Making Hyper-parameters of Proximal Policy Optimization Robust to Time Discretization

Homayoon Farrahi
Department of Computing Science
University of Alberta
Edmonton, AB T6G2E8
farrahi@ualberta.ca

A. Rupam Mahmood
Department of Computing Science
University of Alberta
Edmonton, AB T6G2E8
armahmood@ualberta.ca

Abstract

A small action cycle time can help reinforcement learning agents by granting them fast reaction and a more temporally detailed perception of the environment. The learning performance of both policy gradient and action value methods, however, deteriorates as the cycle time duration is reduced, which necessitates the tuning of the cycle time as a hyper-parameter. Since tuning an additional hyper-parameter is time-consuming when learning on real-world robots, existing algorithms can benefit from having hyper-parameters that are more robust to the choice of cycle time. In this work, we show the inefficacy of the default hyper-parameters of a policy gradient algorithm across different cycle times and propose a novel set of hyper-parameters. We investigate the effectiveness of the new hyper-parameters on a simulated task and validate them on simulated and real-world robotic tasks. Our experiments highlight the superiority of small cycle times over large ones and show that these novel hyper-parameters, unlike the default ones, are robust to different cycle times.

1 Introduction

Real-world robotic control tasks can benefit considerably from a small time discretization, allowing fast interaction between the agent and the environment. In reinforcement learning (Sutton & Barto 2018), the discrete time step, also known as the *action cycle time* δt, refers to the time elapsed in the environment between two consecutive actions of an agent. Choosing a smaller δt entails taking actions more frequently and having faster reactions to changes in the environment, which can improve the performance of reinforcement learning (RL) algorithms in many tasks and might even be crucial for others. Even if a task does not require fast reactions, an agent can still benefit from a small δt by observing the environment more frequently allowing it to observe important changes it might have otherwise missed with a large δt.

Unfortunately, using a small cycle time can hinder both action value and policy gradient methods from effective learning. Baird (1994) illustrated the ineffectiveness of *Q*-learning at small δt, as various actions in a state maintain similar action values, and its inability to learn in continuous time since the action value function becomes equal to the value function. Policy gradient methods, also, suffer from a small δt, as the variance of the policy gradient estimate, or that of the likelihood ratio specifically, grows inversely proportionally to δt, which was shown in a simple example by Munos (2006). These challenges make learning with small δts difficult, even though small δts can provide benefits as mentioned in the above.

Finding a time discretization that provides the right balance between task performance and learnability entails a search over several values of δt, which is time-consuming and costly for real-world robots. The effect of using different values of δt on the learning performance of a 2-dimensional reaching

NeurIPS 2020 3rd Robot Learning Workshop: Grounding Machine Learning Development in the Real World.

task on a real-world robotic arm was investigated by Mahmood et al. (2018a), who showed a δt in the middle of their chosen range to perform better than the smallest or the largest ones. The smallest δt afforded by the hardware is not always the best and the optimal δt might be different for different tasks and robots, necessitating its tuning. While the cost of tuning one additional δt hyper-parameter can be negligible in simulated environments, real-world robots can only collect experience in real-time, which makes the tuning time-consuming and potentially more costly.

Understanding how time discretization affects learning and knowing its relationship to other hyper-parameters may lead to a more efficient hyper-parameter tuning strategy. For instance, using a small δt means that a sample batch of the same size will be collected in less time and that future rewards are discounted more heavily for the same duration in real-time (Doya 2000). Having a set of guidelines and heuristics for adjusting the values of different algorithm hyper-parameters upon changing δt can reduce tuning time and cost substantially, and lead to better task performance.

In this paper, we demonstrate the ineffectiveness of the default hyper-parameters of Proximal Policy Optimization (PPO), a popular policy gradient algorithm (Schulman et al. 2017), when learning at δts other than the simulated environment's default. We then propose a set of modifications, based on δt, to these hyper-parameters to improve upon the performance of the default values and show the advantage of small δts over large ones in achieving superior performance. Lastly, we validate our proposed set of modifications on a simulated and a real-world robotic task.

2 Related works

Reducing δt can inhibit effective learning in RL algorithms. Baird (1994) demonstrated the inefficacy of the action value function in continuous time and proposed to use the advantage function instead. Tallec et al. (2019) extended this work to deep *Q*-learning methods. The continuous-time version of the Bellman equation, Hamilton-Jacobi-Bellman (HJB) equation, has been used by Munos and Bourgine (1998) to develop a model-based algorithm, and Doya (2000) who provided methods for learning the value function and extended the actor-critic method to the continuous-time case. Munos (2006) showed that the variance of the policy gradient estimates that use the likelihood ratio can explode as δt goes toward 0. They formulated an alternative policy gradient estimate, which is difficult to apply to more challenging tasks due to their assumptions about the environment.

Previous works (Mahmood et al. 2018a, Dulac-Arnold et al. 2020) have mentioned the issues that can arise in real-world robotic tasks due to the delays between observations and actions, and efforts have been made to alleviate them (Travnik et al. 2018, Ramstedt and Pal 2019). In this paper, we focus on the time discretization issues and consider that the chosen action cycle times always fit the forward pass for action calculations. Although Dulac-Arnold et al. (2020) showed declining task performance with increasing δt, we show that reducing δt can negatively affect performance as well.

3 Failure of the default hyper-parameters at different time discretizations

In this section, we investigate the robustness of the default hyper-parameters of the algorithm PPO to different δts. We modify the PyBullet (Coumans & Bai 2016) environment *ReacherBulletEnv-v0* by changing its *environment time step* from the default 16.5 ms to a constant 2 ms for all experiments and implementing different δts by making the agent interact with the environment at multiples of 2 ms. To run an experiment with δt of 8 ms, for instance, the agent interacts with the environment every fourth environment step by taking the most recent observation as input and outputting an action, which is repeatedly applied to the environment until the next interaction. The reward for each action is a summation over all individual rewards received every environment step of 2 ms. Each episode lasts for 2.4 simulation seconds. We also slightly modify the environment's state construction to dispense with duplicate states and to make the *stuck joint cost* component of the reward more robust to the physics engine's behaviour. The agent's architecture comprises a two hidden layer neural network with $\tanh$ activations and an independent parameter producing the mean μ and $\log(\sigma)$, respectively, of a normal distribution $\mathcal{N}(\mu, \sigma^2)$ from which the policy π samples its actions.

To study the robustness of algorithm hyper-parameters to different δts, we ran PPO with the default hyper-parameters using different δts and stored the undiscounted episodic returns. The learning curve for each δt was plotted in Figure 1. We further experimented with different values of the batch size and mini-batch size to see if any improvements at all could be made to the learning

performance of different δts. The overall average return for each run was calculated for a set of tuned hyper-parameters for each δt and plotted against δt in Figure 2. All plots were averaged over 30 independent runs each lasting 10 million environment steps.

Our experimental results indicate that the default hyper-parameters can lead to suboptimal performance when changing the time discretization, and that performance can be recovered, at least partially, by using a different set of hyper-parameters. Figure 1 shows that the asymptotic performance declines when using small δts and that large δts hurt the learning speed. We show in Figure 2 that adjusting hyper-parameters for each δt can lead to increased learning performance. This demonstrates the importance of having guidelines for adjusting different hyper-parameters based on δt, to avoid time-consuming hyper-parameter tuning on real-world robots.

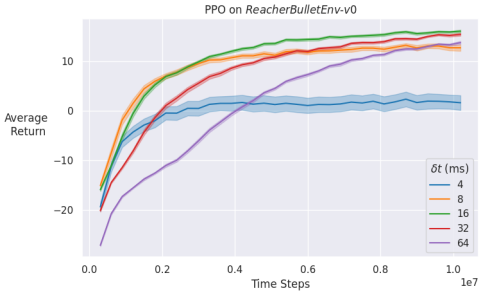

Figure 1: Learning curves for different δt with default PPO hyper-parameters. Smaller δts have worse asymptotic performance while larger δts learn more slowly.

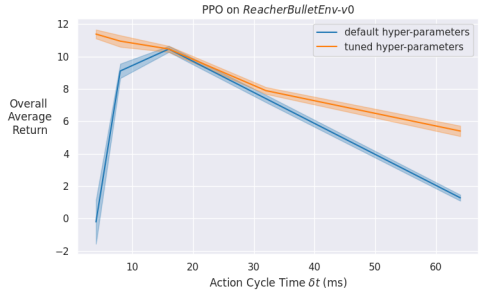

Figure 2: Overall average return vs. δt for different hyper-parameter configurations. Tuned hyper-parameters can improve the learning performance for the majority of δt values.

4 Setting hyper-parameters as a function of the action cycle time

Changing the time discretization affects the relationship between time, and the hyper-parameters batch size b and mini-batch size m. As δt decreases, each fixed-size batch or mini-batch of samples contains less amount of real-time experience. We make the real-time experience content consistent among different δts by scaling the default b and m inversely proportionally to δt, shown in (1) as $b_{\delta t}$ and $m_{\delta t}$. The scaled $b_{\delta t}$ and $m_{\delta t}$ represent the batch size and mini-batch size used when δt is the action cycle time. For instance, reducing δt from the default $\delta t_0 = 16$ ms to 8 ms, makes the batch size b_8 double the size of b_{16}. This keeps the batch time $\delta t \cdot b_{\delta t}$, the time in seconds it takes to collect a batch, consistent between different δts.

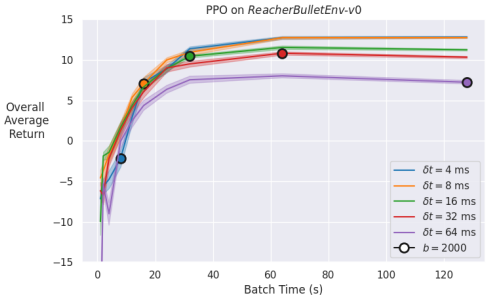

Figure 3: Learning performance using the δt-aware hyper-parameters. Small δts are better than or equal to large δts at all batch times. The circles mark the performance for the default b.

The time discretization affects the behaviour of the discount factor γ and the trace-decay parameter λ, as well, by changing the rate at which rewards and n-step returns are discounted through time (Doya 2000, Tallec et al. 2019). When using a small δt, γ and λ decay faster, as more experience samples are collected in a fixed time interval compared to larger δts. We propose to exponentiate γ and λ to the δt power only for δts larger than the default δt_0 since doing so leads to decreased performance for smaller δts. Appendix A.1 lists the default hyper-parameter values for $\delta t_0 = 16$ ms. The new *δt-aware hyper-parameters* $b_{\delta t}$, $m_{\delta t}$, $\gamma_{\delta t}$ and $\lambda_{\delta t}$, shown in (1), are calculated as:

$$b_{\delta t} = \frac{\delta t_0}{\delta t} b_{\delta t_0}, \quad m_{\delta t} = \frac{\delta t_0}{\delta t} m_{\delta t_0}, \quad \gamma_{\delta t} = \min\left(\gamma_{\delta t_0}, \gamma_{\delta t_0}^{\delta t / \delta t_0}\right), \quad \lambda_{\delta t} = \min\left(\lambda_{\delta t_0}, \lambda_{\delta t_0}^{\delta t / \delta t_0}\right). \quad (1)$$

We evaluated the δt-aware hyper-parameters by running PPO on the environment from the previous section using different batch times and time discretizations each for 10 million environment steps. The overall average return for each learning curve was calculated, averaged over 30 runs, and plotted against the batch times in Figure 3.

Figure 3 illustrates that the δt-aware hyper-parameters make the performance of small δts better than or equal to that of large δts at all batch times and thus showcases the superiority of small δts to larger ones. We also experimented with the δt-aware hyper-parameters but using $\gamma_{\delta t} = \gamma_{\delta t_0}^{\delta t / \delta t_0}$ and $\lambda_{\delta t} = \lambda_{\delta t_0}^{\delta t / \delta t_0}$ instead, which lead to diminished performance for smaller δts at smaller batch times. When using (1) but keeping γ_{16} and λ_{16} constant across all δts, we observed decreased performance for larger δts. The figures for these two modifications are available in Appendix A.4.

5 Validating the new δt-aware hyper-parameters

We ran PPO with the default and the δt-aware hyper-parameters each on a simulated and on a real-world robotic task to validate our proposed set of hyper-parameters. We reduced the environment time step of another PyBullet environment *InvertedDoublePendulumBulletEnv-v0* from 16.5 ms to a constant 4 ms and simulated other δts as integer multiples of 4 ms. We ran the experiments once by keeping the default hyper-parameters of Appendix A.2 ($b_{16} = 4000$) constant across δts, and another time using the δt-aware hyper-parameters with $\delta t_0 = 16$ ms and $b_{16} = 4000$, each for 10 million environment steps. The overall average return was aggregated over 30 runs and plotted against δt in Figure 4. Episodes of this environment lasted for a maximum of 16 simulation seconds.

For the real-world task, we adopted *UR-Reacher-2* developed by Mahmood et al. (2018b) and enlarged the movement boundary to reduce the number of scripted position corrections at the boundary. We set the environment time step to 10 ms and ran three sets of experiments, one with $\delta t = 40$ ms as the benchmark, and two with $\delta t = 10$ ms to compare the default hyper-parameters of Appendix A.3 ($b_{40} = 2000$), kept constant across δts, to the δt-aware hyper-parameters using $\delta t_0 = 40$ ms and $b_{40} = 2000$. All episodes were 4 seconds long. The runs lasted for 600000 environment steps, and the learning curves, averaged over 5 runs, were plotted in Figure 5. All rewards were scaled by δt, and other experimental details were as in the previous section.

On the simulated task, the δt-aware hyper-parameters, compared to the default set, improves the learning performance and makes it more robust to the choice of δt as seen in Figure 4. Figure 5 shows that the δt-aware hyper-parameters recover the asymptotic performance that is lost by using the default hyper-parameters with a small δt. We believe more runs can help reduce the uncertainty in the performance of the default hyper-parameters with $\delta t = 10$ ms.

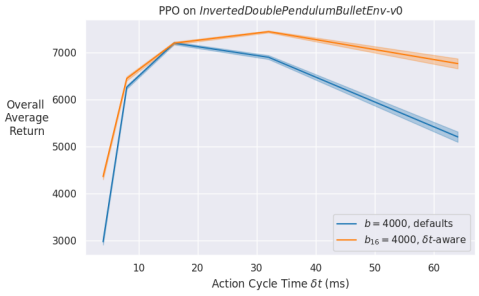

Figure 4: Learning performance of different δts on a simulated validation task. The δt-aware hyper-parameters make the performance more robust to the choice of δt.

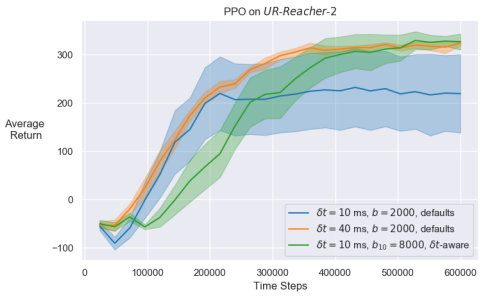

Figure 5: Learning curves of the UR5 robot comparing the δt-aware hyper-parameters to the default ones. Asymptotic performance is recovered by the δt-aware hyper-parameters.

6 Conclusion

We demonstrated the sensitivity of PPO's performance to the choice of δt when using the default hyper-parameters, proposed a replacement set of δt-aware hyper-parameters that scale according to δt, and empirically showed that they improve the robustness of the performance to δt. The δt-aware hyper-parameters showcased the superiority of small δts to large ones and can lessen the need for extensive hyper-parameter tuning, which is time-consuming and costly on real-world robots. We finally showed the advantage of δt-aware hyper-parameters over the default ones by validating them on simulated and real-world robotic tasks.

Acknowledgments

This work was sponsored by the Reinforcement Learning and Artificial Intelligence (RLAI) Lab, Alberta Machine Intelligence Institute (Amii), and Canada CIFAR AI Chairs Program.

References

Baird, L. C. (1994). Reinforcement learning in continuous time: advantage updating. In *Proceedings of 1994 IEEE International Conference on Neural Networks*.

Coumans, E., Bai, Y. (2016). PyBullet, a python module for physics simulation for games, robotics and machine learning. URL `http://pybullet.org`

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

Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Gowal, S., Hester, T. (2020). An empirical investigation of the challenges of real-world reinforcement learning. *arXiv preprint arXiv:2003.11881*.

Mahmood, A. R., Korenkevych, D., Komer, B. J., Bergstra, J. (2018a). Setting up a reinforcement learning task with a real-world robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., Bergstra, J. (2018b). Benchmarking reinforcement learning algorithms on real-world robots. In *Proceedings of the 2nd Annual Conference on Robot Learning*.

Munos, R. (2006). Policy gradient in continuous time. *Journal of Machine Learning Research 7* (May): 771-791.

Munos, R., Bourgine, P. (1998). Reinforcement learning for continuous stochastic control problems. In *Advances in Neural Information Processing Systems*.

Ramstedt, S., Pal, C. (2019). Real-time reinforcement learning. In *Advances in Neural Information Processing Systems*.

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

Sutton, R. S., Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.

Tallec, C., Blier, L., Ollivier, Y. (2019). Making deep Q-learning methods robust to time discretization. *arXiv preprint arXiv:1901.09732*.

Travnik, J. B., Mathewson, K. W., Sutton, R. S., Pilarski, P. M. (2018). Reactive reinforcement learning in asynchronous environments. *Frontiers in Robotics and AI 5* (79).

Appendices

A Default hyper-parameters

A.1 *ReacherBulletEnv-v0*

hyper-parameter	value
b_{16}	2000
m_{16}	50
γ_{16}	0.99
λ_{16}	0.95

A.2 *InvertedDoublePendulumBulletEnv-v0*

hyper-parameter	value
b_{16}	4000
m_{16}	100
γ_{16}	0.99
λ_{16}	0.95

A.3 *UR-Reacher-2*

hyper-parameter	value
b_{40}	2000
m_{40}	50
γ_{40}	0.99
λ_{40}	0.95

A.4 Different approaches for scaling γ and λ

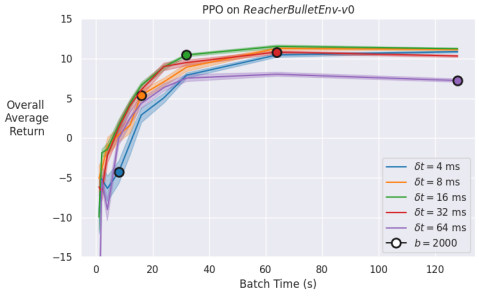

Figure 6: Learning performance of δt-aware hyper-parameters, except, γ and λ are always exponentiated to the δt power. The circles mark the performance for the default b.

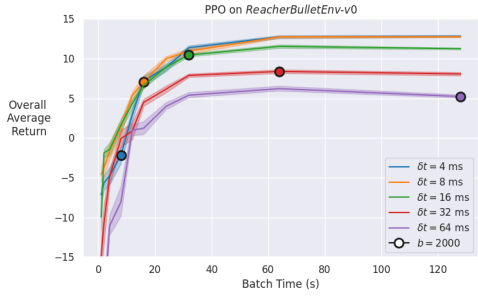

Figure 7: Learning performance of δt-aware hyper-parameters, except, γ_{16} and λ_{16} are constant in all runs. The circles mark the performance for the default b.