Improving Value Estimation Critically Enhances Vanilla Policy Gradient

Tao Wang
UCSD
taw003@ucsd.edu

Sicun Gao
UCSD
sicung@ucsd.edu

Abstract

Modern policy gradient algorithms, such as TRPO and PPO, outperform vanilla policy gradient in many RL tasks. Questioning the common belief that enforcing approximate trust regions leads to steady policy improvement in practice, we show that the more critical factor is the enhanced value estimation accuracy from more value update steps in each iteration. To demonstrate, we show that by simply increasing the number of value update steps per iteration, vanilla policy gradient itself can achieve performance comparable to or better than PPO in all the standard continuous control benchmark environments. Importantly, this simple change to vanilla policy gradient is significantly more robust to hyperparameter choices, opening up the possibility that RL algorithms may still become more effective and easier to use.[1]

1 Introduction

Deep policy gradient methods have demonstrated their strength in various domains, including robot learning [29], game playing [37], and large language model training [22]. Policy gradient methods in its modern versions, such as Trust Region Policy Optimization (TRPO) [26], Proximal Policy Optimization (PPO) [28], have been the popular choice for delivering state-of-the-art performance across the broad range of RL tasks. In comparison, the original policy gradient algorithm, Vanilla Policy Gradient (VPG) [32], typically performs significantly worse than the modern algorithms. The common explanation for why TRPO and PPO outperform VPG is their ability to prevent excessive large policy updates. This is achieved through a surrogate objective that incorporates importance sampling, constrained by a limit on policy step size each iteration. While monotonic improvement is theoretically guaranteed for sufficiently small policy updates [15, 26], empirical results suggest otherwise: smaller learning rates for the policy network do not always yield better performance [3]. According to the trust region theory [26], there is a gap between theory and practice because the theory guarantees improvement of the original policy objective only when the surrogate objective improves by a margin greater than the distance between the old and new policies. However, there is no guarantee that this condition is consistently satisfied in practical policy training.

Moreover, it has been reported that deep policy gradient methods often suffer from implementation issues such as brittleness, poor reproducibility, and sensitivity to hyperparameter choices [11, 8, 3]. Extensive empirical analysis has shown that the behavior of deep policy gradient methods does not align with the predictions of their motivating framework [14]. Recent studies further highlight that the optimization landscape of many RL tasks is highly non-smooth and even fractal, raising questions about the well-posedness of all policy gradient methods [38]. Consequently, there is still no clear answer to the question: *while almost all theoretical assumptions are violated in practice, why does PPO perform better than vanilla policy gradient empirically?*

[1]The original version of this work is published at ICML 2025. Code available at https://github.com/taowang0/value-estimation-vpg

2nd Workshop on Aligning Reinforcement Learning Experimentalists and Theorists (ARLET 2025).

In this paper, we present a different perspective on deep policy gradient methods within the actor-critic framework: accurate value estimation is more critical than enforcing trust regions. We first demonstrate that, in practice, trust region methods do not behave as their theoretical analysis suggests. Then, we analyze how TRPO and PPO implicitly enhance value estimation through their implementation designs. Our theoretical analysis reveals that the true value often grows significantly faster than the critic's predictions, leading to poor value estimation during policy training. Finally, empirical results show that simply increasing the number of value updates enables the basic VPG algorithm to match PPO's performance across multiple continuous control benchmarks in Gymnasium. This highlights the pivotal role of value estimation in improving policy gradient methods.

This paper is organized as follows. In Section 4, we analyze the relationship between trust regions and policy improvement, demonstrating that there is no clear correlation. This suggests that enhancing a trust region may not be the fundamental reason for the success of trust region methods. In Section 5, we explain that both TRPO and PPO perform more value steps per iteration compared to VPG, fundamentally contributing to closer value approximation and improved performance. Additionally, we provide a theoretical framework suggesting that value networks typically require more gradient steps to optimize than policy networks. In Section 6, extensive experiments are conducted and presented to corroborate our theoretical analysis. Specifically, we show that by increasing the number of value steps alone, VPG achieves performance similar to or better than PPO across several Gymnasium benchmarks.

2 Related Work

Performance gap between VPG and PPO. Much work has been done to understand the performance gap between VPG and PPO. It has been shown that the VPG loss performs significantly worse than the PPO loss, even with optimally conditioned hyperparameters [3, 25]. On the theoretical side, the global optimality of TRPO/PPO is proven for overparameterized neural networks under the assumption of a finite action space [17]. It is also shown that ratio clipping may not be necessary in PPO [30]. Another theory suggests that policy gradients work by smoothing the value landscape [39], indicating that the choice of policy objective may not be a critical factor affecting performance, as all of them provide the same smoothing effect through Gaussian kernels. In this work, we take a step forward to demonstrate that the fundamental gap between VPG and PPO lies in value estimation, and by optimizing it, VPG can achieve performance comparable to PPO.

Value estimation in off-policy methods. Although it has been demonstrated that an optimal baseline in policy gradients can effectively reduce the variance of the policy gradient estimator [24], it remains unclear to what extent value estimation affects the performance of on-policy algorithms. In contrast, the importance of value estimation has been extensively studied in off-policy algorithms. In particular, it has been found that the $\max$ operator may lead to overestimated Q values, and double Q learning methods were introduced to mitigate this error in estimation [35, 36]. This method also serves as one of the keystones in advanced off-policy algorithms such as TD3 [9] and SAC [10], compared to the DDPG algorithm [16]. Additionally, it has been numerically shown that using state-action-dependent baselines does not reduce variance compared to state-dependent baselines in benchmark environments [33].

Implementation matters in deep policy gradients. Despite its accomplishments, deep RL methods are notorious for their brittleness, lack of stability and reliability [11, 8]. Furthermore, it has been reported that the performance of PPO heavily relies on code-level optimization techniques, due to its sensitivity to hyperparameter choices [11, 8] and regularization techniques [18]. Recent works aimed at systematically understanding and addressing hyperparameter tuning issues in deep RL have been developed [23, 7, 2]. We contribute to this line of research by directly identifying value estimation as the core component that makes deep policy gradients work.

3 Preliminaries

Consider an infinite-horizon Markov decision process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \Phi, r, \rho_0, \gamma)$, where the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are continuous and compact, $\Phi : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, \infty)$ is the transition probability density function, $R : \mathcal{S} \to \mathbb{R}$ is the reward function, ρ_0 is the distribution

of the initial state s_0, and $\gamma \in (0, 1)$ is the discount factor. For a parameterized stochastic policy π_θ, the policy objective to maximize is given by

$$J(\theta) = \hat{\mathbb{E}}_{(s_t,a_t)\sim\pi_\theta, s_0\in\rho_0}\Big[\sum_{t=0}^{\infty}\gamma^t R(s_t, a_t)\Big], \tag{1}$$

where $\theta \in \mathbb{R}^N$ is the policy parameter. According to [32], the original form of policy gradient estimator is given as:

$$\nabla_\theta J(\theta) \propto \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} Q^\pi(s, a)\nabla_\theta \pi_\theta(a|s)\, \mathrm{d}a\mathrm{d}s, \tag{2}$$

where Q^π is the Q-function of the current policy π and ρ^π the discounted visitation frequencies. While there are various methods for approximating Q values, the most commonly used approach is Monte-Carlo, which estimates the discounted return for each trajectory, i.e., $\hat{G}_t = \sum_{k=t}^{T}\gamma^{k-t}R_k$. The vanilla policy gradient loss is given as

$$L(\theta) = \hat{\mathbb{E}}_{(s_t,a_t)\sim\pi_\theta}\Big[\log \pi_\theta(a_t|s_t)\ \hat{G}_t\Big], \tag{3}$$

which is the REINFORCE algorithm [41].

REINFORCE with baseline. In practice, the variance of the estimation $\hat{Q}_t$ can be large, making it difficult to distinguish between higher-valued actions and less highly valued ones. This motivates the use of baseline functions to reduce the variance [31]. In this case, the objective is given by:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_{(s_t,a_t)\sim\pi_\theta}\Big[\log \pi_\theta(a_t|s_t)\ \hat{A}_t\Big], \tag{4}$$

where $\hat{A}_t = \hat{Q}_t - \hat{V}(s_t)$ is the estimated advantage of (s_t, a_t), and $\hat{V}$ is the estimated value function of the current policy π which serves as the baseline. Let $V_\phi(s) = V(s; \phi)$ denote the value approximation to the true value function V^{π_θ} by a neural network where $\phi \in \mathbb{R}^M$ is the network parameter.

In some literature, this algorithm is also referred to as Advantage Actor-critic (A2C) [21]). To avoid any potential ambiguity, we henceforth refer to the objective in equation (4) whenever mentioning the vanilla policy gradient (VPG) algorithm. In the following sections, we will demonstrate that optimizing the objective in equation (4) is sufficient to achieve performance comparable to PPO, provided that the baseline function $\hat{V}$ is properly optimized.

Trust region methods. According to the conservative policy iteration method [15], a small policy update that improves the corresponding surrogate objective is guaranteed to improve the true policy objective. This result motivated the TRPO algorithm [26], which optimizes the following surrogate objective:

$$L^{TRPO}(\theta) = \hat{\mathbb{E}}_{(s_t,a_t)\sim\pi_\theta}\Big[\frac{\pi_\theta(a_t|s_t)}{\pi(a_t|s_t)}\hat{A}_t\Big], \tag{5}$$

which is further subject to a constraint on the KL divergence between the current and the old policy:

$$\hat{\mathbb{E}}_{s\sim\pi}\Big[D_{KL}(\pi_\theta(\cdot|s)\ \|\ \pi(\cdot|s))\Big] \le \delta, \tag{6}$$

for some $\delta > 0$. In practice, however, TRPO can be expensive, as it involves the computation of second-order derivatives. This motivates the PPO algorithm [28], which simplifies the objective and avoids the need for second-order derivatives:

$$L^{PPO}(\theta) = \hat{\mathbb{E}}_{\pi}\Big[\min\Big(r_t\hat{A}_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon)\hat{A}_t\Big)\Big], \tag{7}$$

where $r_t = \frac{\pi_\theta}{\pi}$ is the probability ratio from importance sampling, and ϵ is the clipping parameter.

4 Revisiting Trust Regions

According to the theoretical analysis in TRPO [26], the true performance $J(\cdot)$ is guaranteed to improve when a policy update is made to improve the TRPO surrogate $L^{TRPO}(\cdot)$, provided that the KL divergence between two consecutive policies is small enough. This condition is expressed through the following inequality:

$$J(\theta) \geq L^{TRPO}(\theta) - \frac{4\beta\gamma}{(1-\gamma)^2} D_{KL}^{max}(\pi_\theta \parallel \pi_{\theta_{old}}) \tag{8}$$

where $\beta = \max_{s,a} |A_\pi(s,a)|$ is the maximum advantage value. In practice, the TRPO algorithm solves the following approximate constrained optimization problem [1]:

$$\begin{aligned} &\underset{\theta}{\text{maximize}}\ \nabla_\theta L^{TRPO}(\theta)\Big|_{\theta=\theta_{old}} \cdot (\theta - \theta_{old}) \\ &\text{subject to}\ \frac{1}{2}(\theta - \theta_{old})^T H (\theta - \theta_{old}) \leq \delta \end{aligned}$$

where H is the Hessian of $\hat{\mathbb{E}}_{s\sim\pi}\Big[D_{KL}(\pi_\theta(\cdot|s)|\pi(\cdot|s))\Big]$, which provides a quadratic approximation to the KL constraint. Solving with conjugate gradient methods leading to the update rule $\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{g^T H g}} H^{-1} g$, where $g = \nabla_\theta L^{TRPO}(\theta)|_{\theta=\theta_k}$ denotes the gradient at $\theta = \theta_k$, $\alpha \in (0,1)$ is the backtracking coefficient, and j is the smallest nonnegative integer that makes the update satisfy the KL constraint (6) while producing a positive surrogate advantage.

However, there are two issues with the design in practice. First, although it is proven that $\nabla_\theta L^{TRPO}(\theta)|_{\theta=\theta_k} = \nabla_\theta J(\theta)|_{\theta=\theta_k}$ under conditions that *assume* the existence of gradients [26], the gradient $\nabla_\theta J(\theta)$ may not exist for many continuous control problems in the first place. This is because the policy optimization landscapes in these cases often exhibit fractal structures, as illustrated in Figure 1 [38]. Consequently, the linear approximation used in the TRPO implementation may poorly represent the original objective, making it difficult to attribute the success of TRPO to this approximation.

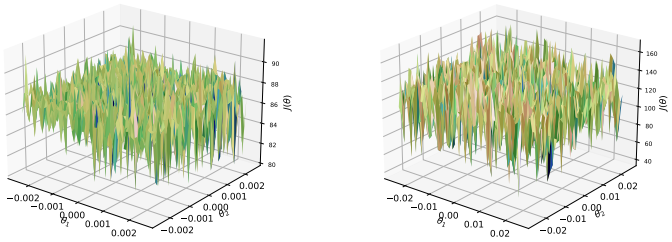

Figure 1: Policy objectives in many continuous-control environments are highly non-smooth and fractal. Left: Hopper-v3; Right: Walker2d-v3.

Second, since the approximate objective is linear in θ and the quadratic constraint induces a convex feasible region, the solution to this problem should always lie on the boundary of the feasible region. As a result, the practical algorithm attempts to take the *largest* possible step in each iteration, whereas the theory suggests taking the *smallest* step to guarantee policy improvement. This discrepancy suggests that the effectiveness of trust region methods may not stem solely from their enforcement of theoretical constraints, but rather from hyperparameter tuning and practical code-level optimizations that extend beyond the theoretical framework [11].

Ratio trust regions in PPO. To simplify implementation, PPO instead defines trust regions using the importance sampling ratio rather than KL divergence. However, despite the potential correlation, there is no strong correspondence between enforcing a trust region and policy improvement. Notably, during PPO training, the maximum probability ratio $\max_t r_t$ can significantly violate the trust region defined by the clipping coefficient ϵ [8]. This occurs because the gradient $\nabla_\theta r_t$ vanishes outside the trust region (e.g., when $r_t > 1 + \epsilon$) due to the clipping operator. Consequently, if an excessively large

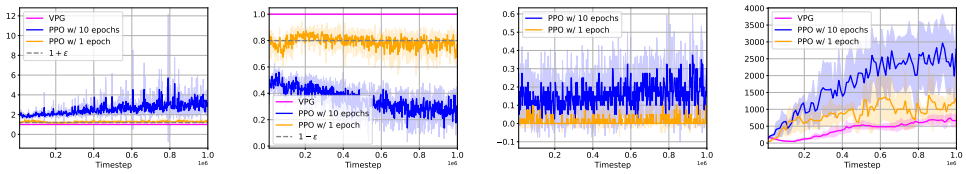

Figure 2: We compare the performance of different implementations on the MuJoCo Hopper task. The clipping parameter is set to $\epsilon = 0.2$ as default. From left to right: maximum ratio $\max r_t$; minimum ratio $\min r_t$; clipping fraction; cumulative reward.

step is taken on $\pi_\theta(a_t|s_t)$ such that r_t leaves the trust region, it may not be pulled back to the trust region anymore due to the vanishing gradient.

Our empirical results support this observation. Figure 2 (a)-(c) show that the standard PPO implementation with 10 epochs per iteration, despite employing the ratio clipping mechanism, consistently violates the ratio bound, coinciding with findings from prior work. In contrast, PPO with only 1 epoch per iteration and VPG both successfully enforce the trust region defined by the ratio bound. However, they are outperformed by the standard PPO implementation as shown in Figure 2 (d).

5 Importance of Value Estimation

In the previous section, we demonstrated that there is no direct relationship between enforcing a trust region and improving algorithmic performance. Nevertheless, trust region methods generally outperform VPG, so it remains to identify the fundamental factors that drive policy improvement in practice. Given that the number of optimization steps plays a crucial role in the outcome, we focus on analyzing this aspect throughout this section and argue that value estimation is the core factor driving policy improvement.

5.1 Increased value steps in TRPO

The VPG algorithm employs a single optimization loop to simultaneously optimize the policy and value networks using automatic differentiation software. In this loop, the value network is trained to minimize the regression loss:

$$L^V(\phi) = \|V_\phi - \hat{V}_{target}\|^2_{\mathcal{D}} \tag{9}$$

where $\mathcal{D}$ represents the collected data, V_ϕ is the parameterized value approximation, and $\hat{V}_{target}$ is the target value, typically obtained through Temporal Difference (TD) estimates. For instance, the target value is given by $\hat{V}_{target}(s_t) = \sum_{k=t}^{T} \gamma^{k-t} R_k + \gamma^{T+1-t} V_\phi(s_{T+1})$ when the GAE factor $\lambda = 1$.

However, the TRPO implementation requires separate optimization loops for the policy and value networks: the policy is optimized using the conjugate gradient algorithm to enforce the trust region constraint, while the value network is updated similarly to VPG. Perhaps surprisingly, this seemingly small modification in the code-level implementation may account for the fundamental difference between the two algorithms. Specifically, the VPG algorithm performs only one optimization step per epoch, which is insufficient for the value network to accurately estimate the true returns. In contrast, TRPO applies multiple optimization steps per iteration for the value network without over-optimizing the policy network. This results in more accurate value estimation and, consequently, better performance. Some examples are presented in Table 1.

5.2 How PPO addresses value estimation?

Having a properly optimized value network is also the key reason why PPO outperforms VPG. To understand this, we analyze two algorithmic techniques in the PPO algorithm and examine how they contribute to improved value estimation.

Mini-batching and multiple epochs. In practice, both of them contribute to increasing the number of value steps in each iteration, as we have

RL libraries	# Value steps	Learning rate
OpenAI Baselines	5	0.001
Spinning Up	80	0.001
Stable-Baseline3	10	0.001
Tianshou	20	0.001

Table 1: The number of value steps per iteration and the learning rate for value networks in default settings of several deep RL libraries [6, 1, 25, 40].

$$\textit{\# gradient steps} = \textit{\# epochs} \times \frac{\textit{full-batch size}}{\textit{mini-batch size}}$$

Therefore, if the entire batch is used with the same number of epochs, the value network will be under-approximated, leading to inaccurate baseline estimation. As shown in Figure 3 (a), PPO completely fails to find a good policy when the entire batch is used for policy training. This failure can be attributed to the poor value estimation displayed in Figure 3 (b), where the value estimation error is calculated through

$$\eta(s_0) = \sum_{k=0}^{T} \gamma^{k-t} R(s_t, a_t) - V(s_0; \phi) \tag{10}$$

where $a_t \sim \pi_\theta$ and $s_0 \sim \rho_0$. This also helps explain why PPO with a single epoch performs poorly as seen in Figure 2.

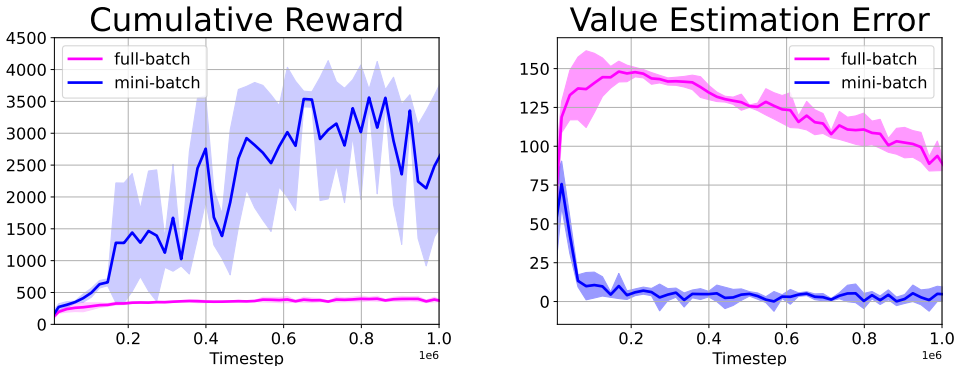

Figure 3: The cumulative reward and value estimation error during PPO training in the Hopper task are compared between full-batch and mini-batch updates. It highlights how the use of full-batch updates leads to suboptimal policy performance, as reflected in the large value estimation errors, while mini-batch updates facilitate more accurate value estimation and better cumulative reward outcomes.

Probability ratio clipping. Like VPG, PPO uses a single optimization loop for both networks as well. Therefore, while increasing the number of optimization steps improves value estimation, it may also lead to over-updating of the policy network. The clipping mechanism in PPO mitigates this by blocking excessive updates made to the policy network, allowing the value network to catch up with true returns.

5.3 Theoretical analysis

We now show that this value estimation issue can be explained from the perspective of optimization landscapes in the policy space. Briefly speaking, the policy objective changes rapidly in many continuous-control environments, which necessitates more update steps for the value network in each iteration. In dynamical systems theory, maximal Lyapunov exponents are used to study chaotic behaviors: given a dynamical system $s_{t+1} = F(s_t), s_0 \in \mathbb{R}^n$, and a small perturbation ΔZ_0 made to s_0, the divergence caused at time t is denoted by $\Delta Z(t)$. For chaotic systems, their dynamics are

sensitive to initial conditions so that it has

$$\|\Delta Z(t)\| \simeq e^{\lambda t}\|\Delta Z_0\|$$

for some $\lambda > 0$. The rigorous definition of Maximal Lyapunov exponents is presented below:

Definition 5.1. (Maximal Lyapunov exponent [19]) For the dynamical system $s_{t+1} = F(s_t), s_0 \in \mathbb{R}^n$, the maximal Lyapunov exponent $\lambda_{\max}$ at s_0 is defined as the largest value such that

$$\lambda_{\max} = \limsup_{t\to\infty} \limsup_{\|\Delta Z_0\|\to 0} \frac{1}{t} \log \frac{\|\Delta Z(t)\|}{\|\Delta Z_0\|}. \tag{11}$$

Specifically, according to Proposition C.2, we have the following estimation for the policy objective update:

$$|J(\theta') - J(\theta)| \sim \mathcal{O}(\|\theta' - \theta\|^{\alpha}) \tag{12}$$

where $\alpha = \frac{-\log\gamma}{\lambda(\theta)}$ and $\lambda(\theta)$ is the maximal Lyapunov exponent of the dynamics controlled by π_θ, which is typically positive in many continuous-control environments. Thus, we may have $\alpha < 1$ when $\lambda(\theta) > -\log\gamma$, which is generally the case when the underlying dynamics is chaotic [19]. This results in fractal landscapes in the policy space as illustrated in Figure 1 , and a rapidly changing policy objective.

The above estimation also suggests that even a small update in the parameters can lead to a significant change in the value function of the policy objective $J(\theta)$. On the other hand, the training of value networks is generally more stable. For most activation functions used in neural networks, such as $\tanh$ and ReLU, the network output is Lipschitz continuous with respect to both the input variables and the network parameters. Specifically, for a given state s and parameters ϕ and ϕ', the difference in the value network output can be estimated by

$$|V(s;\phi') - V(s;\phi)| \sim \mathcal{O}(\|\phi' - \phi\|) \tag{13}$$

when $\|\phi' - \phi\|$ is sufficiently small. This suggests that the landscape of the value regression loss in equation (9) is smooth (e.g., as illustrated in Figure 9), such that small updates in the parameters always lead to small changes in the estimated values. We summarize this analysis with the following theorem, which provides an estimate of the relationship between the updating rates of the value and policy networks:

Theorem 5.2. *Assume that the dynamics, reward function, policy and value networks are all Lipschitz continuous with respect to their input variables. Let β_1, β_2 denote the learning rate for policy and value network, respectively, and K_V denote the number of value steps per epoch. Then for each policy step, a good value estimation requires*

$$K_V \geq \frac{K_1 \beta_1^{\alpha}}{K_2 \beta_2}$$

steps made to the value network when β_1, β_2 are small. $\alpha = \frac{-\log\gamma}{\lambda(\theta)} < 1$ and $\lambda(\theta)$ is the maximal Lyapunov exponent of the dynamics controlled by π_θ, $\gamma \in (0,1)$ the discount factor and $K_1, K_2 > 0$ are constant independent of the learning rates β_1 and β_2.

This result suggests that the value network should be optimized more aggressively than the policy network, with more optimization steps and/or higher learning rates, to ensure that the value estimates made by the critic network can closely track the true values observed in rollouts. Furthermore, to prevent interference between the policy and value networks, separate networks should be used, as also suggested in [5, 12]. Additional details can be found in Appendix C.

6 Experiments

In the previous section, we theoretically demonstrated that value estimation is the core component of on-policy algorithms, and by optimizing it, a simple method is expected to perform as well as more advanced methods. In this section, we evaluate the role of value estimation across a range of continuous-control tasks from the OpenAI Gym benchmarks [4]. Specifically, we first show that the VPG algorithm, with a properly optimized value network, can achieve performance comparable to or even better than PPO with their corresponding default settings. Additionally, we conduct

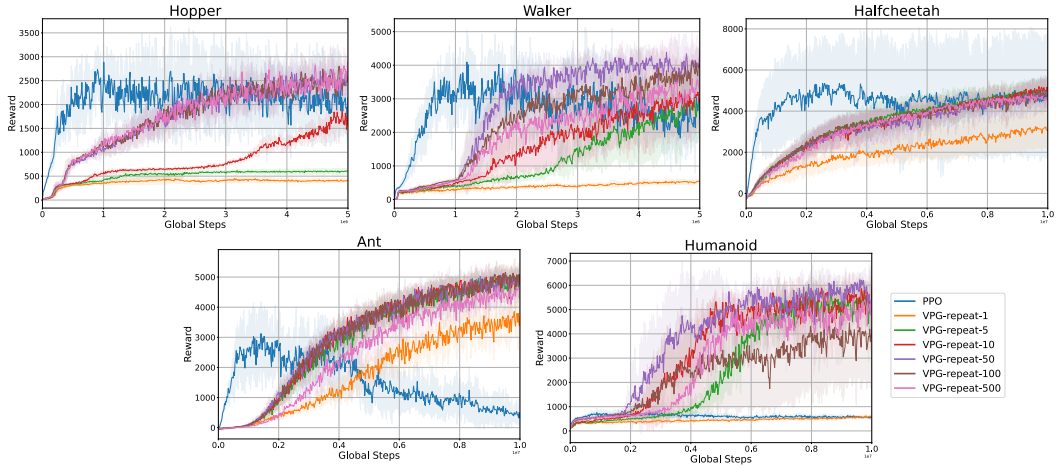

Figure 4: Training curves on Gymnasium benchmarks. The curve *VPG-repeat-k* corresponds to the vanilla policy gradient algorithm with k value steps applied each iteration. For example, *VPG-repeat-1* represents the original vanilla policy gradient implementation. As the number of value steps increases, the performance of vanilla policy gradient consistently improves, eventually converging to or outperforming PPO when the number of value steps reaches 50 or more.

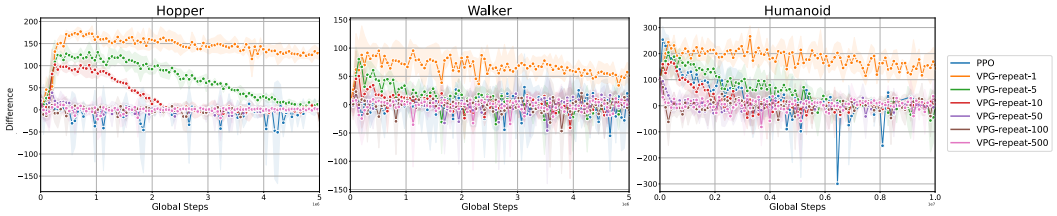

Figure 5: The corresponding value estimation difference in the experiments shown in Figure 4. We disabled exploration during evaluation, using the deterministic policy as the direct output of the policy network. The difference in value estimation is computed through Equation 10. We observe a clear correlation between the value estimation difference in VPG and its performance. As the value steps increase, the estimation error decreases and eventually oscillates around zero, leading to improved performance. More results can be found in Figure 8.

ablation studies on several code-level optimizations that may impact the performance of value estimation in both VPG and PPO. The experimental setup is detailed in Appendix A, and further experimental results can be found in Appendix B. We adapt the implementation and PPO baseline from Tianshou [40]. We also provide a single-file codebase modified from CleanRL [13] which enables implementations in other environments such as DeepMind Control [34] and Isaac Gym [20].

Vanilla policy gradient with multiple value steps. The performance of the VPG algorithm with different numbers of value steps is compared to PPO in Figure 4. The results show that by simply increasing the number of value steps, VPG can eventually achieve performance comparable to PPO in the Halfcheetah, Hopper, and Walker environments, and outperform PPO in the higher-dimensional Ant and Humanoid environments. The performance of VPG stabilizes when the number of value steps reaches 50, at which point the value network is able to accurately capture the true returns, leading to more precise advantage estimations. Full comparisons of algorithmic performance and corresponding value estimation errors for different implementations are presented in Figure 4 and Figure 5. Additionally, Figure 7 shows that VPG is more efficient than PPO in terms of policy steps. Also, VPG consistently improves the policy across all environments, while PPO's performance drops after 2M environment steps in the Walker and Ant tasks. Each task is trained with five random seeds, with solid curves representing the average return and shaded regions indicating the one standard deviation confidence interval.

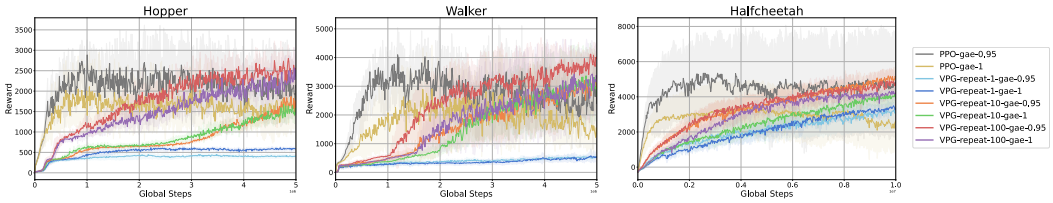

Figure 6: The influence of GAE factor λ across three tasks.

Algorithm	GAE λ	Hopper	Walker	HalfCheetah
VPG	0.95	2601.57 ± 232.14	3457.79 ± 646.70	4928.88 ± 807.04
VPG	1	2323.28 ± 436.24	3123.83 ± 987.58	4381.30 ± 222.15
PPO	0.95	1965.29 ± 478.14	2527.74 ± 507.40	4488.60 ± 2699.54
PPO	1	1611.46 ± 541.32	1431.50 ± 612.17	2604.08 ± 1237.80

Table 2: Performance of VPG and PPO with different GAE factors after 5M environment steps for Hopper and Walker, 10M environment steps for HalfCheetah.

GAE factor λ. The effect of the Generalized Advantage Estimation (GAE) factor is to reduce the variance of value estimation in on-policy algorithms [27]. To assess how the GAE factor influences the performance of VPG and PPO, we consider two cases: $\lambda = 0.95$ (default GAE factor) and $\lambda = 1$ (equivalent to Monte-Carlo). As shown in Figure 6, both the reward curves of VPG and PPO drop when $\lambda = 1$ compared to $\lambda = 0.95$, which can be attributed to the variance reduction effect of the GAE method. Notably, while the performance of VPG with 100 value steps is only slightly affected, the change in the GAE factor has a significant impact on PPO, as shown in Table 2. For example, in Walker and HalfCheetah, PPO's cumulative reward drops by more than 40% when $\lambda = 1$.

7 Conclusion

In this work, we provide both theoretical and empirical analyses to demonstrate that the performance of a policy gradient algorithm is primarily determined by how accurately the value function is estimated, rather than by any specific policy objective design adopted for policy training. Our findings suggest that deep on-policy algorithms may work for a very simple reason that even the simplest vanilla policy gradient method can accomplish.

In the meantime, we should clarify that the ratio clipping mechanism in PPO, although it does not fundamentally contribute to performance improvement, still has its strengths in that it allows multiple policy updates on a single batch, thereby enhancing sample efficiency, sometimes at the cost of reduced robustness. This point is evident in the Ant and Humanoid tasks, where the default PPO implementation updates the policy too aggressively and leads to suboptimal performances. Although our argument may, in principle, be generalizable to other domains such as large language models (LLMs), the present analysis is confined to continuous control problems, and we refrain from asserting its validity beyond this specific context.

There are two key messages we want to convey through this work: First, value estimation is perhaps the most crucial component of on-policy algorithms. By optimizing it, we can significantly improve their performance. This finding encourages us to revisit the foundations of the development of policy gradient methods (namely, from VPG to TRPO/PPO), as the core ideas in these methods (e.g., enforcing a trust region, performing multiple policy updates) may not be the fundamental underpinnings of the improved performance observed in practice. Second, our results indicate that it is possible to develop on-policy algorithms that are robust without the extensive need for hyperparameter tuning and code-level optimization techniques. For instance, VPG with multiple value steps performs well and requires very few hyperparameters to tune, suggesting that complex algorithmic architectures like PPO may not be necessary for effective policy gradient methods. This is important for deep RL in real-world applications, especially for practitioners who want to leverage policy gradient methods in their domains but have little or no expertise in RL. With well-developed parallel computing architectures that accelerate simulation and data collection, vanilla policy gradient has the potential to serve as a robust and effective alternative to PPO in various robot learning tasks.

References

[1] J. Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[2] J. Adkins, M. Bowling, and A. White. A method for evaluating hyperparameter sensitivity in reinforcement learning. In *NeurIPS*, 2024.

[3] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem. What matters for on-policy deep actor-critic methods? A large-scale study. In *ICLR*, 2021.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] K. Cobbe, J. Hilton, O. Klimov, and J. Schulman. Phasic policy gradient. In *ICML*, 2021.

[6] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. OpenAI baselines. *GitHub repository*, 2017.

[7] T. Eimer, M. Lindauer, and R. Raileanu. Hyperparameters in reinforcement learning and how to tune them. In *ICML*, 2023.

[8] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation matters in deep RL: A case study on PPO and TRPO. In *ICLR*, 2020.

[9] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *ICML*, 2018.

[10] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[11] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *AAAI*, 2018.

[12] S. Huang, R. Dossa, A. Raffin, A. Kanervisto, and W. Wang. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022.

[13] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[14] A. Ilyas, L. Engstrom, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. A closer look at deep policy gradients. In *ICLR*, 2020.

[15] S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *ICML*, 2002.

[16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[17] B. Liu, Q. Cai, Z. Yang, and Z. Wang. Neural proximal/trust region policy optimization attains globally optimal policy. In *NeurIPS*, 2019.

[18] Z. Liu, X. Li, B. Kang, and T. Darrell. Regularization matters in policy optimization - an empirical study on continuous control. In *ICLR*, 2021.

[19] E. N. Lorenz. *The Essence of Chaos*. University of Washington Press, 1995.

[20] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. Isaac Gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:1606.01540"*, 2021.

[21] 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 *ICML*, 2016.

[22] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022.

[23] S. Paul, V. Kurin, and S. Whiteson. Fast efficient hyperparameter tuning for policy gradient methods. In *NeurIPS*, 2019.

[24] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008.

[25] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[26] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *ICML*, 2015.

[27] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

[29] Y. Song, A. Romero, M. Müller, V. Koltun, and D. Scaramuzza. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 8(82):eadg1462, 2023.

[30] M. Sun, V. Kurin, G. Liu, S. Devlin, T. Qin, K. Hofmann, and S. Whiteson. You may not need ratio clipping in PPO. *arXiv preprint arXiv:2202.00079*, 2023.

[31] R. S. Sutton and A. Barto. *Reinforcement Learning: an Introduction*. MIT Press, 1998.

[32] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999.

[33] G. Tucker, S. Bhupatiraju, S. Gu, R. E. Turner, Z. Ghahramani, and S. Levine. The mirage of action-dependent baselines in reinforcement learning. In *ICML*, 2018.

[34] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, and Y. Tassa. DM control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.

[35] H. van Hasselt. Double Q-learning. In *NIPS*, 2010.

[36] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-learning. In *AAAI*, 2016.

[37] O. Vinyals, I. Babuschkin, W. Czarnecki, and et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575:350–354, 2019.

[38] T. Wang, S. Herbert, and S. Gao. Fractal landscapes in policy optimization. In *NeurIPS*, 2023.

[39] T. Wang, S. Herbert, and S. Gao. Mollification effects of policy gradient methods. In *ICML*, 2024.

[40] J. Weng, H. Chen, D. Yan, K. You, A. Duburcq, M. Zhang, Y. Su, H. Su, and J. Zhu. Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research*, 23(267):1–6, 2022.

[41] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

A Experiment Hyperparameters

	PPO	**VPG**
Num. env.	16	16
Discount factor (γ)	0.99	0.99
Num. epochs	10	1
Batch size	2048	2048
Minibatch size	64	2048
GAE factor (λ)	0.95	0.95
Optimizer	Adam	Adam
Clipping parameter (ϵ)	0.2	N/A
Advantage normalization	False	False
Observation normalization	True	True
Reward normalization	False	False
Learning rate decay	False	False
Entropy coefficient	0	0
Policy network	[64, 64]	[64, 64]
Value network	[64, 64]	[64, 64]
Activation function	$\tanh$	$\tanh$
Gradient clipping (l_2 norm)	1.0	1.0

Table 3: Default hyperparameters for policy gradient algorithms.

	Hopper	**Walker**	**HalfCheetah**	**Ant**	**Humanoid**
VPG	0.0003	0.0007	0.0007	0.0007	0.0007
PPO	0.0003	0.0003	0.0003	0.0003	0.0003

Table 4: The learning rate used for the policy and value network in each task.

B Additional Experimental Results

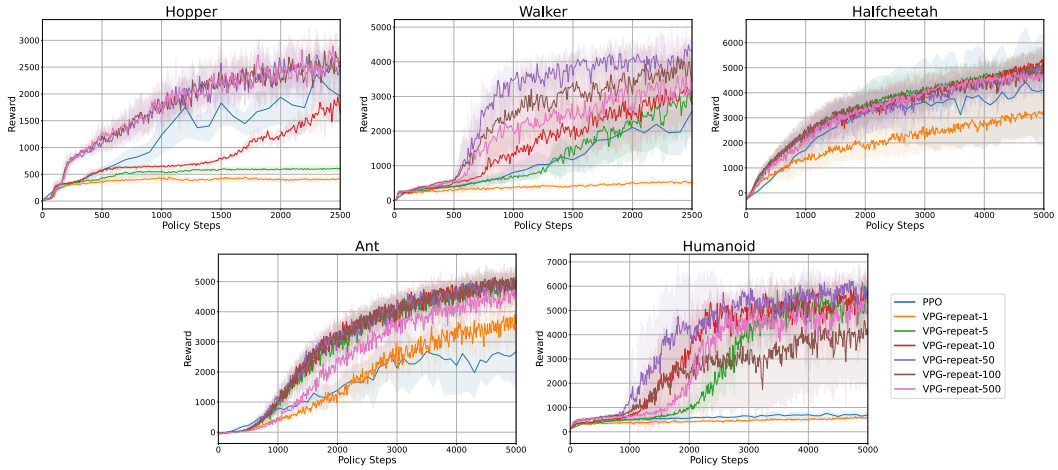

Figure 7: The cumulative rewards versus policy step in each MuJoCo task.

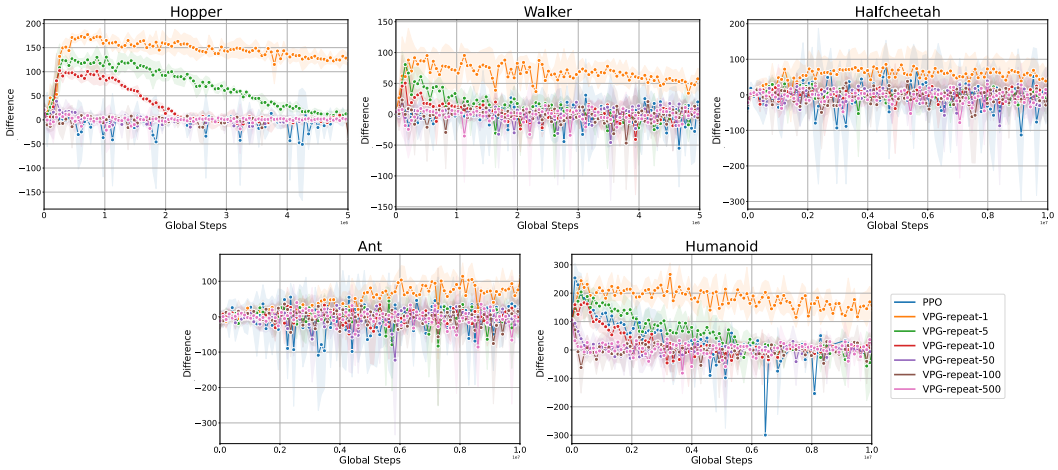

Figure 8: A full comparison of value estimation difference for each implementation.

C Theoretical Analysis of Value Estimation

Existing work has shown that the optimization landscape in the policy space is fractal in many continuous-control environments. To better characterize this behavior, we first introduce the concept of Hölder continuity:

Definition C.1. (Hölder continuity) Let $f : \mathbb{R}^{k_1} \to \mathbb{R}^{k_2}$ be a function. Given $x \in \mathbb{R}^{k_1}$ and $\alpha \in (0, 1]$, we say that f is α-Hölder continuous at x if for any $\delta > 0$, there exists $C_1 > 0$ such that

$$\|f(x') - f(x)\| \leq C_1 \|x' - x\|^{\alpha}$$

for all $x' \in \mathbb{R}^{k_1}$ that has $\|x' - x\| \leq \delta$.

Note that Hölder continuity is equivalent to Lipschitz continuity when $\alpha = 1$, and the function can be highly non-smooth and fractal when $\alpha < 1$. The following theorem establishes a connection between the chaotic behavior in an MDP and the smoothness of the corresponding policy optimization objective:

Proposition C.2. *[38] Assume that the dynamics, reward function and policy are all Lipschitz continuous with respect to their input variables. Let π_θ be a deterministic policy and $\lambda(\theta)$ denote the maximal Lyapunov exponent of the dynamics. Suppose that $\lambda(\theta) > -\log \gamma$ and let $\alpha = \frac{-\log \gamma}{\lambda(\theta)}$, then*

1. *Value function $V^{\pi_\theta}(s)$ is α-Hölder continuous in the state $s \in \mathcal{S}$;*
2. *Q-funtion $Q^{\pi_\theta}(s, a)$ is α-Hölder continuous in the action $a \in \mathcal{A}$;*
3. *Policy objective $J(\theta)$ is α-Hölder continuous in the policy parameter $\theta \in \mathbb{R}^N$.*

Proof of Theorem 5.2. According to Proposition C.2, we have the following estimation:

$$|J(\theta') - J(\theta)| \sim \mathcal{O}(\|\theta' - \theta\|^{\frac{-\log \gamma}{\lambda(\theta)}}) \tag{14}$$

where $\lambda(\theta)$ is the maximal Lyapunov exponent of the dynamics controlled by π_θ, which is typically positive in many continuous-control environments. Illustrations of fractal landscapes in several MuJoCo environments are shown in Figure 1, where even a small update in the parameters can lead to a significant change in the value of the policy objective $J(\theta)$.

Unlike policy networks, which can lead to dramatic changes in the return even with a single parameter update, the training of value networks is generally more stable, as it minimizes the regression loss:

$$L^V(\phi) = \|V_\phi - \hat{V}_{target}\|^2_{\mathcal{D}} \tag{15}$$

where $\mathcal{D}$ represents the collected data, V_ϕ is the parameterized value approximation, and $\hat{V}_{target}$ is the target value.

For most activation functions used in neural networks, such as $\tanh$ and ReLU, the network output is Lipschitz continuous with respect to both the input variables and the network parameters. Specifically, for a given state s and parameters ϕ and ϕ', the difference in the value network output can be estimated by

$$|V(s; \phi') - V(s; \phi)| \sim \mathcal{O}(\|\phi' - \phi\|) \tag{16}$$

when $\|\phi' - \phi\|$ is sufficiently small. This suggests that the landscape of the value regression loss in equation (9) is smooth (e.g., as illustrated in Figure 9), such that small updates in the parameters always lead to small changes in the estimated values.

Now, let us consider the policy improvement. Let θ_k and ϕ_k denote the parameters of the policy and value networks at the k-th iteration, respectively. Note that equation (14) suggests that for a fixed state s, the discounted return $\sum_{t=0}^{\infty} \gamma^t R_t$ is $\frac{-\log \gamma}{\lambda(\theta)}$-Hölder continuous with respect to the policy parameter. Therefore, the difference in the target value between two consecutive policy steps can be estimated by

$$|V_{target,k+1}(s) - V_{target,k}(s)| \simeq K_1 \beta_1^{\frac{-\log \gamma}{\lambda(\theta_k)}}$$

for some constant $K_1 > 0$ where β_1 is the learning rate of policy network. We can also find another constant $K_2 > 0$ such that the difference in the value network output can be estimated by

$$|V(s; \phi_{k+1}) - V(s; \phi_k)| \simeq K_2 K_V \beta_2.$$

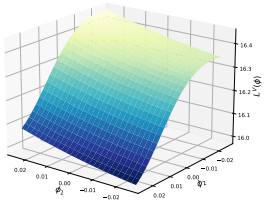

Figure 9: Value objective $L^V(\phi)$ is always smooth in the network parameter ϕ.

where K_V is the number of optimization steps for the value network and β_2 is the learning rate. Therefore, the step sizes should satisfy

$$K_V \geq \frac{K_1 \beta_1^{\alpha}}{K_2 \beta_2} \tag{17}$$

so that the value network continues to provide accurate estimates of the true return and we complete the proof.

D Monte-Carlo Value Estimation

Most on-policy algorithms, including VPG, TRPO, and PPO, estimate the return value using either Monte Carlo methods or their generalizations (e.g., GAE [27]). However, the return estimated by Monte Carlo methods can only reflect how good (or bad) the sequence of actions $\mathbf{a} = (a_0, a_1, \ldots, a_T)$ is as a whole, given the initial state $s_0 \sim \rho$. Consequently, this return may deviate significantly from the true value $V^\pi(s_0)$ due to: (a) randomness in the stochastic policy π_θ, and (b) exponential divergence of perturbed trajectories in the (potentially) chaotic dynamics.

Mollified value landscapes. It has been demonstrated that policy gradient methods work by smoothing the value landscape through a Gaussian kernel, thereby providing a valid updating direction even in the presence of fractal structures within the value landscape [39]. While the averaged return can point to a correct direction, it also implies that a single trajectory is not sufficiently informative. For instance, consider two sampled trajectories with action sequences $\mathbf{a} = (a_0, a_1, \ldots, a_T)$ and $\mathbf{a}' = (a'_0, a'_1, \ldots, a'_T)$, where $a_0 \simeq a'_0$. The resulting returns, G and G', could be entirely different despite the initial actions being nearly identical. This discrepancy arises because subsequent actions may differ. In such cases, $\hat{A}(s_0, a_0)$ and $\hat{A}(s_0, a'_0)$ could have opposite signs, even when $a_0 \simeq a'_0$. Therefore, it underscores the importance of relying on the averaged return rather than focusing too much on individual trajectories.

Understanding the Role of Baselines Theoretically, both REINFORCE (3) and VPG (4) provide the same gradient estimator since

$$\begin{aligned}
&\hat{\mathbb{E}}_{(s_t,a_t)\sim\pi_\theta}\Big[\nabla \log \pi_\theta(a_t|s_t) Q_t\Big] \\
&= \hat{\mathbb{E}}_{(s_t,a_t)\sim\pi_\theta}\Big[\nabla \log \pi_\theta(a_t|s_t)(A_t + V^\pi(s_t))\Big] \\
&= \hat{\mathbb{E}}_{(s_t,a_t)\sim\pi_\theta}\Big[\nabla \log \pi_\theta(a_t|s_t) A_t\Big]
\end{aligned}$$

under the assumption that

$$\hat{\mathbb{E}}_{a_t\sim\pi_\theta}\Big[\nabla \log \pi_\theta(a_t|s_t)\Big] = 0 \tag{18}$$

for any s_t and any probability density function π_θ. However, Equation 18 is not guaranteed in practice, where the number of samples may be insufficient to support it. Here, we demonstrate that the number of samples required to ensure the empirical mean satisfies $|\hat{\mu}| = |\nabla \log \pi_\theta(a_t|s_t)| \leq \epsilon$ becomes prohibitively large when ϵ is very small: Without loss of generality, let π_θ be a one-dimensional Gaussian distribution, i.e., $a \sim \mathcal{N}(\mu(s), \sigma^2)$. Suppose that $a_1, ..., a_K$ are i.i.d. samples obtained from π_θ. The empirical mean

$$\hat{\mu}_K = \frac{1}{K} \log \pi_\theta(a_i|s) = -\frac{1}{K} \sum_{i=1}^{K} \frac{(a_i - \mu(s))^2}{\sigma^2}.$$

This implies that $-\hat{\mu}_K \sim \chi^2(K)$. Therefore, for each state s_t, approximately $\mathcal{O}(K)$ samples are needed. This requirement results in a total of $\mathcal{O}(K^T)$ samples per iteration.

This approach is exhaustive, as it involves exploring sufficiently many actions for each state encountered at every timestep. Consequently, it converges to tabular search approaches, offering no reduction in computational complexity. However, such exhaustive exploration is clearly not the intended purpose of policy gradient methods. In fact, this process resembles the *vine TRPO* algorithm, which partially expands the tree due to computational constraints [26].

It is worth noting that we have disregarded the accumulation of errors arising from the hierarchical structure. If these errors were taken into account, the required number of samples would be even larger, but we do not elaborate further on this point. Therefore, the assumption $\hat{\mathbb{E}}_{a_t\sim\pi_\theta}\Big[\nabla \log \pi_\theta(a_t|s_t)\Big] = 0$ for any s_t is too strong to be realistically satisfied in practice. This implies that using a baseline can significantly influence the performance of the gradient estimator. Specifically, when there is no baseline value function or when the baseline function is poor, the policy gradient estimator tends to converge toward suboptimal solutions. This behavior explains the poor performance of the original VPG algorithm in Gymnasium.