Logarithmic Function Matters Policy Gradient Deep Reinforcement Learning

Qi Liu[1], Jingxiang Guo[1], Zhongjian Qiao[2], Pengbin Chen[1], Yanjie Li[1]*

***Abstract*—This paper studies the influence of logarithmic functions in policy gradient deep reinforcement learning (RL) methods, a subject previously under-emphasized in the field. It delves into how different logarithmic bases, specifically *loge*, *log2*, and *log10*, influence the efficiency of policy gradient methods. We analyze the role of these logarithmic functions and their gradients when using Stochastic Gradient Descent as the optimizer in deep RL. Experimental results show that the choice of the logarithmic base influences the policy gradient methods. *loge* and *log2* are generally more effective than *log10* in deep RL algorithms, but *log10* exhibits more stability. Based on analysis and experimental results, we propose the Logarithmic Basis Policy Gradient (LBPG) and Adaptive LBPG algorithms to enhance policy gradient methods. LBPG denotes policy gradient methods with *loge*, *log2*, and *log10* as the logarithmic base, respectively. Furthermore, Adaptive LBPG dynamically selects a logarithmic basis among *loge*, *log2*, and *log10* based on the variance-to-mean ratio of the returns to optimize speed and stability. Experimental results on video games show the importance of choosing appropriate logarithmic functions for policy gradient RL methods and highlight the potential of Adaptive LBPG to refine learning in deep RL tasks.**

***Index Terms*—Deep reinforcement learning, policy gradient methods, logarithmic basis policy gradient, video games**

I. Introduction

DEEP reinforcement learning (RL) has been applied to various challenging domains, such as video games [1], [2], robotic control [3], [4], and multi-agent systems [5]. Deep RL methods can be divided into value-based methods [1] and policy gradient methods [6]. This paper focuses on policy gradient methods.

A famous policy gradient method is the REINFORCE algorithm [6] that approximates the policy using a network. After that, deep RL methods that approximate the policy and the value function using neural networks are called the actor-critic architecture, where 'actor' denotes the policy, and 'critic' denotes the value function [7]. Modern variants include Natural Actor-Critic [8], Advantage Actor-Critic [9], Trust Region Policy Optimization (TRPO) [10], Proximal Policy Optimization (PPO) [11] algorithms, etc.

This work was supported by the National Natural Science Foundation of China [61977019, U1813206] and Shenzhen Fundamental Research Program [JCYJ20180507183837726, JCYJ20220818102415033, JSGG20201103093802006]. *(Corresponding author: Yanjie Li, autolyj@hit.edu.cn)*

[1]Qi Liu, Jingxiang Guo, Pengbin Chen, and Yanjie Li are with the Guangdong Key Laboratory of Intelligent Morphing Mechanisms and Adaptive Robotics and School of Mechanical Engineering and Automation, the Harbin Institute of Technology Shenzhen, 518055.

[2]Zhongjian Qiao is with Tsinghua Shenzhen International Graduate School, Tsinghua University, China.

Literature [12] presents the n-step TD method to balance the return bias-variance trade-off. The n-step TD method can be generalized to the Monte Carlo (MC) and one-step TD methods. One can shift from one to the other smoothly as needed to meet a particular task's demands. Literature [9] proposes the advantage function to reduce the variance of the estimated value function. After that, general advantage estimation (GAE) [13] uses value functions to substantially reduce variance at the cost of some bias, with an exponentially weighted estimator of the advantage function analogous to TD(λ) [12]. Literature [14] proposes the control variate method to reduce the estimation variance of the return and does not change the expected return value. The literature [15] presents a detailed overview of off-policy [12] evaluation in deep RL.

There are two most widely used algorithms in deep RL: Advantage Actor-Critic (A2C) [9] and General Advantage Estimation (GAE) [13]. A2C is one of the most widely used methods in deep RL. After that, GAE [13] uses value functions to substantially reduce the variance of policy gradient estimates at the cost of some bias, with an exponentially weighted estimator of the advantage function analogous to TD(λ) [12].

Although policy gradient methods have been studied extensively, most work focuses on the value function to address the bias-variance trade-off. Our approach diverges from conventional policy gradient methods by focusing on logarithmic functions within policy gradient methods. We examine how various logarithmic bases impact policy gradient methods. We select the logarithmic bases of *loge*, *log2*, and *log10* because of their prevalence in mathematical computations.

This paper aims to understand how the choice of these bases and their corresponding gradients impact the performance and stability of policy gradient methods. This ability to be adaptive can affect the game process, as shown in section IV. Previous literature [16] has addressed the exploration process using composed parameterized skills. Additionally, literature [17] has validated that adaptive game parameters impact the time required for learning.

The main contributions of this work can be summarized as follows:

- In this paper, we analyze the influence of different logarithmic bases (*loge*, *log2*, and *log10*) on policy gradient methods in policy gradient deep RL methods. Experimental results in classical control benchmark environments illustrate how *loge* and *log2* generally lead to faster learning, whereas *log10* offers enhanced stability. These findings provide practical insights for deep RL researchers in selecting suitable logarithmic functions.

- We propose the adaptive Logarithmic Basis Policy Gradient (LBPG) algorithm that dynamically adapts among *log2*, *loge*, and *log10* in policy gradient based on the variance-to-mean ratio of returns. Experimental results show that Adaptive LBPG can balance learning efficiency and stability under different reward conditions, enabling tailored learning processes that are fine-tuned to environments' unique challenges and dynamics.

II. Preliminaries

Policy gradient methods directly parameterize the policy and optimize the policy parameters by policy gradient [12]. The policy π_θ is usually modeled with a parameterized function for θ. Deep RL aims to learn a policy π_θ that can maximize the expected return $J(\pi_\theta)$:

$$\begin{aligned} J(\pi_\theta) &= \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) V^{\pi_\theta}(s) \\ &= \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} Q^{\pi_\theta}(s,a)\nabla_\theta \pi_\theta(a \mid s) \end{aligned} \tag{1}$$

The parameterized policy π_θ can be updated by the gradient [12]:

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &\propto \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} Q^{\pi_\theta}(s,a)\nabla_\theta \pi_\theta(a \mid s) \\ &= \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} Q^{\pi_\theta}(s,a)\pi_\theta(a \mid s)\frac{\nabla_\theta \pi_\theta(a \mid s)}{\pi_\theta(a \mid s)} \\ &= \mathbb{E}^{\pi_\theta}\left[Q^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a \mid s)\right] \end{aligned} \tag{2}$$

where $\nabla_\theta J(\pi_\theta)$ denotes the gradients of $J(\pi_\theta)$ with respect to θ. $d^{\pi_\theta}(s)$ represents the state distribution under π_θ. V^{π_θ} denotes the state-value function, Q^{π_θ} denotes the action-value function, and $\mathbb{E}^{\pi_\theta}$ denotes the expectation under policy π_θ. At timestep, t, an action a_t is taken under policy π_θ at state s_t, and thereafter following policy π_θ. The policy gradient theorem (Eq. (2)) lays the theoretical foundation for various policy gradient algorithms [6], [8]–[11]. Policy gradient methods maximize the expected total reward by repeatedly estimating the gradient.

Eq. (2) shows that the policy gradient involves the state-action value term $Q^{\pi_\theta}(s,a)$. Much work on policy gradient methods [9], [12], [13] focuses on the value function part ($Q^{\pi_\theta}(s,a)$) in policy gradient, intending to balance the bias-variance trade-off. There are several different related expressions for the policy gradient, which have the form:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}^{\pi_\theta}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right)\right] \tag{3}$$

where Ψ_t may be one of the following: (1) $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory; (2) $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t; (3) $\sum_{t'=t}^{\infty} r_{t'} - b\left(s_t\right)$: base-lined version of (2); (4) $Q^{\pi_\theta}\left(s_t, a_t\right)$: state-action value function; (5) $r_t + V^{\pi_\theta}\left(s_{t+1}\right) - V^{\pi_\theta}\left(s_t\right)$: temporal-difference (TD) residual; (6) $A^{\pi_\theta}\left(s_t, a_t\right) = Q^{\pi_\theta}\left(s_t, a_t\right) - V^{\pi_\theta}\left(s_t\right)$: advantage function. The latter formulas use the following definitions:

$$V^{\pi_\theta}\left(s_t\right) = \mathbb{E}^{\pi_\theta}_{s_{t:\infty}}\left[\sum_{l=0}^{\infty} \gamma^l r_{t+l}\right] \tag{4}$$

$$Q^{\pi_\theta}\left(s_t, a_t\right) = \mathbb{E}^{\pi_\theta}_{\substack{s_{t:\infty}\\ a_{t:\infty}}}\left[\sum_{l=0}^{\infty} \gamma^l r_{t+l}\right] \tag{5}$$

where γ is a discount factor.

Literature [13] proposes γ-just estimator of the advantage function, shown in Eq. (6). The only source of bias in the estimator is the discount factor γ. Note that the discount factor γ can also be seen as a variance reduction technique that reduces the weight of future rewards but introduces some bias.

$$\begin{aligned} &\mathbb{E}^{\pi_\theta}_{\substack{s_{t:\infty}\\ a_{t:\infty}}}\left[\hat{A}_t\left(s_{0:\infty}, a_{0:\infty}\right)\nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right)\right] \\ &= \mathbb{E}^{\pi_\theta}_{\substack{s_{t:\infty}\\ a_{t:\infty}}}\left[A^{\pi_\theta,\gamma}\left(s_t, a_t\right)\nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right)\right] \end{aligned} \tag{6}$$

where $A^{\pi_\theta,\gamma}\left(s_t, a_t\right) = Q^{\pi_\theta,\gamma}\left(s_t, a_t\right) - V^{\pi_\theta,\gamma}\left(s_t\right)$ is the real discounted advantage function following policy π. For the n-step advantage:

$$\begin{aligned} A_t^{(n)} &= \sum_{i=0}^{n-1} r_{t+i} + \gamma^n V\left(s_{t+n}\right) - V\left(s_t\right) \\ &= r_t + \gamma V\left(s_{t+1}\right) - V\left(s_t\right) \\ &\quad + \gamma\left(\sum_{i=1}^{n-1} r_{t+i+1} + \gamma^{n-1} V\left(s_{t+n}\right) - V\left(s_{t+1}\right)\right) \\ &= \delta_t + \gamma\left(\sum_{i=0}^{n-2} r_{t+i+1} + \gamma^{n-1} V\left(s_{t+n}\right) - V\left(s_{t+1}\right)\right) \\ &= \sum_{i=0}^{n-1} \gamma^i \delta_{t+i} \end{aligned} \tag{7}$$

Eq. (7) considers δ_t as a shaped reward with $V^{\pi_\theta,\gamma}\left(s_t\right)$ as the potential function. The n-step advantage equals these shaped rewards' γ discounted sum.

Literature [13] defines the generalized advantage estimator GAE(γ,λ) as the exponentially weighted average of all n-step advantages:

$$\begin{aligned} A_t^{\mathrm{GAE}(\gamma,\lambda)} &= (1-\lambda)\left(\sum_{n=0}^{\infty} \lambda^n A_t^{(n+1)}\right) \\ &= (1-\lambda)\left(\sum_{n=0}^{\infty} \lambda^n \sum_{i=0}^{n} \gamma^i \delta_{t+i}\right) \\ &= (1-\lambda)\left(\sum_{i=0}^{\infty} \gamma^i \delta_{t+i} \sum_{n=i}^{\infty} \lambda^n\right) \\ &= (1-\lambda)\left(\sum_{i=0}^{\infty} \gamma^i \delta_{t+i} \lambda^i \sum_{n=0}^{\infty} \lambda^n\right) \\ &= (1-\lambda)\left(\sum_{i=0}^{\infty} (\gamma\lambda)^i \delta_{t+i} \frac{1}{1-\lambda}\right) \qquad \text{since } \lambda < 1 \\ &= \sum_{i=0}^{\infty} (\gamma\lambda)^i \delta_{t+i} \end{aligned} \tag{8}$$

III. Logarithmic Basis Policy Gradient

This section examines the impact of different logarithmic bases on policy gradient methods. We analyze these effects and propose the Adaptive LBPG method, which dynamically adjusts the logarithmic function based on real-time training metrics.

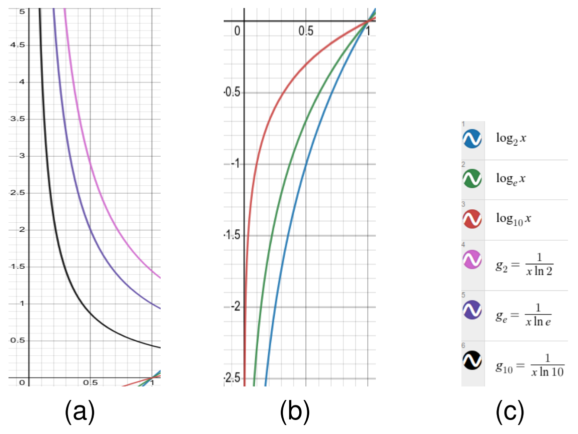

Fig. 1. Logarithmic function and their gradients. (a)(b) The curves of logarithmic function and their gradients. (b) The corresponding curve markings, where x denotes the policy $\pi_\theta\left(a_t \mid s_t\right)$, g_b denotes the gradient of logarithmic function with base b.

Given the widespread use of the advantage function in deep RL methods, as detailed in works like [9] and [11], This paper's investigation is grounded in analyzing the policy gradient with the advantage function:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}^{\pi_\theta}\left[\sum_{t=0}^{\infty} A^{\pi_\theta}\left(s_t, a_t\right) \nabla_\theta \log_b \pi_\theta\left(a_t \mid s_t\right)\right] \quad (9)$$

In Eq (9), the subscript b in $\log_b$ denotes the base of the logarithmic function. Conventional policy gradient methods predominantly use the natural logarithm base $e \approx 2.718$ (i.e., $b = e$). This paper seeks to answer a pivotal question: *How do different logarithmic bases in the policy gradient formula (as shown in Eq. (9)) affect overall performance?* To answer this question, we scrutinize three commonly used bases in mathematical analysis—*loge*, *log2*, and *log10*, and examine their gradients. The gradient of the logarithmic function is:

$$\nabla_\theta \log_b \pi_\theta\left(a_t \mid s_t\right) = \frac{1}{\pi_\theta\left(a_t \mid s_t\right) \ln(b)} \nabla_\theta \pi_\theta\left(a_t \mid s_t\right) \quad (10)$$

Then, Eq. (9) can be written as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}^{\pi_\theta}\left[\sum_{t=0}^{\infty} A^{\pi_\theta}\left(s_t, a_t\right) \frac{\nabla_\theta \pi_\theta\left(a_t \mid s_t\right)}{\pi_\theta\left(a_t \mid s_t\right) \ln(b)}\right] \quad (11)$$

Eq. (10) shows that although the probability of $\pi_\theta\left(a_t \mid s_t\right)$ is equal in $\log_e \pi_\theta\left(a_t \mid s_t\right), \log_2 \pi_\theta\left(a_t \mid s_t\right), \log_{10} \pi_\theta\left(a_t \mid s_t\right)$, there exists a constant $\ln(b)$ in $\nabla_\theta \log_b \pi_\theta\left(a_t \mid s_t\right)$. This leads to a different learning rate for these three bases.

Fig. 1 shows these logarithmic functions with different bases and the gradients of these logarithmic functions. Analyzing Fig. 1, we can find that although the gradients of $log_e\pi_\theta\left(a_t \mid s_t\right)$, $log_2\pi_\theta\left(a_t \mid s_t\right)$, and $log_{10}\pi_\theta\left(a_t \mid s_t\right)$ have the same varying tendency, their values vary wildly. This undoubtedly affects the policy gradients and the final performances of the policy after multi-step updates.

Furthermore, although $log_e x$, $log_2 x$, and $log_{10} x$ have the same tendency, the function values vary significantly for the same x. In most cases, the probability of taking a good action, $\pi(a_{good} \mid s)$, is higher than that of taking a wrong action, $\pi(a_{bad} \mid s)$. This means that if we take the logarithm of these probabilities, the absolute value of $\log \pi(a_{good} \mid s)$ will be smaller than that of $\log \pi(a_{bad} \mid s)$. When $\pi(a_{good} \mid s)$ is close to 1, the absolute value of $\log \pi(a_{good} \mid s)$ is very small, almost 0. This means that the contribution of a good action sample to the gradient update is minor. On the other hand, when $\pi(a_{bad} \mid s)$ is small, the absolute value of $\log \pi(a_{bad} \mid s)$ is more significant than that of $\log \pi(a_{good} \mid s)$. This means that the contribution of a lousy action sample to the gradient update is more significant than that of a good action sample. To solve this problem, we can modify the policy gradient formula of $\log \pi(a \mid s)$.

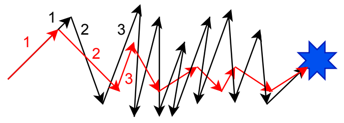

Fig. 2. Schematic diagram of multi-step gradient updates with different bases

Note that the probability of $\pi_\theta\left(a_t \mid s_t\right)$ is not equivalent in $\log_e \pi_\theta\left(a_t \mid s_t\right)$, $\log_2 \pi_\theta\left(a_t \mid s_t\right)$, and $\log_{10} \pi_\theta\left(a_t \mid s_t\right)$. Fig. 2 illustrates a diagram of multi-step gradient updates with different bases. The red and black lines represent the update trajectories of $\log_e \pi_\theta\left(a_t \mid s_t\right)$ and $\log_{10} \pi_\theta\left(a_t \mid s_t\right)$, respectively. As shown in Fig. 2, even if the directions of the first gradient steps in $\log_e \pi_\theta\left(a_t \mid s_t\right)$ and $\log_{10} \pi_\theta\left(a_t \mid s_t\right)$ are the same, there is a constant update size due to the logarithmic function gradient with different bases. In the second update process, the vector sums of the gradients in two consecutive steps differ. Thus, after many gradient updates, the policy parameter of $\log_e \pi_\theta\left(a_t \mid s_t\right)$ and $\log_{10} \pi_\theta\left(a_t \mid s_t\right)$ will be different. Moreover, as shown in Eq. (10), the policy parameter differences superpose with the influence of $ln(b)$, leading to a non-negligible effect on policy gradient.

We can conclude that: *The logarithmic function with different bases in the policy gradient has different performances. Thus, the logarithmic function matters in policy gradient deep RL.* Notably, while $loge$ and $log2$ generally facilitate faster learning, $log10$ exhibits superior stability. These insights guide profound RL practitioners in tailoring log functions to specific challenges.

Based on the above insight, we propose the Logarithmic Basis Policy Gradient (LBPG) algorithm, which means the logarithmic function with different bases in the policy gradient. LBPG can be applied to various policy gradient algorithms. We write the general expression of the policy gradient with LBPG:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}^{\pi_\theta}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log_b \pi_\theta\left(a_t \mid s_t\right)\right] \quad (12)$$

Where $b = [e, 2, 10]$. Specifically, when Ψ_t denotes different values, we get:

- $\Psi_t = \sum_{t'=t}^{\infty} r_{t'}$: LB-REINFORCE (LBREINFORCE).
- $\Psi_t = Q^{\pi_\theta}(s_t, a_t)$ approximated by neural network: LB-Actor-Critic (LBAC).
- $\Psi_t = A^{\pi_\theta}(s_t, a_t)$: LB-Advantage Actor-critic (LBA2C).
- $\Psi_t = A_t^{\mathrm{GAE}(\gamma,\lambda)}$: LB-GAE (LBGAE)

LBPG algorithm is summarized in Algorithm 1.

Furthermore, we propose the Adaptive LBPG algorithm to capitalize on the insights regarding the distinct impacts of various logarithmic bases on policy gradient RL performance. Adaptive LBPG algorithm firstly calculates returns $\mathcal{R}$ using rewards, then computes the return's mean $\mu_{\mathcal{R}}$ and the return's variance $\sigma_{\mathcal{R}}^2$. We define return's variance-mean ratio $\rho_{\mathcal{R}}$:

$$\rho_{\mathcal{R}} = \frac{\sigma_{\mathcal{R}}^2}{\mu_{\mathcal{R}}} \tag{13}$$

Adaptive LBPG algorithm determines the log base b with segmentation function, comparing $\rho_{\mathcal{R}}$ with low threshold τ_{low}, mid threshold τ_{mid} and high threshold τ_{high}:

$$b = \begin{cases} 2, & \text{if } \rho_{\mathcal{R}} < \tau_{low}, \\ e, & \text{if } \tau_{mid} \leq \rho_{\mathcal{R}} < \tau_{high}, \\ 10, & \text{otherwise.} \end{cases} \tag{14}$$

A high $\rho_{\mathcal{R}}$ means a significant return's relative variance, representing that the policy indicates instability. Thus, we must choose a lower log base to stabilize the learning process. For example, Adaptive LBPG may lean towards a lower base, such as $log2$, in environments with a lower return variance-mean ratio, signaling more stable learning conditions. On the contrary, a low $\rho_{\mathcal{R}}$ requires a higher base like $log10$ that would increase the learning speed.

Adaptive LBPG shifts between logarithmic bases during training, separating returns into positive and negative categories to calculate mean and variance independently. Algorithm 2 summarizes the process.

Algorithm 1 LBPG: Log-Bases Policy Gradient

```
Parameters: total training epoch $L$; learning rate $\alpha$, $\beta$
1: Initialize policy network parameters $\theta$ and value function network parameters $\omega$
2: for $l = 1, \ldots, L$ do
3:     while not done do
4:         Observe state $s_t$, get action $a_t$ from the policy network
5:         Receive $s_{t+1}$ and reward $r_t$ from the environment
6:         Append $(s_t, a_t, r_t, s_{t+1})$ to the transition-sample list
7:     end while
8:     Update value network (by TD residual, advantage value, etc.): For example, TD residual: $\delta_\omega = \frac{1}{M}\sum_m \nabla_\omega \mathbb{E}^{\pi_\theta}[(r_t + Q(s_{t+1}, a) - Q(s_t, a_t))]^2$ $\omega \leftarrow \omega + \beta_t \delta_\omega$
9:     Update the policy network: $\nabla_\theta J(\pi_\theta) = \mathbb{E}^{\pi_\theta}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log_b \pi_\theta(a_t \mid s_t)\right]$ with $b = [2, e, 10]$, where $\Psi_t$ the advantage function and value function, etc. $\theta \leftarrow \theta + \beta \nabla_\theta J(\pi_\theta)$
10: end for
Output: policy and critic parameters $\theta, \omega$
```

Fig. 3. Two game environments in OpenAI gym: Cartpole and Lunarlander

IV. Experiments and Analysis

This section conducted a series of experiments to validate the efficiency of the proposed methods. The test environments are the widely recognized control benchmark environments in OpenAI Gym [18]. We applied LBPG and Adaptive LBPG to several prominent policy gradient deep RL algorithms, including REINFORCE, Actor-Critic, A2C, and GAE, to verify the efficiency of the proposed methods.

In our game environment (Fig.3), we have used the algorithms mentioned earlier to demonstrate the difference in their performance. The first environment, Cartpole (a), is based on the classic cart-pole problem, described by Barto, Sutton, and Anderson in Literature [19]. A pendulum is attached to a cart that moves on a frictionless track. The goal is to balance the pole by applying forces left and right on the cart. The Lunarlander (b) environment is a rocket trajectory optimization problem with discrete actions of turning the engine on or off. The landing pad is always located at specific coordinates. The agent has infinite fuel and can learn to fly and land.

Experimental results in Fig. 4 demonstrate the influence of varying logarithmic bases when used in conjunction with the SGD optimizer. Fig. 4 shows that *loge* and *log2*, facilitate faster learning in most of these algorithms compared to *log10*. However, *log10* exhibits superior stability, suggesting its suitability in scenarios where training stability is paramount.

The experimental results suggest that deep RL practitioners can choose either *log2* or *log10* based on their specific requirements, leading to improved performance in fluctuating RL scenarios.

Building on the initial findings, we further explore the learning dynamics using LBPG. Fig. 5 illustrates the Relationship between average return and the standard deviation-to-mean ratio of batch returns in REINFORCE, AC, A2C, and GAE algorithms. Figs. 5 (a)-(h) show that when the return variance increases, meanwhile, the return decreases. The intuitive idea is that we should decrease the learning rate to enhance stability in training. A small learning rate can achieve this; thus, $log10$ can be chosen.

The extended experiments in Fig. 6 emphasize the Adaptive LBPG method's adaptive nature. For the thresholds, We set $\tau_{low} = 0.1$, $\tau_{mid} = 0.3$ and $\tau_{high} = 0.57$. Adaptive LBPG optimizes the learning rate by dynamically adjusting the logarithmic base between *log2*, *loge*, and *log10* based on the

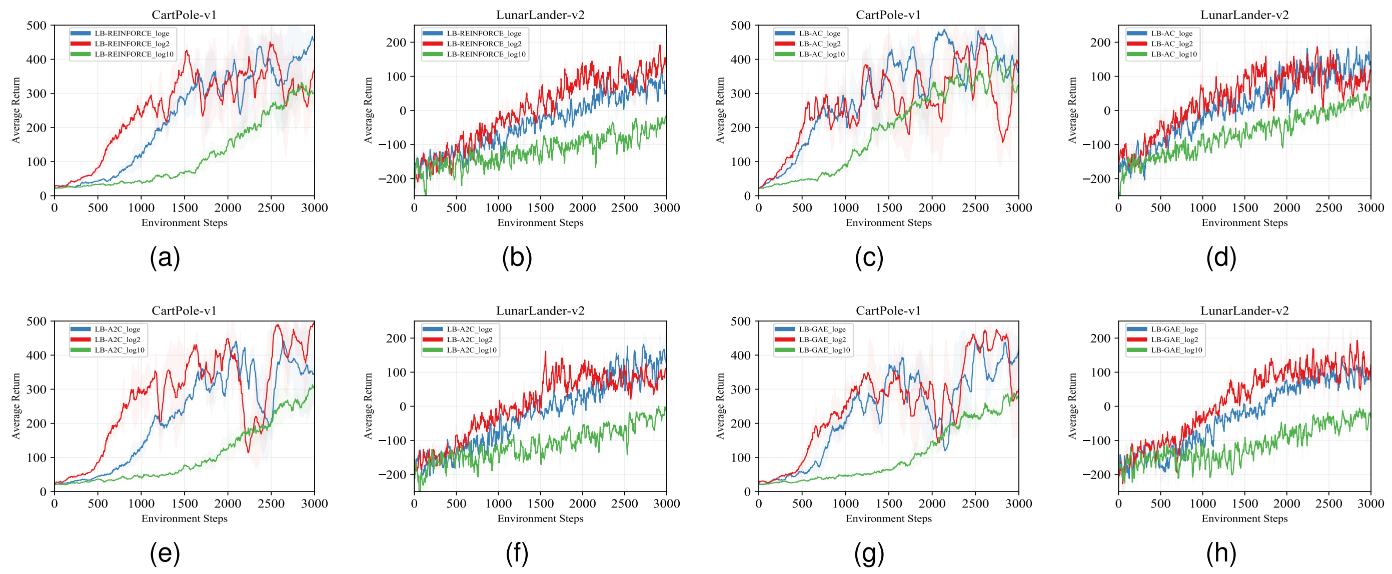

Fig. 4. Performances of LBREINFORCE, LBAC, LBA2C, and LBGAE on CartPole-v1 and LunarLander-v2. (a) LBREINFORCE-CartPole. (b) LBREINFORCE-LunarLander. (c) LBAC-CartPole. (d) LBAC-LunarLander. (e) LBA2C-CartPole. (f) LBA2C-LunarLander. (g) LBGAE-CartPole. (h) LBGAE-LunarLander.

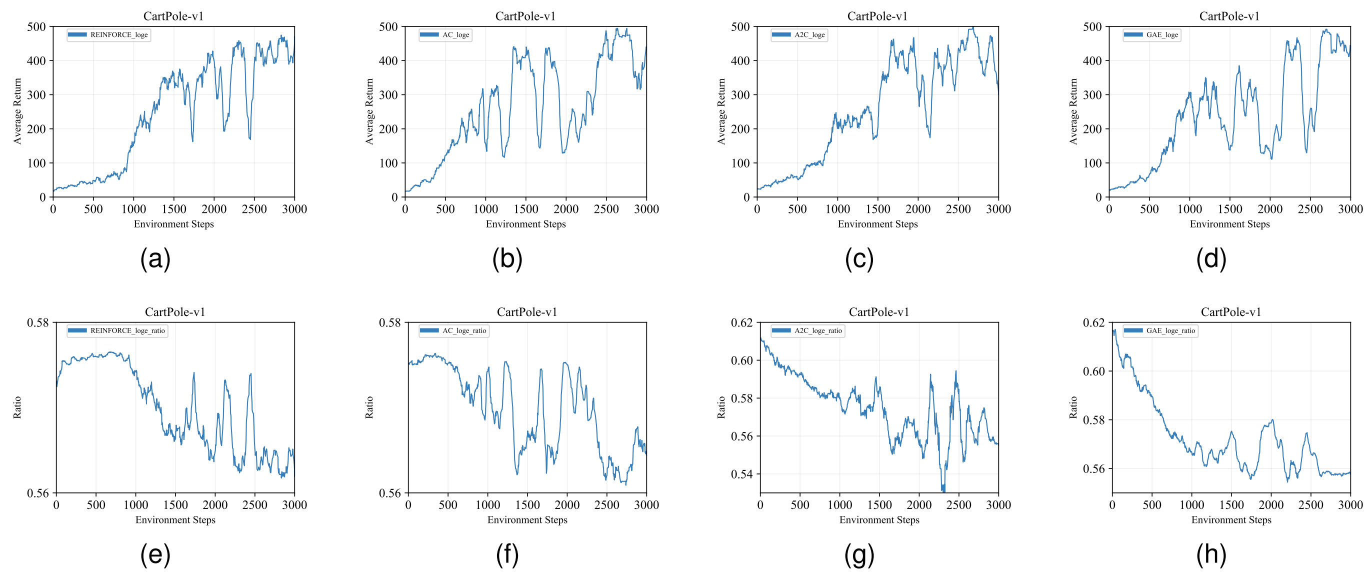

Fig. 5. Relationship of return and batch return variance on CartPole-v1. (a) (b) (c) (d) return of REINFORCE, AC, A2C, GAE. (e) (f) (g) (h) return std-mean ratio of REINFORCE, AC, A2C, and GAE

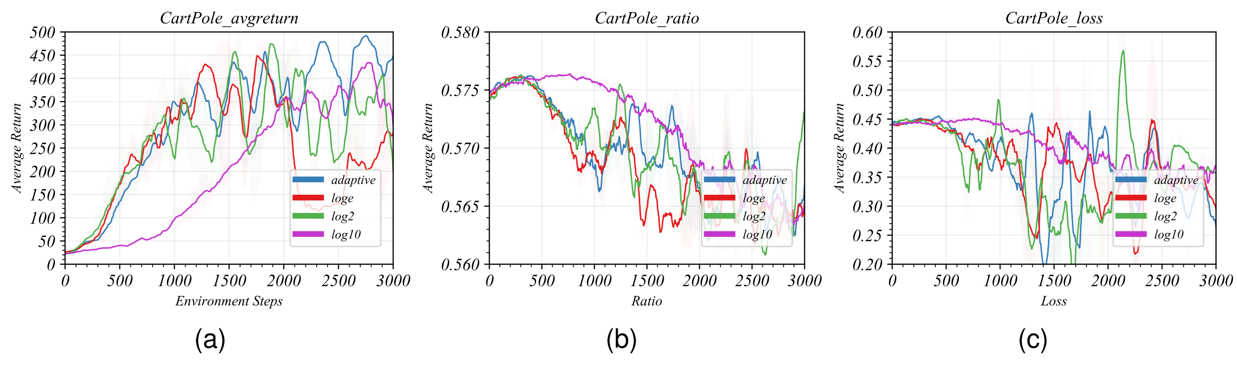

Fig. 6. Comparative analysis of return, loss, and ratio in the CartPole-v1 environment for various methods. (a) Returns using adaptive, log2, log10, and loge methods. (b) Ratio evaluations using adaptive, log2, log10, and loge methods. (c) Loss values for adaptive, log2, log10, and loge.

Algorithm 2 Adaptive LBPG: Adaptive Log-Bases Policy Gradient

Parameters: total training epochs L; learning rates α, β; low threshold τ_{low}, mid threshold τ_{mid}, high threshold τ_{high}

```
1: Initialize policy network parameters $\theta$ and value function network parameters $\omega$
2: for $l = 1, \ldots, L$ do
3:     Initialize episode reward to 0
4:     Reset environment and get initial state $s_0$
5:     while not done do
6:         Observe state $s_t$, get action $a_t$ and value estimate $v_t$ from the policy network
7:         Execute action $a_t$, receive reward $r_t$ and new state $s_{t+1}$ from the environment
8:         Store $(s_t, a_t, r_t, s_{t+1})$ in the transition sample list
9:         Update episode reward $R$ by appending $r_t$
10:    end while
11:    Compute mean $\mu_{\mathcal{R}}$ and variance $\sigma^2_{\mathcal{R}}$ for $R$
12:    Calculate returns ratio $\rho_{\mathcal{R}} = \frac{\sigma^2_{\mathcal{R}}}{\mu_{\mathcal{R}}}$
13:    if $\rho_{\mathcal{R}} < \tau_{low}$ then
14:        $b = 2$
15:    else if $\tau_{mid} \leq \rho_{\mathcal{R}} < \tau_{high}$ then
16:        $b = e$
17:    else
18:        $b = 10$
19:    end if
20:    Update value network:
21:    $\delta_\omega = \frac{1}{M} \sum_m \nabla_\omega \mathbb{E}^{\pi_\theta} [(r_t + Q(s_{t+1}, a) - Q(s_t, a_t))]^2$
22:    $\omega \leftarrow \omega + \beta \delta_\omega$
23:    Update the policy network with selected log base $b$:
24:    $\nabla_\theta J(\pi_\theta) = \mathbb{E}^{\pi_\theta} \left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log_b \pi_\theta (a_t \mid s_t)\right]$
25:    $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$
26: end for
```

Output: Optimized policy and critic parameters θ, ω

comparison variance-to-mean ratio of returns and thresholds. This adaptability is evident in Fig. 6, where Adaptive LBPG outperforms other methods regarding return, stability, and loss. Figs. 6 (a)-(c) show the average return, ratio, and loss in the CartPole-v1 environment.

Limitation: LBPG has a stronger impact on policy gradient methods with SGD optimizer than Adam optimizer. This could be due to the momentum characteristics of Adam's optimizer. Choosing the right optimizer with LBPG for maximum efficacy in RL scenarios is important.

V. Conclusions

This paper studied the influence of logarithmic functions with different bases (*loge*, *log2*, and *log10*) in policy gradient deep RL methods. We analyze the role of these logarithmic functions and their gradients when using SGD as the optimizer in deep RL. We proposed the LBPG and Adaptive LBPG algorithms to enhance policy gradient methods. The experimental results showed that the choice of the logarithmic base influences the policy gradient methods. *loge* and *log2* are generally more effective than *log10* in deep RL algorithms, but *log10* exhibits more stability. Furthermore, experimental results showed the importance of choosing appropriate logarithmic functions for policy gradient RL methods and highlighted Adaptive LBPG's potential to refine learning in deep RL tasks. For future work, we will study to apply Adaptive LBPG to multi-agent RL problems.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] Q. Liu, Y. Li, S. Chen, K. Lin, X. Shi, and Y. Lou, "Distributional reinforcement learning with epistemic and aleatoric uncertainty estimation," *Information Sciences*, vol. 644, p. 119217, 2023.

[3] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1861–1870.

[4] Q. Liu, Y. Li, and L. Liu, "A 3d simulation environment and navigation approach for robot navigation via deep reinforcement learning in dense pedestrian environment," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2020, pp. 1514–1519.

[5] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4295–4304.

[6] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

[7] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, vol. 12, 1999.

[8] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7-9, pp. 1180–1190, 2008.

[9] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2016, pp. 1928–1937.

[10] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*. PMLR, 2015, pp. 1889–1897.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[13] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *International Conference on Learning Representations*, 2016.

[14] N. Vlassis, A. Chandrashekar, F. Amat, and N. Kallus, "Control variates for slate off-policy evaluation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 3667–3679, 2021.

[15] M. Uehara, C. Shi, and N. Kallus, "A review of off-policy evaluation in reinforcement learning," *arXiv preprint arXiv:2212.06355*, 2022.

[16] M. Dann, F. Zambetta, and J. Thangarajah, "Exploration in continuous control tasks via continually parameterized skills," *IEEE Transactions on Games*, vol. 10, no. 4, pp. 390–399, 2018.

[17] M. Hendrix, T. Bellamy-Wood, S. McKay, V. Bloom, and I. Dunwell, "Implementing adaptive game difficulty balancing in serious games," *IEEE Transactions on Games*, vol. 11, no. 4, pp. 320–327, 2019.

[18] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[19] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983.