Mitigating the Curse of Horizon in Monte-Carlo Returns

Alex Ayoub[1,2*], **David Szepesvari**[1,2*], **Francesco Zanini**[1,2*], **Bryan Chan**[1,2*], **Dhawal Gupta**[3], **Bruno Castro da Silva**[3], **Dale Schuurmans**[1,2,4]
{aayoub,david.szepesvari,fzanini,bryan.chan,daes}@ualberta.ca,
{dgupta,bsilva}@cs.umass.edu
[1]University of Alberta [2]Alberta Machine Intelligence Institute
[3]University of Massachusetts [4]Google DeepMind

Abstract

The standard framework in reinforcement learning (RL) dictates that an agent should use every observation collected from interactions with the environment when updating its value estimates. As this sequence of observations becomes longer, the agent is afflicted with the curse of horizon since the computational cost of its updates scales linearly with the length of the sequence. In this paper, we propose methods to mitigate this curse and improve sample efficiency for continuous-time value estimation with Monte-Carlo methods. This is accomplished by selecting a subsequence of observations on which the value estimates are computed. We empirically demonstrate on standard RL benchmarks that adopting an adaptive sampling scheme outperforms the default uniform sampling procedure.

1 Introduction

The reinforcement learning (RL) framework is typically modeled as a Markov Decision Process in discrete-time intervals (Sutton & Barto, 2018), in which the interaction between agent and environment evolves at a predetermined time interval, fixed a priori. There are two evident shortcomings in carelessly adopting the latter model: first, the discrete-time nature of the process might not capture the nature of many real-world dynamics, in which treating time as a continuous variable can be more appropriate and advantageous; second, the fixed-time assumption is taking away a degree of freedom from the framework.

The second issue is particularly interesting in RL as the greater freedom can be exploited to design algorithms with better performance. Note that usually the system is assumed to have a fixed discretization step, and current algorithms make use of all the samples they are given. It has been recently established by Zhang et al. (2024) that using only a subset of the trajectory tuples can yield more sample-efficient learning when performing Monte-Carlo value estimation. This paper goes beyond uniform temporal discretization schemes and considers a procedure that makes use of an adaptive integration scheme taken from the numerical integration literature. Such an adaptive scheme does not fix a uniform sampling time, but instead adapts the sampling time to balance the error in approximating the integral of a single trajectory with collecting several trajectories to reduce variance from the system's inherent stochasticity. Therefore the question this work aims to investigate is:

Can non-uniform discretization be leveraged to improve sample efficiency in continuous-time value estimation?

With uniform sampling, employing a finer discretization leads to a better approximation of the continuous-time system from discrete measurements; however, considering a fixed sample budget,

[*]These authors contributed equally to this work.

more samples within each trajectory results in fewer collected trajectories, leading to an increase in estimation error due to system stochasticity. Conversely, a coarser discretization leads to more collected trajectories at the expense of a worse approximation of the continuous-time system.

With an adaptive scheme, we can employ a finer discretization in the volatile regions of the continuous-time system and employ a coarser discretization in the smoother regions of the system. Thus fewer measurements of the system can be made to achieve a comparable approximation error to a uniform sampling scheme, which allocates the same discretization to all regions of the system. This, in turn, enables the adaptive scheme to allocate more of its sample budget to collecting trajectories to reduce its estimation error.

Our main contribution is demonstrating that an adaptive algorithm achieves better policy evaluation than previous uniform discretization methods. This is done through empirically evaluating both the adaptive and uniform methods in standard RL benchmarks such as the Gymnasium Classic Control suite (Towers et al., 2023) and MuJoCo (Todorov et al., 2012) environments. The impact of this work is evident as using significantly less samples enhances training efficiency thereby accelerating the learning process.

2 Problem Setting

The continuous-time reinforcement learning (RL) problem is modeled as a Markov Decision Process (MDP), which is characterised by the tuple $(\mathcal{S}, \mathcal{A}, f, r, \gamma, \eta)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, f specifies the state evolution dynamics, r the reward function, γ is the discount factor, and η represents the initial state distribution. Through this model, an agent and the environment interact over time, denoted as t. This interaction is initialized by sampling a state from the initial state distribution $\eta \in \mathcal{M}_1(\mathcal{S})$, where $\mathcal{M}_1(X)$ denotes the set of distribution supported on X, for any measurable set X. Given the current state $s(t) \in \mathcal{S}$ and current action $a(t) \in \mathcal{A}$, the environment transitions to a new state. This transition can be described through a differential equation as

$$\frac{\mathrm{d}\, s(t)}{\mathrm{d}t} = f(s(t), a(t)) \tag{1}$$

where $f(s(t), a(t))$ represents the dynamics of the environment. In what follows, we will consider deterministic dynamics as is common in the continuous-time RL literature (Doya, 2000; Yildiz et al., 2021). The behavior of the agent also affects the reward, which is a scalar-valued function given by

$$r(t) = r(s(t), a(t)). \tag{2}$$

The reward is an immediate measure the agent's performance. Given a fixed horizon T, the return, or cumulative reward, is defined as

$$\begin{aligned} G_T &= \int_0^T \gamma^t r(s(t), a(t))\, dt, \\ \text{subject to} \quad & \frac{\mathrm{d}\, s(t)}{\mathrm{d}t} = f(s(t), a(t)), \end{aligned} \tag{3}$$

where without loss of generality we considered the initial time to be $t_0 = 0$. The return is a random variable since it is subject to the variability of the initial state, which is sampled i.i.d. from η in each trajectory. In RL, the agent's objective is to find a policy $a(t) = \pi(s(t))$ that maximizes the expected return, also known as the value function, given by:

$$v_T^\pi(\eta) = \mathbb{E}_\pi \left[G_T | s(0) \sim \eta \right]. \tag{4}$$

Maximizing $v_T^\pi(\eta)$ is accomplished by learning an optimal policy, defined as $\pi^\star \in \arg\max_\pi v_T^\pi(\eta)$. An important step towards this goal is to estimate the value function for a fixed policy, which correspond to the task of policy evaluation.

2.1 Monte-Carlo Policy Evaluation

The main focus of this work is *policy evaluation*, which aims to estimate the expected return from discrete-time observations. Given N samples $\{(s(t_n), a(t_n), r(t_n))\}_{n=0}^{N-1}$, observed at discrete-time points $0 = t_0 < t_1 < \cdots < t_{N-1} = T$, we consider the First-Order-Hold (FOH) scheme (Franklin et al., 1997) to approximate the reward function, which yields a piecewise linear function:

$$\hat{r}\,(t) = r\,(t_{n-1}) + \frac{r\,(t_n) - r\,(t_{n-1})}{t_n - t_{n-1}}\,(t - t_{n-1})\,, \tag{5}$$

$\forall t \in [t_{n-1}, t_n]$. By defining

$$\overline{r}_n = \frac{\gamma^{t_n} r(t_n) + \gamma^{t_{n-1}} r(t_{n-1})}{2}\,(t_n - t_{n-1}) \tag{6}$$

we can approximate the integral in Equation (3) by the discrete-time return

$$\hat{G} = \sum_{n=1}^{N-1} \overline{r}_n\,, \tag{7}$$

which amounts to using the trapezoidal rule for numerical integration. To estimate $v_T^{\pi}(\eta)$, we average M independent trajectories with return estimates $\hat{G}_1, \hat{G}_2, \ldots, \hat{G}_M$ to obtain the Monte-Carlo estimator

$$\hat{V}_M = \frac{1}{M} \sum_{m=1}^{M} \hat{G}_m, \tag{8}$$

where the initial states are sampled independently from the initial state distribution η.

The main goal of this work is to empirically study the relationship between the number of samples used, i.e., the budget $B = M \cdot N$, in constructing the Monte-Carlo estimator $\hat{V}_M$ and the absolute error of our estimator under various discretization schemes. When considering the uniform discretization scheme, only one degree of freedom remains in choosing the discretization points $t_0, t_1, \ldots, t_{N-1}$ for a given trajectory $m \in \{1, \ldots, M\}$, for a fixed budget B. If a coarse discretization is used (N small), then more of the *sample budget* can be allocated to collecting trajectories. Conversely, if a fine discretization is used (N large), then less of the budget is allocated to collecting trajectories. Thus the choice of N controls the approximation error due to discretization and the statistical error due to stochasticity through $M = B/N$. Note that in a deterministic setting, statistical errors are inherently nonexistent, therefore the optimal strategy entails allocating the entire budget to a single trajectory.

3 Algorithms

In the following, we detail both an *adaptive* and *uniform* method for approximating the integral in Equation (3) from which our Monte-Carlo estimator $\hat{V}_M$ is constructed. The adaptive method is detailed in Algorithm 1 while the uniform method is detailed in Algorithm 2. To the best of our knowledge, our work is the first to propose and study such an adaptive method for Monte-Carlo policy evaluation in RL.

3.1 Adaptive Integration

In Algorithm 1, we detail an adaptive integration scheme, often referred to as *adaptive quadrature* in the numerical integration literature, that uses a finer discretization in the parts of the domain of integration where it is harder to get good accuracy and a coarser discretization in the parts of the domain of integration where it is easy to get good accuracy.

The basis for adaptive integration schemes is the additive property of definite integrals, namely that for any $c \in [a, b]$ it holds that $\int_a^b f(x)dx = \int_a^c f(x)dx + \int_c^b f(x)dx$. Thus if the two integrals on

Algorithm 1 ADAPTIVE

To approximate $\int_{\tau_1}^{\tau_2} r(t)dt$ within tolerance ε.
Input: The rewards r, the limits of integration τ_1 and τ_2, and the tolerance ε
$\tau_3 = \frac{\tau_1+\tau_2}{2}$
$Q_{\tau_i,\tau_j} = \frac{\gamma^{\tau_i} r(\tau_i)+\gamma^{\tau_j} r(\tau_j)}{2}(\tau_j - \tau_i)$ for $(i,j) = \{(1,2),(1,3),(3,2)\}$.
if $|Q_{\tau_1,\tau_2} - Q_{\tau_1,\tau_3} - Q_{\tau_3,\tau_2}| > \varepsilon$ **then**
    $Q = \text{ADAPTIVE}(r, \tau_1, \tau_3, \varepsilon/2) + \text{ADAPTIVE}(r, \tau_3, \tau_2, \varepsilon/2)$
else
    $Q = Q_{\tau_1,\tau_2}$
end if
return Q

the right can be approximated within an arbitrary tolerance ε, then their sum gives an approximation of the desired integral which is within 2ε. Otherwise, we can recursively apply the additive property on the sub-intervals $[a, c]$ and $[c, b]$. Note the each subdivision of the intervals results in the tolerance getting halved. Thus, we can expect this adaptive integration scheme to adapt to the integrand "automatically", partitioning the interval into sub-intervals with fine discretization where the integrand changes rapidly and coarse discretization where the integrand changes slowly.

Algorithm 1 is implemented with the trapezoidal integration rule and a Newton-Cotes quadrature rule, i.e., equally spaced points. The algorithm begins by applying a Newton-Cotes quadrature rule to compute the sub-interval, i.e. $\tau_3 = (\tau_1 + \tau_2)/2$. Then the trapezoidal rule is exploited to approximate the integral on the intervals $[\tau_1, \tau_2]$, $[\tau_1, \tau_3]$ and $[\tau_3, \tau_2]$. If the absolute difference between the approximation on the original interval, $[\tau_1, \tau_2]$, and the approximations on the sub-intervals, $[\tau_1, \tau_3]$ and $[\tau_3, \tau_2]$, is larger than the prescribed tolerance, then the procedure is run recursively on the sub-intervals. Otherwise, the method accepts the approximation and it is added to the running sum.

Note that this method can also be applied with different integration rules, such as Simpson's 3/8 rule, and different quadrature rules, such as Gaussian quadrature. Our method is presented with the trapezoidal rule and Newton-Cotes rule for simplicity of exposition. It is often the case in adaptive integration that other combinations of integration and quadrature rules significantly outperform the combination used in Algorithm 1. For a more detailed discussion of the different adaptive integration schemes, we refer the reader to (Davis & Rabinowitz, 2007; Gonnet, 2012).

3.2 Uniform Integration

Algorithm 2 details the trapezoidal rule for approximation a definite integral. Using N uniformly spaced points, Algorithm 2 computes the trapezoidal rule on each of the uniformly spaced sub-intervals $a = t_0 < t_0 + h < t_0 + 2h < \cdots < t_{N-1} = b$ where $h = (b-a)/(N-1)$. The method then sums all the approximations on the sub-intervals and returns the approximation of the integral. As the resolution of the partition increases, the approximation becomes more accurate.

As with adaptive integration schemes, both the spacing of the points, i.e., the quadrature rule, and the integration rule can be varied to achieve better approximations with fewer points. However, unlike adaptive integration schemes, uniform schemes must use all the points pre-specified by the user when performing numerical integration and does not attempt to adapt to the integrand.

4 Properties of Trapezoidal Integration Schemes

In this section, we provide standard error analysis of the numerical integration schemes detailed in Algorithms 1 and 2. Put simply, the following results detail the amount of points needed to produce a good approximation of the integral and allow for a better insight into the behavior of the two

Algorithm 2 UNIFORM

To approximate $\int_a^b r(t)dt$ with uniformly spaced points.
Input: The rewards r, the number of points N.
$h = \frac{b-a}{N-1}$
$Q = h \cdot \frac{\gamma^{t_1} r(t_1) + \gamma^{t_2} r(t_2)}{2}$
for $i = 0, \ldots, N-1$ **do**
  $t_i = a + ih$
  $Q = Q + h \cdot \gamma^{t_i} r(t_i)$
end for
return Q

different sampling schemes. The next proposition gives a uniform bound on the approximation of the uniform sampling scheme, for sufficiently smooth functions.

Proposition 4.1. *Let $\mathcal{F}$ be the class of twice differentiable functions such that for all $x \in [a,b]$ and $f \in \mathcal{F}$ it holds that $|f''(x)| \leq C$. For any $\varepsilon > 0$ there exists $f \in \mathcal{F}$ such that Algorithm 2 returns an approximation Q of the integral $\int_a^b f(x)dx$ whose absolute error is no greater than ε, i.e., $|Q - \int_a^b f(x)dx| \leq \varepsilon$, only if $N \geq 1 + (b-a)\sqrt{C/(12\varepsilon)}$.*

Proof. From Equation 5.21 of Sauer (2006) we have that

$$\int_a^b f(x)dx = \frac{b-a}{2}\left(f(a) + f(b)\right) - \frac{(b-a)^3}{12} f''(c), \tag{9}$$

where $c \in [a,b]$. For a natural number N, define $h = (b-a)/(N-1)$, and let $a = x_0$, $x_i = x_0 + ih$ for $i \in \{1, \ldots, N-1\}$. Then the additive property of integrals gives us

$$\int_a^b f(x) = \int_{x_0}^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx + \cdots + \int_{x_{N-2}}^{x_{N-1}} f(x)dx\,. \tag{10}$$

Plugging in Equation (9) for each integral on the right hand side, we get

$$\int_a^b f(x) = h\left(\frac{f(x_{N-1}) + f(x_0)}{2} + \sum_{i=1}^{N-2} f(x_i)\right) + \frac{h^3}{12}\sum_{i=1}^{N-1} f''(c_i), \tag{11}$$

where $c_i \in [x_{i-1}, x_i]$. Rearranging and taking the absolute value yields

$$\left|\int_a^b f(x) - \underbrace{h\left(\frac{f(x_{N-1}) + f(x_0)}{2} + \sum_{i=1}^{N-2} f(x_i)\right)}_{=Q}\right| = \left|\frac{h^3}{12}\sum_{i=1}^{N-1} f''(c_i)\right| \tag{12}$$

$$\leq \left|\frac{(N-1)Ch^3}{12}\right| \tag{13}$$

where the inequality uses $f''(c) \leq C$ for all $c \in [a,b]$. Equality in (13) is attained by $f_0 \in \mathcal{F}$ with $f''(c_i) = C\ \forall i$. Substituting in the definition of h we have that

$$\frac{(N-1)Ch^3}{12} = \frac{(b-a)^2 C}{12(N-1)^2}\,. \tag{14}$$

Therefore if $N \geq 1 + (b-a)\sqrt{C/(12\varepsilon)}$, we have that

$$\left|\int_a^b f(x) - Q\right| \leq \varepsilon\,, \tag{15}$$

and that f_0 requires this many points to be approximated sufficiently precisely. □

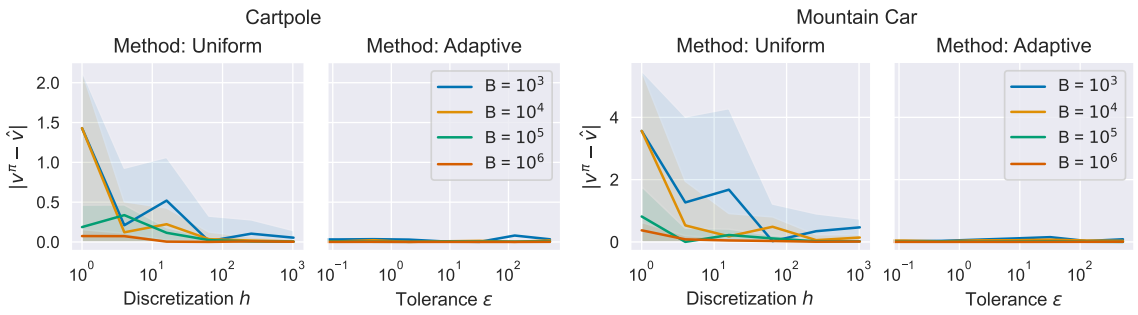

Figure 1: The absolute error in value estimates returned by the uniform and adaptive methods in Cartpole (left) and MountainCar (right). Each colored curve corresponds to a fixed budget, and displays the relationship between the hyperparameter of the method and the experienced error. The adaptive method experiences near zero error in these environments for all tolerances and even the smallest budget of 1000 samples.

The following corollary immediately follows from Proposition 4.1 and gives a worst-case guarantee on the amount of times Algorithm 1 calls itself to produce a good approximation of the integral of interest.

Corollary 4.2. *Under the conditions of Proposition 4.1, there exists a function $f \in \mathcal{F}$ such that Algorithm 1 needs to call itself at least $1 + (b-a)\sqrt{C/(12\varepsilon)}$ times to return an approximation Q such that $|Q - \int_a^b f(x)dx| \leq \varepsilon$.*

In order to show this, it suffices to observe that in the worst-case Algorithm 1 needs to consider at least N uniformly spaced points in the interval $[a, b]$.

5 Numerical Experiments

In order to understand the performance of the different integration schemes presented in Section 3, the adaptive and uniform methods are evaluated on several standard RL benchmarks. These include a selection of the environments from the Gymnasium Classic Control suite (Towers et al., 2023), which provides low-dimensional control tasks, and from the MuJoCo physics-based simulation environments (Todorov et al., 2012), offering more complex dynamics. We adopt the Monte-Carlo estimator described in Equation (8). Since all of our experiments are finite horizon, we set $\gamma = 1$. Our findings demonstrate that the adaptive method generally performs better than the uniform method without an extensive hyperparameter search.

5.1 Classic Control Environments

We evaluate well-performing policies in the MountainCar (Moore, 1990) and Cartpole (Barto et al., 1983) environments in Gymnasium. In MountainCar, the agent receives the same negative reward in every time step until it either reaches the top of the mountain and the episode terminates or it runs out of time. The policy we evaluate reaches the goal from the vast majority of start states, though the number of steps this takes and therefore the total reward depends on the start state. In Cartpole, the agent receives a reward of $+1$ for each time step the pole is balanced, and once the pole falls the episode ends. Since a good policy can balance the pole forever, the policy we evaluate artificially drops the pole after some time. The random start state determines the exact time to fall and therefore the total reward. For both environments, we sample the start states uniformly from a set of 100 unique states. To approximate continuous-time interaction, we collect data at 100 times the base frequency, which leads to long sequences of observations.

We estimate the value of these policies using both the adaptive and uniform methods. We consider tolerances $\varepsilon \in \{0, 2^{-3}, 2^{-1}, 2^1, \ldots, 2^9\}$ for the adaptive method and discretizations $h \in$

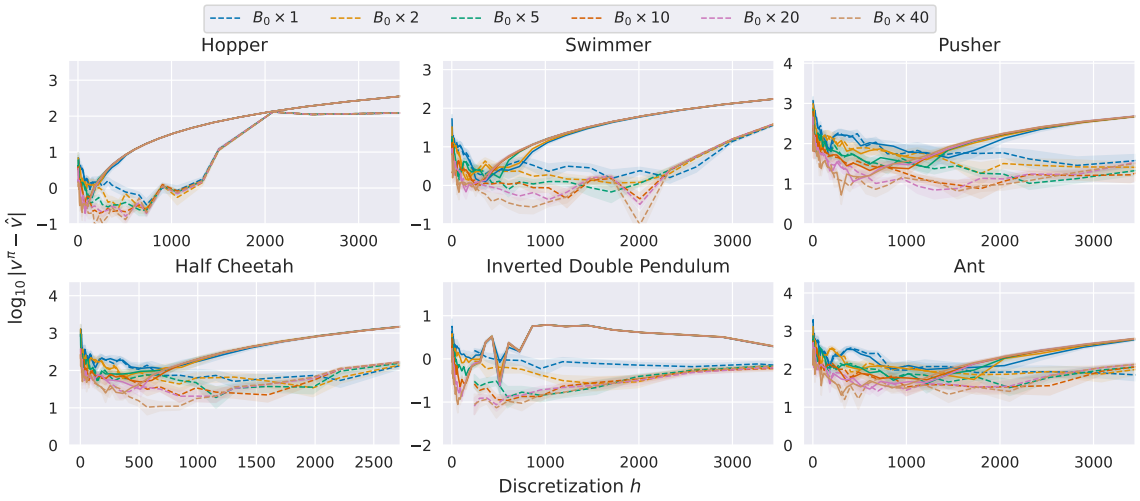

Figure 2: The absolute value error, in log scale, based on discretization h. Each colored curve represents a fixed budget $B = B_0 \times K$, where $K = \{1, 2, 5, 10, 20, 40\}$, used for estimating the value function v^π. Solid curves correspond to the uniform method and dashed curves correspond to the adaptive method. Note that in some cases the lowest tolerance considered $\varepsilon = 0.125$ is not mapped through (16) to the finest discretization of $h = 2$, i.e., $\hat{h}(\varepsilon) > 2$. The adaptive method is generally better than the uniform method on larger h.

$\{1, 4, 16, \ldots, 1024\}$ for the uniform method. We run 100 trials for each hyperparameter. Figure 1 plots the absolute error in the value estimates for each of these settings.

The adaptive method, irrespective of the tolerance parameter, successfully exploits the unique structure of the reward in these environments and therefore is able to achieve a low error with a limited sample budget. The reward is constant in these environments, hence an approximation based on just the first and last points recovers the correct integral. The adaptive method discovers this in the first iteration and hence focuses its budget on reducing the statistical error by sampling many trajectories. The uniform method also gets zero approximation error but must use all the points pre-specified by the user. This leaves less of the budget for reducing statistical error. As the average number of points per trajectory grows with finer discretization parameters, fewer trajectories can be observed which given a fixed budget leads to larger statistical error.

5.2 MuJoCo Environments

In our evaluation of MuJoCo environments, a policy is trained for each task using the Deep Advantage Updating (DAU) (Tallec et al., 2019) algorithm. DAU can be seen as the deep version of Advantage Updating (Baird, 1994) and it is specifically designed to perform well on problems with high-frequency observations. The policy exhibits stable trajectories, ensuring for instance that the inverted double pendulum remains predominantly upright or the ant moving forward, without inducing premature termination of the episodes. The data is collected at a fine rate of $\delta t = 0.001$ as a proxy for continuous-time behavior.

The reward function has a high-frequency component since it is directly influenced by rapidly changing actions. This violates the operating conditions for both methods to be effective, Proposition 4.1 and Corollary 4.2, thus a smoothing function has been applied to the rewards. Consequently, both methods take as input the smoothed version of the rewards for the different trajectories. In particular, for each trajectory, we fit a 7 degree polynomial function to the rewards and obtain the smoothed rewards using the fitted polynomial.

We sweep over tolerances $\varepsilon \in [0.125, 4096]$ for the adaptive method and discretizations $h \in [2, 3444]$ for the uniform method. We run 10 trials for each hyperparameter—we note that the only source the randomness comes from the uniform sampling of a set of 10000 unique initial states.

To effectively compare both methods, we map the tolerance ε in the adaptive method onto the average discretization:

$$\hat{h}(\varepsilon) = \frac{1}{M} \sum_{m=1}^{M} \left(\frac{N_{\text{fg}}}{N_m(\varepsilon) - 1} \right) \approx \mathbb{E}_Z \left[\frac{N_{\text{fg}}}{N_Z(\varepsilon) - 1} \right], \tag{16}$$

where $N_{\text{fg}} = T/\delta t = 50000$ (40000 for Swimmer environment) is the fixed horizon of the fine-grained trajectories, Z is a random trajectory, and $N_m(\varepsilon)$ is the number of discrete points used for trajectory z_m, generated by the adaptive method with tolerance ε. Figure 2 indicates that the adaptive method generally outperforms the uniform method in estimating the value function, especially when a larger discretization h, which corresponds to a larger tolerance ε, is used. Notably, the adaptive method is more robust to a larger range of discretization. Note that while we expect low h to have larger variance, we emphasize that the MuJoCo environments are deterministic with small variations in the initial states. Consequently, the return variance of a good policy will be small as the returns will mostly differ in the initial timesteps.

6 Related Work

The continuous-time problem setting has received considerable attention in previous works (Doya, 2000; Lee & Sutton, 2021; Yildiz et al., 2021; Lutter et al., 2021a;b), which has attempted to address problems in the regime of deterministic dynamics (Kim et al., 2021) and stochastic dynamics (Munos & Bourgine, 1997; Munos, 2006), aiming to solve the continuous-time problem rather than discretizing it. Tallec et al. (2019) showed that action-value function do not hold the same meaning in near-continuous time control and developed methods based on advantage estimation (Baird, 1994) for the deep RL setting. The study of time discretization in RL tackles the challenges stemming from the necessity to adapt continuous-time models to the discrete computational frameworks utilized in practice. Previous studies have focused on this issue, ranging from uniform temporal resolution (Metelli et al., 2020; Zhang et al., 2024) to adopting non-uniform strategies for determining the optimal timing for actions (Biedenkapp et al., 2021; Sharma et al., 2017; Lakshminarayanan et al., 2017; Park et al., 2021).

Improved temporal resolution can also be viewed as a means for conducting more effective and focused exploration (Dabney et al., 2021). Jacq et al. (2022) considers temporal resolution as a method for the agent to determine when to execute actions that maximize impact while minimizing the number of decisions made. Patel et al. (2023) extends this concept, balancing the quantity of decisions to facilitate learning policies that react either quickly or slowly. Additionally, temporal resolution can be examined through the lens of temporal abstraction within the options framework (Sutton et al., 1999), where each option comprises primitive actions and the agent aims to identify a termination condition. Temporal resolution can also be interpreted as introducing skip connections in the transition function (Biedenkapp et al., 2021) to enhance sample efficiency while preserving optimality. Ultimately, these concepts converge on the goal of implementing real-time RL systems (Ramstedt & Pal, 2019), where continuous-time problems are discretized for control and the environment's state does not halt for the agent to decide on its next action.

7 Conclusion and Future Directions

We have investigated an adaptive temporal discretization algorithm for Monte-Carlo policy evaluation in RL. Our results demonstrate that for sufficiently smooth reward functions the data efficiency of policy evaluation can be significantly improved. It remains future work to extend the adaptive discretization scheme to non-smooth reward functions or reward functions exhibiting highly stochastic behaviour.

Acknowledgement

We appreciate funding from the CCAI Chairs program, the RLAI laboratory, Amii, and NSERC of Canada.

References

L.C. Baird. Reinforcement learning in continuous time: advantage updating. In *IEEE International Conference on Neural Networks (ICNN'94)*, 1994.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 1983.

André Biedenkapp, Raghu Rajan, Frank Hutter, and Marius Lindauer. Temporl: Learning when to act. In *International Conference on Machine Learning*, 2021.

Will Dabney, Georg Ostrovski, and Andre Barreto. Temporally-extended ε-greedy exploration. In *International Conference on Learning Representations*, 2021.

Philip J Davis and Philip Rabinowitz. *Methods of numerical integration*. Courier Corporation, 2007.

Kenji Doya. Reinforcement learning in continuous time and space. *Neural computation*, 2000.

Gene F. Franklin, Michael L. Workman, and Dave Powell. *Digital Control of Dynamic Systems*. Addison-Wesley Longman Publishing Co., Inc., 1997.

Pedro Gonnet. A review of error estimation in adaptive quadrature. *ACM Computing Surveys (CSUR)*, 2012.

Alexis Jacq, Johan Ferret, Olivier Pietquin, and Matthieu Geist. Lazy-mdps: Towards interpretable rl by learning when to act. In *International Conference on Autonomous Agents and Multiagent Systems*, 2022.

Jeongho Kim, Jaeuk Shin, and Insoon Yang. Hamilton-jacobi deep q-learning for deterministic continuous-time systems with lipschitz continuous controls. *Journal of Machine Learning Research*, 2021.

Aravind Lakshminarayanan, Sahil Sharma, and Balaraman Ravindran. Dynamic action repetition for deep reinforcement learning. *AAAI Conference on Artificial Intelligence*, 2017.

Jaeyoung Lee and Richard S Sutton. Policy iterations for reinforcement learning problems in continuous time and space—fundamental theory and methods. *Automatica*, 2021.

Michael Lutter, Boris Belousov, Shie Mannor, Dieter Fox, Animesh Garg, and Jan Peters. Continuous-time fitted value iteration for robust policies. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021a.

Michael Lutter, Shie Mannor, Jan Peters, Dieter Fox, and Animesh Garg. Value iteration in continuous actions, states and time. In *International Conference on Machine Learning*, 2021b.

Alberto Maria Metelli, Flavio Mazzolini, Lorenzo Bisi, Luca Sabbioni, and Marcello Restelli. Control frequency adaptation via action persistence in batch reinforcement learning. In *International Conference on Machine Learning*, 2020.

Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, 1990.

Rémi Munos. Policy gradient in continuous time. *Journal of Machine Learning Research*, 2006.

Rémi Munos and Paul Bourgine. Reinforcement learning for continuous stochastic control problems. *Advances in neural information processing systems*, 1997.

Seohong Park, Jaekyeom Kim, and Gunhee Kim. Time discretization-invariant safe action repetition for policy gradient methods. *Advances in Neural Information Processing Systems*, 2021.

Devdhar Patel, Terrence Sejnowski, and Hava Siegelmann. Temporally layered architecture for efficient continuous control. *arXiv preprint arXiv:2305.18701*, 2023.

Simon Ramstedt and Chris Pal. Real-Time Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2019.

Timothy Sauer. *Numerical Analysis*. Addison-Wesley Publishing Company, 2006.

Sahil Sharma, Aravind Srinivas, and Balaraman Ravindran. Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint arXiv:1702.06054*, 2017.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 1999.

Corentin Tallec, Léonard Blier, and Yann Ollivier. Making deep Q-learning methods robust to time discretization. In *International Conference on Machine Learning (ICML)*, 2019.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, 2023.

Cagatay Yildiz, Markus Heinonen, and Harri Lähdesmäki. Continuous-time model-based reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2021.

Zichen Vincent Zhang, Johannes Kirschner, Junxi Zhang, Francesco Zanini, Alex Ayoub, Masood Dehghan, and Dale Schuurmans. Managing temporal resolution in continuous value estimation: A fundamental trade-off. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.