Tutorial: Reinforcement Learning for Character Animation

Akihisa Watanabe
akihisa@ruri.waseda.jp
Waseda University
Tokyo, Japan

ABSTRACT

This tutorial presents a detailed exploration of Reinforcement Learning (RL) with a specific focus on its application in physics-based character animation. Centered around Policy Gradient methods, the paper methodically introduces foundational concepts such as Markov Decision Processes and progresses to more advanced topics including Actor-Critic Algorithms and Generalized Advantage Estimation. A key feature of this tutorial is its practical approach, incorporating step-by-step applications and examples drawn from the CS285 course at UC Berkeley. This paper is designed to serve both beginners and experienced individuals in the field, offering a balanced mix of theoretical knowledge and hands-on techniques for applying RL in character animation, and providing insights into the potential benefits and applications of these techniques within the domain. Our code is available at https://github.com/Akihisa-Watanabe/rl-character-animation-tutorial.

CCS CONCEPTS

• **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

KEYWORDS

Do, Not, Us, This, Code, Put, the, Correct, Terms, for, Your, Paper

ACM Reference Format:
Akihisa Watanabe. 2023. Tutorial: Reinforcement Learning for Character Animation. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Developing controllers for physics-based character animation is a significant challenge, merging the need for aesthetic appeal with physical accuracy and reactivity. The integration of reinforcement learning (RL) has transformed this field, enabling the creation of characters that not only move realistically but also exhibit a broad range of athletic abilities, from basic walking to intricate acrobatics. This tutorial delves into these advancements, with a focus on how RL aids in animating characters that interact dynamically with their environments.

We start with an introduction to RL, designed for those new to the concept, then move to its specific application in the realm of character animation. Special attention is given to the insights from Sergey Levine's CS 285 lecture series at UC Berkeley, which provides a fundamental understanding of RL principles and their direct application in computer animation.

The tutorial concludes with a practical guide, featuring step-by-step instructions and examples from current research. It is tailored for readers involved in computer animation, ranging from novices to seasoned professionals. The goal is to impart a thorough understanding of how RL can be utilized to generate advanced, dynamic, and lifelike animations, equipping you with the necessary theoretical knowledge and practical skills.

2 PRELIMINARIES OF REINFORCEMENT LEARNING

Our task is structured as a standard reinforcement learning problem where an agent interacts with its environment according to a policy to maximize a reward signal. In this section, we define basic reinforcement learning concepts, following the standard definition [1, 7]. Reinforcement learning addresses the problem of learning to control a dynamical system in a general sense. The dynamical system is fully defined by a fully-observed or partially-observed Markov decision process (MDP).

Definition 2.1. In reinforcement learning, a Markov Decision Process is defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, r, \gamma, p(s_0))$, where $\mathcal{S}$ is the set of states $s \in \mathcal{S}$, $\mathcal{A}$ is the set of actions $a \in \mathcal{A}$, T defines the transition probabilities $T(s_{t+1}|s_t, a_t)$, r is the reward function mapping state-action pairs to scalar rewards, γ is the discount factor, and $p(s_0)$ represents the initial state distribution. The inclusion of $p(s_0)$ acknowledges the importance of the starting conditions in determining the trajectory of the agent's actions and states under a specific policy.

Time: In physics-based character animation, such as in Isaac Gym[2], it's important to differentiate between simulation time steps (Δt_{sim}) and control time steps (Δt_{ctrl}). Simulation time steps, often much finer, e.g., 1/120th of a second, update the physical state of the environment. Control time steps, possibly less frequent like every 1/60th of a second, update the control actions. This distinction allows for accurate simulation while managing computational efficiency in control algorithms. In this paper, the time step t refers to the control time step Δt_{ctrl}, a standard approach in physics-based character animation.

State: The statess_t at time step$t$ consists of a set of features that describe the configuration of the character's body in the context of character animation. An example of how the state is conceptualized

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

in this context is provided, as demonstrated in the paper [8]:

$$\boldsymbol{s}_t = (\boldsymbol{p}_t, \dot{\boldsymbol{p}}_t, \boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \hat{\boldsymbol{p}}_{t+1}, \hat{\boldsymbol{q}}_{t+1}) \tag{1}$$

where

- $\boldsymbol{p}_t$: joint positions in the character's root coordinates at time step t.
- $\dot{\boldsymbol{p}}_t$: joint linear velocities in the character's root coordinates at time step t.
- $\boldsymbol{q}_t$: joint rotations in the joints' local coordinates at time step t.
- $\dot{\boldsymbol{q}}_t$: joint angular velocities in the joints' local coordinates at time step t.
- $\hat{\boldsymbol{p}}_{t+1}$: anticipated target joint positions for the next time step $t+1$.
- $\hat{\boldsymbol{q}}_{t+1}$: anticipated target joint rotations for the next time step $t+1$.

Action: In many prior works[3, 4, 8], each action $\boldsymbol{a}_t$ specifies the target state (such as rotations) for PD controllers positioned at each of the character's joints. For example, we can define action as follows:

$$\boldsymbol{a}_t = \boldsymbol{u}_t \tag{2}$$

where $\boldsymbol{u}_t$ is the target joint angles for the PD controller. At each time step t, the joint torques τ_t are computed as:

$$\tau_t = \boldsymbol{K}_p \cdot (\boldsymbol{a}_t - \boldsymbol{q}_t^{nr}) - \boldsymbol{K}_d \cdot \dot{\boldsymbol{q}}_t^{nr} \tag{3}$$

where $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$ denote the parameters of the PD controllers that determine the stiffness and damping of each joint, $\boldsymbol{q}_t^{nr}$ and $\dot{\boldsymbol{q}}_t^{nr}$ are the joint rotations and angular velocities of the non-root joints.

Reward: The reward function plays a role in guiding the agent's actions. The reward, denoted as $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$, is computed when the agent performs an action $\boldsymbol{a}_t$ that transitions it from state $\boldsymbol{s}_t$ to state $\boldsymbol{s}_{t+1}$. The design of the reward function can be leveraged to make the agent imitate certain motion data or to specify a particular task that the agent needs to accomplish.

In the study [8], a part of the reward function used for imitating motion data is given by

$$r_t^v = \exp\left[-\alpha_v \sum_j \left(||\dot{\boldsymbol{q}}_t^j - \hat{\dot{\boldsymbol{q}}}_t^j||^2\right)\right]. \tag{4}$$

Here, α_v is a scaling factor that adjusts the sensitivity of the reward function to velocity discrepancies. The index j represents the joint number, indicating that the computation is performed for each joint individually. The reward r_t^v quantifies the difference between the simulated motion's local joint velocities $\dot{\boldsymbol{q}}_t^j$ and the reference motion's velocities $\hat{\dot{\boldsymbol{q}}}_t^j$, encouraging the agent to closely mimic the reference motion at each joint.

The final goal in a reinforcement learning problem is to learn a policy, which defines a distribution over actions conditioned on states, $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$. From these definitions, we can derive the trajectory distribution. The trajectory is a sequence of states and actions of length T, given by $\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, \cdots, \boldsymbol{s}_T, \boldsymbol{a}_T)$, where T may be infinite. The trajectory distribution p_π for a given MDP M and policy π is given by

$$p_\pi(\tau) = p(\boldsymbol{s}_0) \prod_{t=0}^{T} \pi(\boldsymbol{a}_t|\boldsymbol{s}_t) p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{5}$$

This represents the probability of obtaining the trajectory τ as a result of the agent selecting actions $\boldsymbol{a}_t \sim \pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ according to the policy π. Notably, in an MDP framework, the current state $\boldsymbol{s}_t$ and action $\boldsymbol{a}_t$ are determined based on the information from only the immediate previous time step. This principle, known as the Markov property, implies that the future states and actions are independent of the past, given the current state and action. As a result, the trajectory distribution is expressed as a product of the policy $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ and the state transition probability $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$, simplifying the computation and analysis of the problem.

The objective function in reinforcement learning, denoted as $J(\pi)$, is typically defined as the expected cumulative reward , also known as the 'return', under a given policy π. This return is calculated as an expectation under the trajectory distribution $p_\pi(\tau)$:

$$J(\pi) = \mathbb{E}_{\tau\sim p_\pi(\tau)}\left[\sum_{t=0}^{T-1} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right]. \tag{6}$$

Here, $\gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ represents the discounted reward at each time step t, where γ is the discount factor that determines the present value of future rewards. The final goal of the agent is to find an optimal policy π^* that maximizes this objective function $J(\pi)$:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, J(\pi). \tag{7}$$

In other words, the optimal policy is the one that, on average, results in the highest cumulative reward throughout the trajectory.

3 POLICY GRADIENT METHODS

Policy Gradient Methods are a class of reinforcement learning algorithms that optimize the policy by learning its parameters[5], denoted as θ. This approach is similar to how parameters are updated in traditional supervised learning algorithms. However, in this context, the parameters θ are associated with the policy denoted as π_θ. The parameters are updated iteratively in the direction that maximizes the performance measure $J(\theta)$. The simplest form of this update is given by the gradient ascent rule:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta). \tag{8}$$

Where, α is the learning rate, and $\nabla_\theta J(\theta)$ is the gradient of the performance measure with respect to the policy parameters.

Theorem 3.1. (Policy Gradient Theorem). Given a policy parameterized by θ, denoted as π_θ, the gradient of the objective function $J(\theta)$ with respect to the policy parameters θ can be expressed as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_{\pi_\theta}(\tau)}\left[\sum_{t=0}^{T-1} \gamma^t \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t) \left(\sum_{t=0}^{T-1} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right)\right]. \tag{9}$$

In this equation, τ represents a trajectory sampled from the policy π_θ, γ^t is the discount factor at time step t, $\nabla_\theta \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)$ is the gradient of the log-probability of the action $\boldsymbol{a}_t$ given state $\boldsymbol{s}_t$ under policy π_θ, and $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ is the reward received at time step t for taking action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$.

The term $\gamma^t \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)$ in Equation (9) is similar to the maximum likelihood. Intuitively, this means that we are assigning more weight to more rewarding trajectories by making trajectories with higher rewards more probable.In other words, the algorithm is designed to learn from the data in a way that it is most likely to choose trajectories that yield higher rewards.

One challenge when directly applying the Policy Gradient Theorem for updating θ is the high variance of the policy gradient estimates. This is because the gradient is estimated based on sampled trajectories, and the rewards along these trajectories can vary significantly. This variability in rewards leads to a high variance in the gradient estimates, which can in turn lead to unstable learning and make it harder for the algorithm to converge to an optimal policy. Various techniques, such as using a baseline or employing advanced variance reduction methods, have been proposed to address this issue.

3.1 Causality and Baselines

Causality: One simple way to tackle high variance is to use the fact of causality: policy at time t' cannot affect reward at t if $t < t'$. This simple, commonsensical idea allows us to discard some operands in the summation

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_{\pi_\theta}(\tau)} \left[\sum_{t=0}^{T-1} \gamma^t \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t) \left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) \right) \right]. \tag{10}$$

and we define the second item in the summation as the "reward-to-go". Notice that in the reward-to-go term, we start the summation from time t instead of 1, by causality. The idea is that we are multiplying the likelihood by smaller numbers due to the reduction of the summation term, so we can reduce the variance to some extent.

Baselines: Another way to reduce variance is to subtract a baseline from the reward function. The baseline $b(\boldsymbol{s}_t)$ helps stabilize learning by reducing variance. The objective function we aim to optimize in policy gradient methods can be represented by the equation:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_{\pi_\theta}(\tau)} \left[\sum_{t=0}^{T-1} \gamma^t \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t) \times \left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) - b(\boldsymbol{s}_t) \right) \right]. \tag{11}$$

Subtracting a baseline $b(\boldsymbol{s}_t)$ in policy gradient methods reduces the variance of gradient estimates. This is because the subtraction leads to smaller squared terms in the variance calculation, thereby reducing the overall variance and stabilizing learning.

The baseline $b(\boldsymbol{s}_t)$ in policy gradient methods is a function designed to estimate the expected rewards for a given state $\boldsymbol{s}_t$. Its training involves comparing the baseline's predictions of these expected rewards with the actual returns (target values) from those states. This process utilizes the mean squared error (MSE) loss, which is calculated as the average squared difference between the predicted values $b(\boldsymbol{s}_i)$ by the baseline and the actual target values y_i:

$$l(b) = \mathbb{E}_{\boldsymbol{s}_i, y_i \sim D} \left[||y_i - b(\boldsymbol{s}_i)||^2 \right], \tag{12}$$

where D is the dataset accumulated by the agent. The objective is to accurately adjust the baseline function to predict the expected rewards for states, thereby helping to reduce the variance of the policy gradient estimates.

3.2 Advantage Function

Now, we introduce the Advantage function, a key concept that further refines policy gradient methods. The Advantage function $A^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)$ quantifies the relative value of taking a specific action $\boldsymbol{a}_t$ in a given state $\boldsymbol{s}_t$, compared to the average value of all possible actions in that state according to policy π. The Advantage function is defined as the difference between the Q-function, $Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)$, which represents the expected future return of taking action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$ and then following policy π, and the value function, $V^\pi(\boldsymbol{s}_t)$, which represents the expected future return of being in state $\boldsymbol{s}_t$ and then following policy π:

Definition 3.2. The Q-function represents the expected future return of taking action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$ and then following policy π for all future timesteps:

$$Q^{\pi,\gamma}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathbb{E}_{\tau \sim p_{\pi_\theta}(\tau|\boldsymbol{s}_t)} \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) \right]. \tag{13}$$

Definition 3.3. The value function represents the expected future return of being in state $\boldsymbol{s}_t$ and then following policy π for all future timesteps:

$$V^{\pi,\gamma}(\boldsymbol{s}_t) = \mathbb{E}_{\boldsymbol{a}_t \sim p_{\pi_\theta}(\tau|\boldsymbol{a}_t, \boldsymbol{s}_t)} \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) \right]. \tag{14}$$

Definition 3.4. Given a policy π and a discount factor γ, the Advantage function $A^{\pi,\gamma}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ quantifies the relative value of taking a specific action $\boldsymbol{a}_t$ in a given state $\boldsymbol{s}_t$ over the average value of all possible actions in that state under the policy π. It is defined as:

$$A^{\pi,\gamma}(\boldsymbol{s}_t, \boldsymbol{a}_t) = Q^{\pi,\gamma}(\boldsymbol{s}_t, \boldsymbol{a}_t) - V^{\pi,\gamma}(\boldsymbol{s}_t). \tag{15}$$

The advantage function thus quantifies the relative benefit of taking action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$ by comparing the Q-function to the value function. A positive advantage function indicates that taking action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$ is expected to yield a higher return than the average return of all possible actions in state $\boldsymbol{s}_t$. Conversely, a negative advantage function indicates that taking action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$ is expected to yield a lower return than the average return of all possible actions in state $\boldsymbol{s}_t$.

With the advantage function, we can express a policy gradient estimate using a Monte Carlo estimate:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_{\pi_\theta}(\tau)} \left[\sum_{t=0}^{T-1} \gamma^t \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t) A^{\pi,\gamma}(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]. \tag{16}$$

This equation can be intuitively understood as follows: a step in the direction of the policy gradient should increase the likelihood of actions that perform better than average, and decrease the likelihood of actions that perform worse than average.

3.3 Actor Critic Algorithm

Building upon the previously introduced concepts of the Advantage function, Q-function, and Value function, we now derive the Actor Critic Algorithm. In the context where $\boldsymbol{s}_t, \boldsymbol{a}_t$ are not random variables, the Q function can be expressed as follows:

$$\begin{aligned} Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) &= r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \mathbb{E}_{\tau \sim p_{\pi_\theta}(\tau)} \left[\sum_{t'=t+1}^{T-1} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) \right] \\ &= r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1} \sim p_{\pi_\theta}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)} \left[V^{\pi}(\boldsymbol{s}_{t+1}) \right]. \end{aligned} \tag{17}$$

For practical purposes, we introduce a simplifying assumption. Although it may not always hold true in reality, we consider the actual $\boldsymbol{s}_{t+1}$ observed in the current trajectory as a representative of the average of $\boldsymbol{s}_{t+1}$ that we would obtain. This allows us to approximate the Q function as:

$$Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) \approx r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma V^{\pi,\gamma}(\boldsymbol{s}_{t+1}). \tag{18}$$

This approximation is appealing because it simplifies the learning of the advantage function. The advantage function now only depends on V, which in turn only depends on s, unlike Q which depends on both a and s. This reduces the complexity of learning since Q requires a larger sample size due to its increased dimensionality. Using this approximation, we can substitute into the advantage function:

$$\begin{aligned} A^{\pi,\gamma}(\boldsymbol{s}_t, \boldsymbol{a}_t) &= Q^{\pi,\gamma}(\boldsymbol{s}_t, \boldsymbol{a}_t) - V^{\pi,\gamma}(\boldsymbol{s}_t) \\ &\approx r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma V^{\pi,\gamma}(\boldsymbol{s}_{t+1}) - V^{\pi,\gamma}(\boldsymbol{s}_t). \end{aligned} \tag{19}$$

This leads us to the actor-critic formulation:

$$\begin{aligned} \nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_{\pi_\theta}(\tau)} \Bigg[\sum_{t=0}^{T-1} \gamma^t \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t) \\ \times \left(r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma V^{\pi,\gamma}(\boldsymbol{s}_{t+1}) - V^{\pi,\gamma}(\boldsymbol{s}_t) \right) \Bigg]. \end{aligned} \tag{20}$$

In policy gradient with baselines, there is no bias in our estimation, but there might be high variance due to our single-sample estimation of the Q-function. On the other hand, in the actor-critic algorithm, we have lower variance due to the critic, but we end up having a biased estimation because of the possible modeling error in $V_{\phi}^{\pi,\gamma}(\cdot)$.

3.4 Generalized Advantage Estimation

The actor-critic advantage function (equation 19) and the Monte Carlo policy gradient (baseline) present a trade-off between bias and variance. The actor-critic advantage function has lower variance but higher bias, while the Monte Carlo policy gradient has lower bias but higher variance. This trade-off becomes more pronounced as we project our trajectory further into the future, where the variance increases due to the insufficiency of the current single sample approximation for future representation. To manage this trade-off, it is advantageous to use the actor-critic based advantage for long-term predictions, which incorporates a larger number of states for better approximation in the long run, and the Monte Carlo based one for short-term predictions, which excels in achieving accurate values in the near term.

To avoid the trajectory extending to a point where the variance becomes excessively large, we can truncate it. This concept is illustrated in Figure 1, where the trajectory is cut off before the variance escalates excessively. We can estimate the advantage function by integrating the two approaches, applying the Monte Carlo approach only for the initial n steps. This is mathematically expressed as:

$$A_n^{\pi,\gamma}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \sum_{t'=t}^{t+n-1} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) - \hat{V}_{\phi}^{\pi,\gamma}(\boldsymbol{s}_t) + \gamma^n \hat{V}_{\phi}^{\pi,\gamma}(\boldsymbol{s}_{t+n}). \tag{21}$$

In this equation, we apply an n-step value function estimator, which sums the reward from the current time to n steps in the future. Often, setting $n > 1$ yields superior performance. The term $\sum_{t'=t}^{t+n-1} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) - \hat{V}_{\phi}^{\pi}(\boldsymbol{s}_t)$ is the Monte Carlo method policy gradient applied over n steps. The last term, $\gamma^n \hat{V}_{\phi}^{\pi}(\boldsymbol{s}_{t+n})$, is the discounted value function at the end of the n-step trajectory. This term is used to estimate the future returns beyond the n-step trajectory, and it is where the actor-critic approach comes into play.

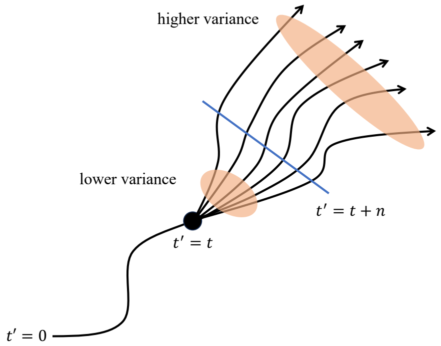

Figure 1: Depiction of trajectory and variance over time. Multiple trajectories are showing that the dispersion is large for distant time steps and small for near time steps. This highlights the need to truncate the trajectory at a certain point ($t' = t + n$) to prevent the variance from escalating excessively.

However, the choice of n is a hyperparameter that needs to be tuned. A small n can lead to high bias and low variance, while a large n can lead to low bias and high variance. To address this issue, we can use a weighted average of all possible n-step estimators, which is known as Generalized Advantage Estimation (GAE) [6]. The GAE is defined as:

$$A_{\text{GAE}}^{\pi,\gamma,\lambda}(\boldsymbol{s}_t, \boldsymbol{a}_t) = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} A_n^{\pi,\gamma}(\boldsymbol{s}_t, \boldsymbol{a}_t), \tag{22}$$

where $\lambda \in [0, 1]$ is a hyperparameter that determines the trade-off between bias and variance. When $\lambda = 0$, the GAE reduces to the one-step actor-critic estimator, which has high bias and low variance. When $\lambda = 1$, the GAE reduces to the Monte Carlo estimator, which has low bias and high variance. By choosing an appropriate value of λ, we can balance the trade-off between bias and variance.

The equation 22 can be expanded as:

$$\begin{aligned} A_{GAE}^{\pi,\gamma,\lambda}(\boldsymbol{s}_t, \boldsymbol{a}_t) &= (1 - \lambda)(A_1^{\pi,\gamma} + \lambda A_2^{\pi,\gamma} + \lambda^2 A_3^{\pi,\gamma} + \cdots) \\ &= \sum_{t'=t}^{\infty} (\gamma\lambda)^{t'-t} \delta_{t'}. \end{aligned} \tag{23}$$

Here, $\delta_{t'}$ is defined as $\delta_{t'} = r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) + \gamma V^{\pi,\gamma}(\boldsymbol{s}_{t'+1}) - V^{\pi,\gamma}(\boldsymbol{s}_{t'})$, which is the temporal difference (TD) residual of V with discount γ, following the description by [7]. Notably, $\delta_{t'}$ serves as an estimate of the advantage of the action $\boldsymbol{a}_{t'}$, representing the benefit of taking action $\boldsymbol{a}_{t'}$ in state $\boldsymbol{s}_{t'}$ compared to the average value of that state.

For the finite horizon case, the following representation is apt:

$$A_{GAE}^{\pi,\gamma,\lambda}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \sum_{t'=t}^{T-1} (\gamma\lambda)^{t'-t} \delta_{t'}. \tag{24}$$

This formulation allows for an efficient implementation of the generalized advantage estimator, employing a recursive calculation:

$$A_{\text{GAE}}^{\pi,\gamma,\lambda}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \delta_t + \gamma\lambda A_{GAE}^{\pi,\gamma,\lambda}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}). \tag{25}$$

This approach systematically balances between bias and variance, thus optimizing the learning process.

REFERENCES

[1] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *CoRR* abs/2005.01643 (2020). arXiv:2005.01643 https://arxiv.org/abs/2005.01643

[2] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. 2021. Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning.

[3] Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. 2022. ASE: Large-scale Reusable Adversarial Skill Embeddings for Physically Simulated Characters. *ACM Trans. Graph.* 41, 4, Article 94 (July 2022).

[4] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. 2021. AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control. *ACM Trans. Graph.* 40, 4, Article 1 (July 2021), 15 pages. https://doi.org/10.1145/3450626.3459670

[5] Jan Peters and Stefan Schaal. 2008. Reinforcement learning of motor skills with policy gradients. *Neural Networks* 21, 4 (2008), 682–697. https://doi.org/10.1016/j.neunet.2008.02.003 Robotics and Neuroscience.

[6] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2016. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[7] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA.

[8] Haotian Zhang, Ye Yuan, Viktor Makoviychuk, Yunrong Guo, Sanja Fidler, Xue Bin Peng, and Kayvon Fatahalian. [n. d.]. Learning Physically Simulated Tennis Skills from Broadcast Videos. *ACM Trans. Graph.* ([n. d.]), 14 pages. https://doi.org/10.1145/3592408

Received 20 November 2023; revised 20 November 2023; accepted 20 November 2023