
Time Limits in Reinforcement Learning

Fabio Pardo, Arash Tavakoli, Vitaly Levdiik & Petar Kormushev
Imperial College London
London SW7 2AZ, United Kingdom
{f.pardo,a.tavakoli,v.levdik,p.kormushev}@imperial.ac.uk

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 investigate theoretically how time limits could effectively be handled in each of the two cases. In the first one, we argue that the terminations due to time limits are in fact part of the environment, and propose to include a notion of the remaining time as part of the agent’s input. In the second case, the time limits are not part of the environment and are only used to facilitate learning. We argue that such terminations should not be treated as environmental ones and propose a method, specific to value-based algorithms, that incorporates this insight by continuing to bootstrap at the end of each partial episode. To illustrate the significance of our proposals, we perform several experiments on a range of environments from simple few-state transition graphs to complex control tasks, including novel and standard benchmark domains. Our results show that the proposed methods improve the performance and stability of existing reinforcement learning algorithms.

1 Introduction

The reinforcement learning framework (Sutton & Barto, 1998; Bertsekas & Tsitsiklis, 1996; 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 *return*:

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

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 :

$$G_{t:T}^\gamma \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-t-1} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \quad (2)$$

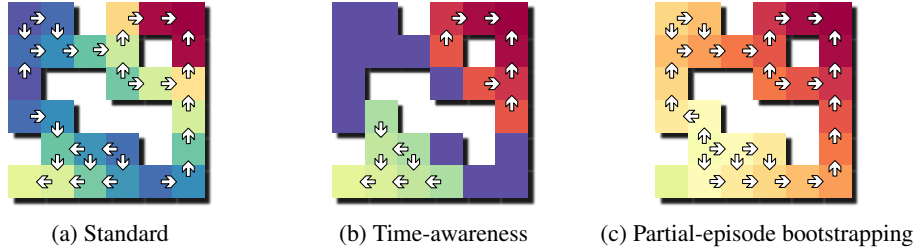


Figure 1: Illustrations of color-coded state-values and policies overlaid on our Two-Goal Gridworld problem with two rewarding terminal states (50 for reaching the top-right corner and 20 for the bottom-left), a penalty of -1 for moving, and a time limit $T = 3$. (a) A standard agent without time-awareness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

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 end of 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 propose to include a notion of the remaining time in the agent’s input, an approach that we refer to as *time-awareness*. 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 time-aware agents.

Optimizing for the expectation of the return specified by Equation 1 is relevant for *time-unlimited tasks* where the 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 propose 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 method as *partial-episode bootstrapping*. 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 our method can significantly improve the performance.

We evaluate the impact of the proposed methods on a range of novel and popular benchmark domains using a deep reinforcement learning (Arulkumaran et al., 2017; Henderson et al., 2017) algorithm called 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 the PPO algorithm with the hyperparameters reported by Schulman et al. (2017), unless stated otherwise. All novel environments are implemented using the OpenAI Gym framework (Brockman et al., 2016) and the standard benchmark domains are from the MuJoCo (Todorov et al., 2012) Gym collection. We modified the TimeLimit wrapper to include remaining time in the observations for the proposed time-aware agent and a flag to separate timeout terminations from environmental ones for the proposed partial-episode bootstrapping agent. For every task involving PPO, to have perfect reproducibility, we used the same 40 seeds from 0 to 39 to initialize the pseudo-random number generators for the agents and environments. Every 5 training cycles (i.e. 10240 time steps), we perform an evaluation on a complete episode and store the sums of rewards, discounted returns, and estimated state-values. For generating the performance plots, we average the values across all runs and then apply smoothing with a sliding window of size 10. The performance graphs show these smoothed averages as well as their standard error.

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 nish”. 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 2×10^6 steps, our agent manages to learn to hop for at least 10^6 time steps (two hours). Detailed results are available in the Appendix, the source code and videos can be found <https://sites.google.com/view/time-limits-in-rl>.

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 $G_{0:T}$ 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 must 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 $t \in \{0, \dots, T-1\}$, actions result in transitioning to the next MDP in the stack.

Thus, a time-unaware agent effectively has to act on 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 ϵ , 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 propose time-awareness for reinforcement learning agents in time-limited domains by including directly the remaining time 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 by Harada (1997), but seems to have been overlooked in the design of the benchmark domains and the evaluation of reinforcement learning agents. A major difference between the approach of Harada (1997) and that of ours, 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 as it is the case for the learning algorithm proposed by the author.

Here, we illustrate the issues faced by time-unaware agents by 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:

$$v(s; T-t) \doteq E[G_{t:T} | S_t = s] \quad (3)$$

By denoting an estimate of the state-value function by v , the targets for the temporal-difference (TD) updates (Sutton, 1988), after transitioning from a state s^0 and receiving a reward r as a result of an action a , are:

$$r \quad \text{at termination (including when } t = T-1) \\ r + \gamma v(s^0; T-t-1) \quad \text{otherwise} \quad (4)$$

The proposed added notion of the remaining time is indicated in bold blue. A time-unaware agent would be deprived of this information and thus would update $v(s)$ with or without bootstrapping from the estimated value v^p 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, but for more generality, we chose to always represent the remaining time. In practice, we simply normalized the remaining time from 0 and concatenated it to the observations from the Gym environments by modifying the TimeLimit wrapper.

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: different instances of the time-unaware PPO agent.

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. 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 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 $T - 1$.

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 reward 10 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 T 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 $\gamma = 0.99$.

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 19 and 19, respectively. Then, since $\epsilon = 3$, from each remaining cell, the agent has between 1 and 3 steps. If it moves it receives a penalty and 2/3 of the times bootstraps from the successor cell. Thus, $v(s) = \arg \max_a q(s; a)$ and $N(s)$ denoting the neighbors of s for states non adjacent to the goals we have $v(s) = 2 = 3(1 + \max_{s' \in N(s)} v(s')) + 1 = 3(1)$. 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 possibility to attempt to overtake by taking the “dangerous” action that has 80% probability to advance but poses 10% chance of collision with the oncoming traffic and terminating the episode. The agent receives a reward unless it reaches the destination, where the episode terminates with a reward of 1.

Figure 3: Comparison of PPO with and without the remaining time in input. Left: Performance on Reacher-v1 ($\gamma = 50$). Right: Performance on InvertedPendulum-v1 ($\gamma = 1000$) and the learned state-value estimations. Top: The results for $\gamma = 0.99$. Bottom: The result for $\gamma = 1$.

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 different time-unaware PPOs fail to do so. A discount factor of 1 was used for both agents.

2.4 Standard control tasks

In this section, we evaluate the performance of PPO with the remaining time as part of the agent's input on a set of continuous control tasks from the OpenAI's MuJoCo Gym benchmarks (Brockman et al., 2016; Duan et al., 2016). By default, these environments use predefined time limits.

Figure 3 shows the performance of a time-unaware PPO against a time-aware one on Reacher-v1 and InvertedPendulum-v1, demonstrating that time-awareness significantly improves the performance. The learned state values shown for the InvertedPendulum-v1 task illustrate perfectly 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.

Figure 4 (left) shows the performance comparisons of PPO with and without time-awareness in the Hopper-v1 domain with time limit $T = 300$. With a discount rate of 0.99, the standard PPO is initially on par with the time-aware PPO and later starts to plateau. As the agents become better, they start to experience terminations due to the time limit more frequently, at which point the time-unaware agent begins to perceive inconsistent returns for seemingly similar states. The advantage of the time-aware PPO becomes even clearer in the case of a discount rate where the time-unaware PPO diverges quite drastically. A possible reason is that the time-unaware PPO agent experiences much more significant conflicts as the returns are now the sum of the undiscounted rewards.

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 quite interesting ways of ensuring to achieve 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 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 proposed to include 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

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

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 in the Hopper environment, the learned policy that maximizes the return over time 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 of time. 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 propose a method that enables to effectively learn in such domains from partial episodes.

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 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 $G_{0:T}$ (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 propose to continue bootstrapping 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 $G_{t:T}$ and the bootstrapped value from the last state (S_T):

$$v(s) \doteq E[G_{t:T} + \gamma^{T-t} v(S_T) | S_t = s] \quad (5)$$

By denoting an estimate of the state-value function by v , the targets for the temporal-difference update rule after transitioning from a state s^0 to a state s^1 and receiving a reward r as a result of an action are:

$$r + \gamma v(s^1) \quad \text{at environmental termination} \\ r + \gamma v(s^0) \quad \text{otherwise (including when } t = T - 1) \quad (6)$$

The proposed partial-episode bootstrap is indicated in bold and green. An agent without this modification would update $v(s)$ with or without bootstrapping from the estimated values of $v(s^1)$ 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.

Figure 5: Performance evaluations of PPO with and without partial-episode bootstrapping. Left: Hopper-v1 with $T = 200$ during training and 10^6 during evaluations. Right: In niteCubePusher with $T = 50$ during training and 1000 during evaluations. PPO degrades drastically after some time.

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 partial 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 use PPO but with two key modifications. We modified the Gym's TimeLimit wrapper to not include the remaining time (as needed for Section 2), but instead to include a flag to differentiate the terminations due to the time limits only. We also modified the PPO's implementation to enable continuing to bootstrap when the flag is active. This involves changing the implementation of the generalized advantage estimation (GAE) (Schulman et al., 2016). While GAEs use an exponentially-weighted average of 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 proposed approach.

3.2 Hopper

Here, we consider the Hopper-v1 environment from Section 2.4, but instead aim to learn a policy that maximizes the agent's expected return over a time-unlimited horizon. We do not revisit Reacher-v1 and the InvertedPendulum-v1 environments as their extensions to time-unlimited domains is not of particular value—that is, staying at the target position long after the target is reached (Reacher-v1) or maintaining the pendulum's vertical pose long after it is balanced (InvertedPendulum-v1). 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) demonstrates performance evaluations of the standard PPO against one with the proposed partial-episode bootstrapping modification. The agents were trained on time-limited episodes of maximum 200 time steps, and were evaluated in the same environment, but with maximum time steps. The proposed bootstrapping method significantly outperforms the standard PPO. During the evaluations, PPO managed to reach a maximum of 28 time steps on only one of the 40 training seeds, while our agent managed to reach the evaluation time limit of 100 time steps on 7 occasions. This is quite impressive as it corresponds to more than 2 hours of rendered hopping (see [videos](#)).

3.3 The Indefinite Cube Pusher task

To demonstrate the ability of our agent to optimize for an indefinite horizon (no terminal state), we propose a novel MuJoCo environment 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

Due to the absence of reward shaping, it is necessary to limit episodes in time to diversify the experiences and learn to solve the task. Therefore, during training, a time limit of 50 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 (right) shows the performance comparison of the standard PPO against one with the proposed modification. An entropy coefficient of 0.03 was used to encourage exploration and increase the chance of reaching some targets. We found this value to yield best performance for both agents for 0.01; 0.05. While the performance of PPO degrades significantly after some time, our agent performs significantly better. The maximum number of targets reached by our agent in a single episode of evaluation gains 21 for PPO.

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 the time limit is so long that terminations due to time limits 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 environment that are correlated with the remaining time (e.g. the forward distance), if the relationship among the state-dependent action advantages remains preserved, 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 scenarios with 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 proposed approach is rather generic and can potentially be applied to domains with varying time limits. Finally, in real-world applications, such as robotics, the proposed approach could easily be adapted by using the real time instead of simulation time steps.

In order for the proposed partial-episode bootstrapping 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 the proposed partial-episode bootstrapping is quite generic in that it is not restricted to partial episodes caused only due to time limits. In fact, this approach is valid for any early termination causes. For instance, it is common in the curriculum learning literature to start from near the goal states (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 state that is already well known.

Since the proposed methods were shown to enable to better optimize for the time-limited and time-unlimited domains, we believe that they have the potential to improve the performance and stability of a large number of existing reinforcement learning algorithms. We also propose that, since reinforcement learning agents are in fact optimizing for the expected returns, and not the undiscounted sum of rewards, it is more appropriate to consider this measure for performance evaluation.

5 Conclusion

We considered the problem of learning optimal policies in time-limited and time-unlimited domains using time-limited interactions. We showed 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. Not doing so can cause state-aliasing which in turn can result in suboptimal policies, instability, and slower convergence. 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 environmental ones. In both cases, we illustrated that our proposed methods can significantly improve the performance of PPO and allow to optimize more directly, and accurately, for either of the optimality models.

Acknowledgments

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

References

- Kai Arulkumaran, Marc P Deisenroth, Miles Brundage, and Anil A Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.
- Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: an evaluation platform for general agents. *Journal of Artificial Intelligence Research (JAIR)*, 47:253–279, 2013.
- Dimitri P. Bertsekas and John Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
- Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.
- Yan Duan, Xi Chen, Rein Houthoofd, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *International Conference on Machine Learning (ICML)*, pp. 1329–1338, 2016.
- Carlos Florensa, David Held, Markus Wulfmeier, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*, 2017.
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- Daishi Harada. Reinforcement learning with time. *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 577–582. AAAI Press, 1997.
- Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.
- Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.
- Christopher Hesse, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. OpenAI Baselines. <https://github.com/openai/baselines>, 2017.
- Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research (JAIR)*, 4:237–285, 1996.
- Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. URL <http://dx.doi.org/10.1038/nature14539>.
- William S Lovejoy. 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>.
- Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the Arcade Learning Environment: evaluation protocols and open problems for general agents. *arXiv preprint arXiv:1709.06009*, 2017.
- Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. Wiley & Sons, 2014.
- Jürgen Schmidhuber. Deep learning in neural networks: an overview. *Neural Networks*, 61:85–117, 2015. URL <https://doi.org/10.1016/j.neunet.2014.09.003>.
- John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *International Conference on Learning Representations (ICLR)*, 2016.

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

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

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

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

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

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

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

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

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

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

SUPPLEMENTARY MATERIAL: TIME LIMITS IN REINFORCEMENT LEARNING

A All results for time-aware PPO

A.1 Queue of Cars

A.2 InvertedPendulum-v1

A.3 Reacher-v1

A.4 Hopper-v1

