented horizontally. Left: A time-aware PPO agent at various time steps: the agent learns to optimally select the dangerous action. Right: 5 different instances of the time-unaware PPO agent.

50 for reaching the top-right and 20 for the bottom-left cells. The agent has 5 actions: to move in cardinal directions or to stay in place. Any movement incurs a penalty of -1 while staying in place generates no reward. Episodes terminate after 3 time steps or if the agent has reached a goal. The initial state is randomly selected for every episode, excluding goals. We used a tabular Q-learning (Watkins & Dayan, 1992) with random actions, trained until convergence with a decaying learning rate and a discount factor of 0.99.

The time-aware agent has a state-action value table for each time step and easily learns the optimal policy which is to go for the closest goal when there is enough time, and to stay in place otherwise. For the time-unaware agent, the greedy values of the cells adjacent to the top-right and bottom-left goals converge to 49 and 19, respectively. Then, since $T = 3$, from each remaining cell, the agent has between 1 and 3 steps. If it moves it receives a penalty and for $2/3$ of the times bootstraps from the successor cell. Thus, for $v(s) = \arg\max_a q(s,a)$ and $N(s)$ denoting the neighbors of s, for states non adjacent to the goals we have: $v(s) = 2/3(-1 + \gamma \max_{s' \in N(s)} v(s')) + 1/3(-1)$. This learned value function leads to a policy that always tries to go for the closest goal even if there is not enough time. While the final optimal policy does not actually require time information, this example clearly shows that the confusion during training due to state-aliasing can create a leakage of the values to states that are out of reach. It is worth noting that, Monte Carlo methods such as REINFORCE (Williams, 1992; Sutton et al., 2000) are not susceptible to this leakage as they use complete returns instead of bootstrapping. However, without awareness of the remaining time, Monte Carlo methods would still not be able to learn an optimal policy in many cases, such as the Last Moment problem.

2.3. The Queue of Cars problem



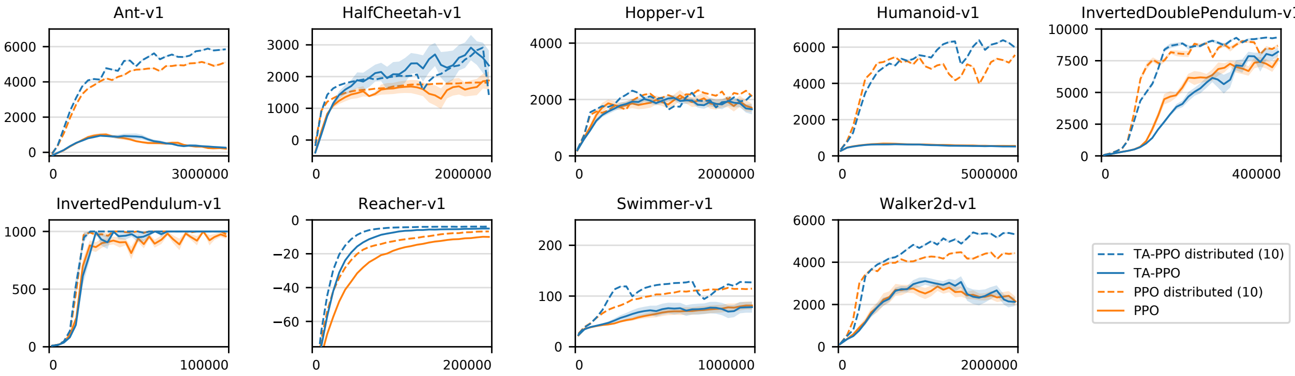
An interesting property of time-aware agents is the ability to dynamically adapt to the remaining time that can, for example, be correlated with the current progress of the agent. To illustrate this, we introduce an environment which we call Queue of Cars where the agent controls a vehicle that is held up behind an intermittently moving queue of cars. The agent's goal is to reach an exit located 9 slots away from its starting position. At any time, the agent can choose the "safe" action to stay in the queue which may result in advancing to the next slot with 50% probability. Alternatively, it has the option to attempt to overtake by taking the "dangerous" action that has 80% probability to advance but poses a 10% chance of collision with the oncoming traffic and terminating the episode. The agent receives 0 reward unless it reaches the destination, where the episode terminates with a reward of $+1$.

In this task, an agent can have a lucky sequence of safe transitions and reach the destination within the time limit without ever needing to attempt an overtake. However, the opposite can also happen in which case the agent would need to overtake the cars to reach its destination in time. Time-unaware agents cannot possibly gauge the necessity to rush and thus can only learn a statistically efficient combination of dangerous and safe actions based on position only. Figure 2 (left) shows a time-aware agent which adapts to the remaining time and its distance to the goal, while (right) illustrates that 5 different time-unaware PPOs fail to do so. A discount factor of $\gamma = 1$ was used for both agents.

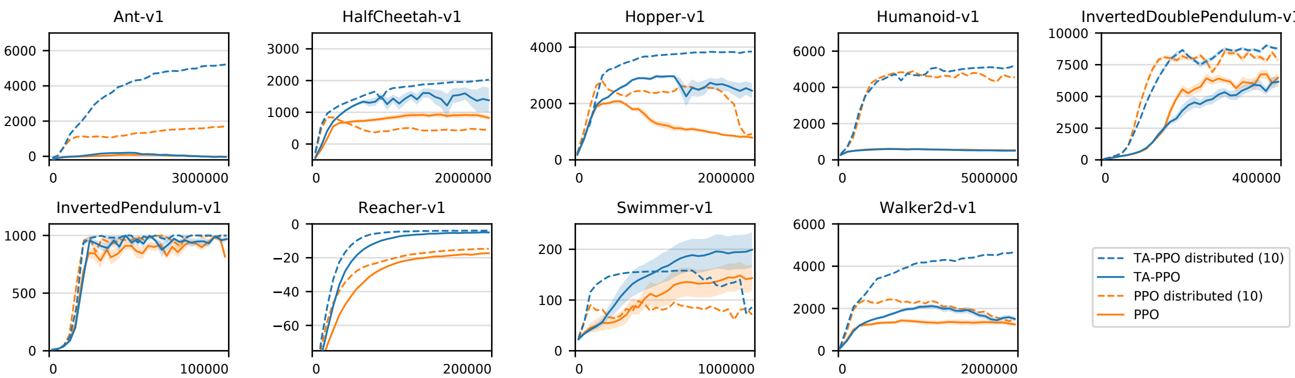
2.4. Standard control tasks

In this section, we compare the performance of PPO with and without the remaining time as part of the agent's input on a wide range of continuous control tasks from the OpenAI's MuJoCo Gym benchmarks (Brockman et al., 2016; Duan et al., 2016) (see Figure 3). By default, these environments use predefined time limits. The results demonstrate that time-awareness (TA) significantly improves the performance and stability of PPO. To better illustrate the differences between the time-aware agent and the time-unaware one, we now provide several more observations.

As illustrated in Figure 3a, for a discount rate of 0.99, often, the standard PPO is initially on par with the time-aware PPO and later starts to plateau (e.g. Walker2d-v1 and Humanoid-v1). This is due to the fact that, in some domains, the agents start to experience terminations due to the time limit more frequently as they become better, at which point the time-unaware agent begins to perceive inconsistent returns for seemingly similar states. The advantage of time-awareness becomes even clearer in the case of a discount rate of 1



(a) Results for $\gamma = 0.99$.



(b) Results for $\gamma = 1$.

Figure 3: Performance comparison of PPO with and without the remaining time in input on several continuous control tasks from the OpenAI's MuJoCo Gym. (a) $\gamma = 0.99$. (b) $\gamma = 1$. For time-aware (TA-PPO) and standard PPO, we compared simple (averaged over 10 seeds) and distributed implementations (10 synchronous workers).

where the time-unaware PPO often diverges drastically (see Figure 3b). This is mainly because, in this case, the time-unaware agent experiences much more significant conflicts as returns are now the sum of the undiscounted rewards.

Figure 4 (left) shows the learned state-value estimations for InvertedPendulum-v1 which perfectly illustrate the difference between a time-aware agent and a time-unaware one in terms of their estimated expected returns. While time-awareness enables PPO to learn an accurate exponential or linear decay of the expected return with time, the time-unaware one only learns a constant estimate.

Time-awareness does not only help agents by avoiding the conflicting updates. In fact, in naturally time-limited tasks where the agents have to maximize their performance for a limited time, time-aware agents can demonstrate interesting ways of achieving this objective. Figure 4 shows the average final pose of the time-aware (middle) and time-unaware (right) agents. We can see that the time-aware agent robustly learns to jump towards the end of its time in order to maximize its expected return, resulting in a "photo finish". Finally, Figure 4 (bottom-right) shows an interesting behavior robustly demonstrated by the time-unaware PPO in the case of $\gamma = 1$ that is to actively stay in place, accumulating at least the rewards coming from the bonus for staying alive.

In this section, we explored the scenario where the aim is to learn a policy that maximizes the expected return over a limited time. We argued for inclusion of a notion of the remaining time as part of the agent's observation to avoid state-aliasing which can cause suboptimal policies and instability. However, this scenario is not always ideal as there are cases where, even though the agent experiences time limits in its interaction with the environment, the objective is to learn a policy for a time-unlimited task. For instance, as we saw for Hopper-v1, the learned policy that maximizes the return over 300 steps generally results in a photo finish which would lead to a fall and subsequent termination if the simulation was to be extended. Such a policy is not viable if the goal is to learn to move forward for an indefinite period. One solution is to not have time limits during training. However, it is often more efficient to instead have short snippets of interactions to expose the agent to diverse experiences. In the next section, we explore this case and show how to effectively learn in such domains from partial episodes.

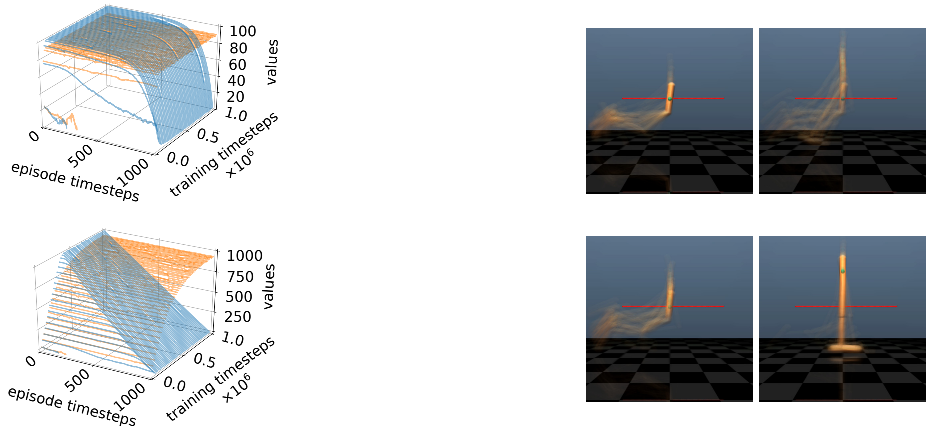


Figure 4: Comparison of PPO with and without the remaining time in input. Left: Learned state-value estimations of time-aware (blue) and standard PPO (orange) on InvertedPendulum-v1 ($T = 1000$). Average last pose on Hopper-v1 ($T = 300$) of the time-aware PPO agent (Middle) and time-aware PPO (Right). Top: $\gamma = 0.99$. Bottom: $\gamma = 1$. A large discount factor highly destabilizes the time-unaware PPO. The vertical termination threshold of 0.7 meters is indicated in red. The time-aware agent learns to jump forward before the time limit.

3. Partial-episode bootstrapping for time-unlimited tasks

In tasks that are not time-limited by nature, the learning objective is to optimize the expectation of the return G_0^γ from Equation 1. While the agent has to maximize its expected return over an indefinite (possibly infinite) period, it is desirable to still use time limits in order to frequently reset the environment and increase the diversity of the agent's experiences. A common mistake, however, is to then consider the terminations due to such time limits as environmental ones. This is equivalent to optimizing for returns $G_{0:T}^\gamma$ (Equation 2), not accounting for the possible future rewards that could have been experienced if no time limits were used.

In the case of value-based algorithms, we argue for continuing to bootstrap at states where termination is due to the time limit. The state-value function of a policy at time t can be rewritten in terms of the time-limited return $G_{t:T}^\gamma$ and the bootstrapped value from the last state $v_\pi(S_T)$:

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_{t:T}^\gamma + \boldsymbol{\gamma^{T-t} v_\pi(S_T)} | S_t = s \right] \tag{5}$$

By denoting an estimate of the state-value function by $\hat{v}_\pi$, the targets for the temporal-difference update rule after transitioning from a state s to a state s' and receiving a reward r as a result of an action a are:

$$\begin{cases} r & \text{at } \textbf{environmental} \text{ termination} \\ r + \gamma \hat{v}_\pi(s') & \text{otherwise } (\textbf{including for } \boldsymbol{t = T - 1}) \end{cases} \tag{6}$$

The partial-episode bootstrapping method is indicated in bold and green. An agent without this modification would update $\hat{v}_\pi(s)$ with or without bootstrapping from the estimated value of s' depending on whether there is some remaining time or not. Similarly to Equation 4, the conflicting updates for estimating the value of the same state leads to an approximate average of these updates.

In the previous section, one of the issues came from bootstrapping values from states that were out-of-reach, letting the agent falsely believe that more rewards were available after. On the opposite, the problem presented here is when systematic bootstrapping is not performed from states at the time limit and thus, forgetting that more rewards would actually be available thereafter.

3.1. The Two-Goals Gridworld problem

We revisit the gridworld environment from Section 2.2. While previously the agent's task was to learn an optimal policy for a given time limit, we now consider how an agent can learn a good policy for an indefinite period from partial-episode experiences. The same setup and tabular Q-learning from Section 2.2 were used, but instead of considering terminations due to time limits as environmental ones, bootstrapping is maintained from the non-terminal states that are reached at the time limits. This modification allows our agent to learn the time-unlimited optimal policy of always going for the most rewarding goal (see Figure 1c). On the other hand, while the standard agent that is not performing the final bootstrapping (see Figure 1a) had values from out-of-reach cells leaking into its learned value function, these updates did not occur in sufficient proportion to let the agent learn the time-unlimited optimal policy.

For the next experiments, we again used PPO but with two key modifications for the partial-episode-bootstrapping (PEB) agent. First we removed the Gym's TimeLimit wrap-

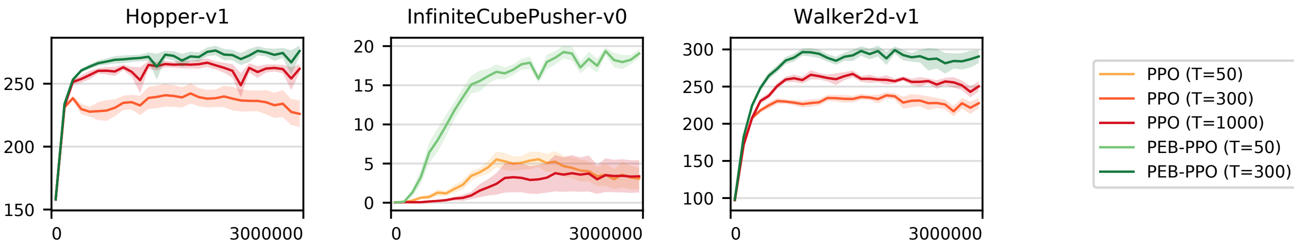


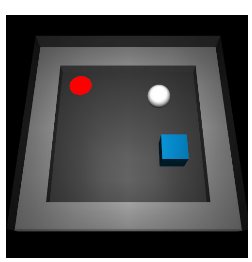
Figure 5: Performance comparison of PPO with and without partial-episode bootstrapping with $\gamma = 0.99$ on several continuous control tasks. For Hopper-v1 and Walker-v1, the evaluation episodes are limited to 10^6 time steps and the discounted sum of rewards is represented, while for InfiniteCubePusher-v0 the evaluations are limited to 1000 time steps and the number of targets reached is represented.

per that is included by default for all environments and enforces termination when time limits are reached. Second, we modified the PPO's implementation to enable continuing to bootstrap when the environment is reset but no termination is encountered. This involves changing the implementation of the generalized advantage estimation (GAE) (Schulman et al., 2016). While GAE uses an exponentially-weighted average of n-step value estimations for bootstrapping that are more complex than the one-step lookahead bootstrapping explained in Equation 6, continuing to bootstrap from the last non-terminal states is the only modification required for the considered approach.

3.2. Hopper and Walker

Here, we consider the Hopper-v1 and Walker2d-v1 environments from Section 2.4, but instead aim to learn a policy that maximizes the agent's expected return over a time-unlimited horizon. The aim here is to show that by continuing to bootstrap from episode terminations that are due to time limits only, we are able to learn good policies for time-unlimited domains. Figure 5 (left and right) demonstrates performance evaluations of the standard PPO against one with partial-episode bootstrapping (PEB). During training, partial episodes were limited to 300 time steps, while during evaluation episodes were limited to 10^6 time steps. The PPO agent with the partial-episode bootstrapping method significantly outperforms the standard PPO.

3.3. The Infinite Cube Pusher task



To demonstrate the ability of our agent in optimizing for an infinite-horizon (no terminal state) objective, we propose a novel MuJoCo domain consisting of a torque-controlled ball that is used to push a cube to specified target positions. Once the cube has touched the target, the agent is rewarded and the target is moved away from the cube to a new random position. Because the task lacks terminal states, it can continue indefinitely. The objects are surrounded by fixed bounding walls. The inner edge of the walls stops the cube but not the ball in order to let the agent move the cube even if it is in a corner. The movements of the ball are limited to the horizontal plane and to the area defined by the outer edge of the walls. The environment's state representation consists of the objects' coordinates and velocities, and the cube's rotation. The agent receives no rewards unless the cube reaches a target location, at which point the agent receives a reward of $+1$. Due to the absence of reward shaping, it is necessary to limit episodes in time to diversify the experiences in order to learn to solve the task. Therefore, during training, a time limit of 50 time steps was used, sufficient to push the cube to one target in most cases. During evaluation, however, 1000 steps were used to allow successfully reaching several targets. Figure 5 (middle) shows the performance comparison of the standard PPO against one with our modification. An entropy coefficient of 0.01 was used to encourage exploration.We found this value to yield best performance for both agents. The partial-episode bootstrapping method drastically outperforms the standard PPO.

3.4. Experience replay

Sampling batches of transitions from a buffer of past experience, known as experience replay, has proved to be highly effective in stabilizing the training of artificial neural networks by decorrelating updates and avoiding the rapid forgetting of rare experiences (Mnih et al., 2015; Schaul et al., 2016). However, we argue that the perceived non-stationarity, induced by not properly handling time limits, is incompatible with experience replay. Indeed, the timeout-occurrence distribution changes with the behavior of the agent, and thus past transitions become obsolete.

While both time-awareness and partial-episode bootstrapping (PEB) provide ways to solve this issue, we chose to illustrate the effect of PEB on one of the tasks presented in (Zhang & Sutton, 2017). In the latter, the authors demon-

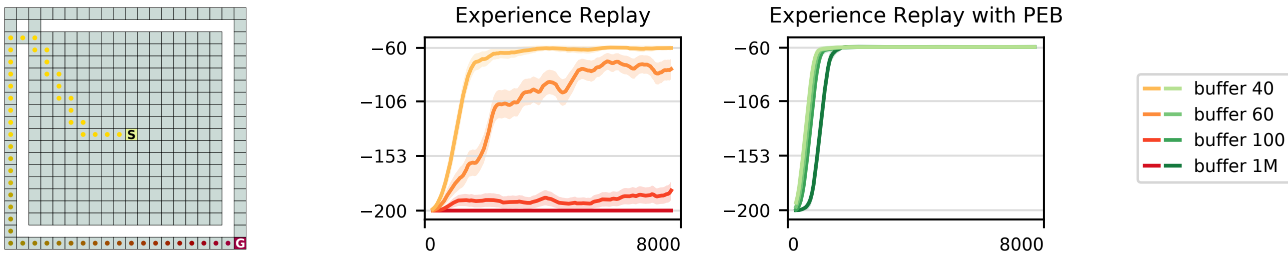


Figure 6: Sum of rewards with respect to episodes of training using a tabular Q-learning on the Difficult Gridworld problem presented in (Zhang & Sutton, 2017). When timeout terminations are not properly considered, experience replay significantly hurts the performance, while by simply continuing to bootstrap whenever a timeout termination is encountered, the learning is much faster and varying the buffer size has almost no effect.

strate that experience replay can significantly hurt the learning process if the size of the replay buffer is not tuned well. One of the environments used is a deterministic gridworld with a fixed starting point and goal, shown in Figure 6 with a solution. As proposed by the authors, a tabular Q-learning algorithm is used with values initialized to 0, a penalty of -1 at each time step, no discount, a time limit $T = 200$, and an ϵ-greedy exploration using a fixed 10% chance of random actions. Figure 6 shows the performance with respect to the number of training episodes, averaged over 30 seeds, from 0 to 29. We successfully replicated the figure showing that the performance deteriorates very quickly with buffer size and demonstrate that by simply bootstrapping from states when the time limit is reached, as proposed in PEB, the effect of the buffer size is vastly diminished.

4. Discussion

We showed in Section 2 that time-awareness is required for correct credit assignment in domains where the agent has to optimize its performance over a time-limited horizon. However, common time-unaware agents still often manage to perform relatively well. This could be due to several reasons including: if time limits are so long that timeouts are hardly ever experienced (e.g. in the Arcade Learning Environment (ALE) (Bellemare et al., 2013; Machado et al., 2017) domains where $T = 5$ minutes), if there are clues in the observations that are correlated with time (e.g. the forward distance), if it is not likely to observe the same states at different remaining times, or if the discount factor is sufficiently small to reduce the impact of the confusion. Furthermore, many methods exist to handle POMDPs (Lovejoy, 1991). In deep learning (LeCun et al., 2015; Schmidhuber, 2015), it is highly common to use a stack of previous observations or recurrent neural networks (RNNs) (Goodfellow et al., 2016) to address partial observations (Wierstra et al., 2009). These solutions may to an extent help when a notion of the remaining time is not included as part of the agent's input. However, including this information is much simpler and allows better diagnosis of the learned policies. The considered approach is rather generic and can be applied to domains with varying time limits. Finally, in real-world applications, such as robotics, the real clock time can be used in place of discrete time steps.

In order for the partial-episode bootstrapping method in Section 3 to work, as is the case for value-based methods in general, the agent needs to bootstrap from reliable estimated predictions. This is in general resolved by enabling sufficient exploration. However, when the interactions are limited in time, exploration of the full state-space may not be feasible from some fixed starting states. Thus, a good way to allow appropriate exploration in such domains is to sufficiently randomize the initial states. It is worth noting that partial-episode bootstrapping is quite generic in that it is not restricted to partial episodes only due to time limits. In fact, this approach is valid for any early termination causes. For instance, it is common in curriculum learning to start from states nearby the goals (easier tasks), and gradually expand to further states (more difficult tasks) (Florensa et al., 2017). In this case, it can be helpful to stitch the learned values by terminating the episodes and bootstrapping as soon as the agent enters a well-known state.

5. Conclusion

We considered the problem of learning optimal policies in time-limited and time-unlimited domains using time-limited interactions. We showed that time limits should be carefully manipulated to avoid state-aliasing and perceived non-stationarity of the environment. We explained that when learning policies for time-limited tasks, it is important to include a notion of the remaining time as part of the agent's input. We then showed that, when learning policies that are optimizing for time-unlimited tasks, it is more appropriate to continue bootstrapping at the end of the partial episodes when termination is due to time limits, or any early termination causes other than the environmental ones. In both cases we observed significant improvements in the performance of the considered reinforcement learning algorithms.

Acknowledgments

Research presented in this paper has been supported by Dyson Technology Ltd.

References

Arulkumaran, Kai, Deisenroth, Marc P, Brundage, Miles, and Bharath, Anil A. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

Bellemare, Marc G, Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The Arcade Learning Environment: an evaluation platform for general agents. *Journal of Artificial Intelligence Research (JAIR)*, 47:253–279, 2013.

Bertsekas, Dimitri P. *Dynamic programming and optimal control*. Athena Scientific, 1995.

Bertsekas, Dimitri P. and Tsitsiklis, John. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

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

Duan, Yan, Chen, Xi, Houthooft, Rein, Schulman, John, and Abbeel, Pieter. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning (ICML)*, pp. 1329–1338, 2016.

Florensa, Carlos, Held, David, Wulfmeier, Markus, and Abbeel, Pieter. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*, 2017.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016.

Harada, Daishi. Reinforcement learning with time. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 577–582. AAAI Press, 1997.

Heess, Nicolas, Sriram, Srinivasan, Lemmon, Jay, Merel, Josh, Wayne, Greg, Tassa, Yuval, Erez, Tom, Wang, Ziyu, Eslami, Ali, Riedmiller, Martin, and Silver, David. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

Henderson, Peter, Islam, Riashat, Bachman, Philip, Pineau, Joelle, Precup, Doina, and Meger, David. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

Hesse, Christopher, Plappert, Matthias, Radford, Alec, Schulman, John, Sidor, Szymon, and Wu, Yuhuai. OpenAI Baselines. `https://github.com/openai/baselines`, 2017.

Kaelbling, Leslie Pack, Littman, Michael L, and Moore, Andrew W. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research (JAIR)*, 4:237–285, 1996.

LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *Nature*, 521(7553):436–444, 2015. URL `http://dx.doi.org/10.1038/nature14539`.

Lovejoy, William S. A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991. URL `https://doi.org/10.1007/BF02055574`.

Machado, Marlos C, Bellemare, Marc G, Talvitie, Erik, Veness, Joel, Hausknecht, Matthew, and Bowling, Michael. Revisiting the Arcade Learning Environment: evaluation protocols and open problems for general agents. *arXiv preprint arXiv:1709.06009*, 2017.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. URL `http://dx.doi.org/10.1038/nature14236`.

Puterman, Martin L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

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

Schmidhuber, Jürgen. Deep learning in neural networks: an overview. *Neural Networks*, 61:85–117, 2015. URL `https://doi.org/10.1016/j.neunet.2014.09.003`.

Schulman, John, Moritz, Philipp, Levine, Sergey, Jordan, Michael, and Abbeel, Pieter. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016.

Schulman, John, Wolski, Filip, Dhariwal, Prafulla, Radford, Alec, and Klimov, Oleg. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sutton, Richard S. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1): 9–44, 1988. URL `https://doi.org/10.1007/BF00115009`.

Sutton, Richard S and Barto, Andrew G. *Reinforcement Learning: an Introduction*. MIT Press, 1998.

Sutton, Richard S, McAllester, David A, Singh, Satinder P, and Mansour, Yishay. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1057–1063, 2000.

Szepesvari, Csaba. *Algorithms for Reinforcement Learning*. Morgan and Claypool, 2010.

Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. MuJoCo: a physics engine for model-based control. In *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pp. 5026–5033, 2012.

Watkins, Christopher J. C. H. and Dayan, Peter. Q-learning. *Machine Learning*, 8(3):279–292, 1992. URL `http://dx.doi.org/10.1007/BF00992698`.

Whitehead, Steven D. and Ballard, Dana H. Learning to perceive and act by trial and error. *Machine Learning*, 7(1): 45–83, 1991. URL `https://doi.org/10.1023/A:1022619109594`.

Wierstra, Daan, Förster, Alexander, Peters, Jan, and Schmidhuber, Jürgen. Recurrent policy gradients. *Logic Journal of IGPL*, 18(5):620–634, 2009.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992. URL `https://doi.org/10.1007/BF00992696`.

Zhang, Shangtong and Sutton, Richard S. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.