An In-Depth Analysis of Advanced Policy Gradient Methods in Reinforcement Learning

By Taha Majlesi

July 17, 2025

Contents

1 The Foundations of Policy Gradient Methods

Policy gradient methods represent a cornerstone of modern reinforcement learning (RL), offering a powerful and direct approach to solving complex decision-making problems. Unlike methods that learn the value of states or state-action pairs, policy gradient algorithms directly parameterize and optimize the agent's policy. This section establishes the theoretical foundations of this approach, beginning with its place in the broader RL landscape, deriving the fundamental policy gradient theorem, and culminating in an analysis of the primary challenge that has driven decades of research in this area: the high variance of gradient estimates.

1.1 A Taxonomy of Model-Free Reinforcement Learning

Model-free reinforcement learning algorithms learn optimal behavior through direct interaction with an environment, without building an explicit model of the environment's dynamics. Within this domain, three primary families of methods can be distinguished.

- **Value-based methods:** These algorithms, such as the well-known Q-learning and its deep learning variant Deep Q-Networks (DQN), focus on learning a value function. This function, typically a state-value function $V(s)$ or an action-value function $Q(s, a)$, estimates the expected future return from a given state or state-action pair. The policy is then derived implicitly from this value function, for instance, by always selecting the action with the highest estimated Q-value (a greedy policy) [2].

- **Policy-based methods:** In contrast, policy-based methods directly learn a parameterized policy, $\pi_\theta(a|s)$, which is a probability distribution over actions given a state. The parameters θ (e.g., the weights of a neural network) are optimized directly to maximize the expected total reward. These methods do not necessarily require a value function for the policy update [2]. This direct optimization makes them particularly well-suited for high-dimensional or continuous action spaces where calculating maximum action values can be intractable [2].

- **Actor-Critic methods:** This class of algorithms represents a hybrid approach that combines the strengths of the previous two. An actor-critic agent learns both an explicit policy (the "Actor") and a value function (the "Critic"). The Actor is responsible for controlling the agent's behavior, while the Critic evaluates the actions taken by the Actor, providing a feedback signal that guides the policy updates [3]. As will be detailed, this architecture is central to modern advanced policy gradient methods.

1.2 The Policy Gradient Theorem: Direct Policy Optimization

The central objective in policy-based RL is to find the policy parameters θ that maximize an objective function, $J(\theta)$, which represents the expected cumulative reward. For an episodic task, this is the expected return from the start-state distribution. The policy gradient theorem provides a way to compute the gradient of this objective function with respect to the policy parameters, enabling optimization via gradient ascent [2, 4].

The objective function is defined as the expectation of the total reward for a trajectory τ, where trajectories are sampled according to the policy π_θ: $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[r(\tau)]$. A practical, sample-based estimate of this gradient, often used in the REINFORCE algorithm, is given by:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left(\sum_{t'=1}^{T} r(s_{i,t'}, a_{i,t'}) \right) \tag{1}$$

The intuition behind this formula is that it directly formalizes the concept of "trial and error" [5]. The agent performs a "trial" by sampling a trajectory and evaluates the "error" or outcome by calculating the total reward. The policy is then updated based on this outcome. The term $\nabla_\theta \log \pi_\theta(a|s)$, known as the score function [4], indicates the direction in parameter space that would make the sampled action more likely. This direction is then scaled by the total reward:

- If the total reward is high (a good outcome), the update moves the parameters θ to increase the probability of the actions taken. "Good stuff is made more likely."
- If the total reward is low or negative (a bad outcome), the update moves the parameters θ to decrease the probability of the actions taken. "Bad stuff is made less likely."

This process, repeated over many trials, gradually shifts the policy towards behaviors that yield higher rewards.

1.3 The Problem: High Variance in Gradient Estimates

While the policy gradient estimator is elegant and intuitive, it suffers from a critical flaw that can render it impractical for many problems: extremely high variance [2, 6]. The estimator is statistically *unbiased*, meaning its expected value is equal to the true gradient of the objective function [7]. However, the variance of the estimate for any finite number of samples can be enormous. This high variance stems from the multiple sources of randomness inherent in the RL process [7]:

1. **Stochastic Policy:** The policy $\pi_\theta(a|s)$ is often a probability distribution, meaning the agent can select different actions in the same state on different runs.
2. **Stochastic Environment Dynamics:** The environment's transition function $p(s_{t+1}|s_t, a_t)$ may be probabilistic, leading to different next states even with the same state-action pair.
3. **Stochastic Initial State Distribution:** The agent may start in different states in each episode.

These factors combine to make the total reward of a trajectory, $r(\tau)$, a highly variable quantity. Since this reward term directly scales the gradient estimate, the estimate itself becomes very noisy [8]. The consequences of high variance are severe and directly impede the learning process [2, 5]:

- **Unstable Training:** The gradient estimates can fluctuate wildly from one batch of samples to the next, leading to an unstable and inefficient optimization path [2].
- **Slow Convergence:** With a noisy gradient signal, the agent struggles to discern the true direction of improvement. A very small learning rate is often required, which dramatically slows down convergence [4, 7].

- **Poor Reproducibility:** High variance is a major contributor to the reproducibility crisis in RL. Different random seeds can lead to vastly different training outcomes [9, 10, 11].
- **Risk of Policy Collapse:** A single, unluckily sampled batch can produce a gradient estimate that is far from the true gradient, potentially pushing the policy into a very poor region of the optimization landscape [7].

Fundamentally, the high variance of the policy gradient is a direct manifestation of a poor **credit assignment** problem. The vanilla formula uses the total reward of the entire trajectory to reinforce every single action taken within that trajectory. This inefficiently dilutes the learning signal and misattributes credit [8]. Therefore, the quest to reduce variance in policy gradients is synonymous with the quest to develop more intelligent and precise credit assignment mechanisms.

2 Foundational Techniques for Variance Reduction

The primary challenge in making policy gradient methods practical is mitigating the high variance of the gradient estimator. This section details a series of foundational techniques, each designed to refine the learning signal by improving the credit assignment mechanism.

2.1 Causality and Reward-to-Go

The first and most intuitive improvement comes from applying the principle of causality. An action taken at time t can only influence rewards that occur at or after time t. To correct for this, we modify the update rule to scale the score function by the sum of rewards from that point forward, known as the "reward-to-go." The policy gradient formulation is thus updated to:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) \right) \tag{2}$$

Here, the term $\hat{Q}_{i,t} = \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})$ represents the empirical reward-to-go. This modification does not change the expected value of the gradient but significantly reduces its variance.

2.2 Temporal Discounting

Temporal discounting further refines the reward-to-go by acknowledging that immediate rewards are often more predictable and more directly related to an action than rewards far in the future [5]. By introducing a discount factor $\gamma \in [0, 1]$, we down-weight the influence of highly uncertain, distant rewards:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'}) \right) \tag{3}$$

This is the standard definition of the discounted return and is mathematically necessary for dealing with continuing (infinite-horizon) tasks.

2.3 Baselines for Variance Reduction

A powerful technique for variance reduction is the introduction of a baseline. The intuition is that the absolute magnitude of a return is less important than whether that return was better or worse than expected. By subtracting a baseline value, b, from the reward term, we can center the learning signal around zero. The policy gradient formula with a baseline is:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p_\theta(\tau_i)[r(\tau_i) - b] \tag{4}$$

A crucial property of the baseline is that, as long as it does not depend on the action, it does not introduce any bias into the gradient estimate. The expected value of the subtracted term is zero:

$$\begin{aligned}
\mathbb{E}[\nabla_\theta \log p_\theta(\tau) b] &= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) b \, d\tau \\
&= \int \nabla_\theta p_\theta(\tau) b \, d\tau \\
&= b \nabla_\theta \int p_\theta(\tau) d\tau = b \nabla_\theta 1 = 0
\end{aligned}$$

This allows us to subtract any function of the state, $b(s_t)$, from our return estimate without biasing the policy update. The theoretically optimal baseline is the state-value function itself: $b(s_t) = \mathbb{E}_\pi[G_t | S_t = s_t] = V^\pi(s_t)$. The resulting learning signal, $\hat{Q}_{i,t} - V^\pi(s_t)$, is an estimate of the **Advantage function**, which logically motivates the need for a mechanism to estimate the value function—the role of the "Critic."

3 The Actor-Critic Paradigm

Actor-Critic methods emerge as a natural and powerful extension of policy gradients with baselines. They formalize the idea of using a learned value function as a sophisticated, state-dependent baseline to achieve a dramatic reduction in variance.

3.1 The Actor and the Critic: A Division of Labor

Actor-Critic architectures are composed of two distinct but interacting components:

- **The Actor:** This component is the policy, $\pi_\theta(a|s)$, parameterized by θ. Its role is to interact with the environment, select actions, and update its parameters to improve its decision-making strategy.
- **The Critic:** This component is a value function approximator, typically the state-value function $\hat{V}^\pi_\phi(s)$, parameterized by ϕ. Its role is to "criticize" the actions taken by the Actor by evaluating the quality of the states visited, providing a high-quality, low-variance learning signal to the Actor [3].

3.2 The Advantage Function: A Superior Learning Signal

The core of the interaction between the Actor and the Critic is the **Advantage Function**, $A^\pi(s,a)$, formally defined as:

$$A^\pi(s_t,a_t) = Q^\pi(s_t,a_t) - V^\pi(s_t) \tag{5}$$

The advantage function measures how much better a specific action a_t is compared to the average action that would be taken in state s_t. Using the advantage function, the policy gradient becomes:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) A^\pi(s_{i,t},a_{i,t}) \tag{6}$$

3.3 Practical Advantage Estimation

Since the true advantage function is unknown, we estimate it using our learned Critic. A common and effective method is to leverage the relationship between the Q-function and the V-function. We can express the Q-value as the immediate reward plus the discounted value of the next state:

$$Q^\pi(s_t,a_t) \approx r(s_t,a_t) + \gamma V^\pi(s_{t+1}) \tag{7}$$

Substituting this into the definition of the advantage function, we get a practical estimate using only our learned Critic, $\hat{V}^\pi_\phi(s)$:

$$\hat{A}^\pi(s_t,a_t) \approx r(s_t,a_t) + \gamma \hat{V}^\pi_\phi(s_{t+1}) - \hat{V}^\pi_\phi(s_t) \tag{8}$$

This expression is known as the **Temporal Difference (TD) error**. It provides a way to compute a low-variance estimate of the advantage using only a single learned state-value function.

3.4 Training the Critic: Policy Evaluation

The Critic, $\hat{V}^\pi_\phi(s)$, must be trained to accurately predict the expected return from a given state. This is a policy evaluation problem, which can be approached in several ways that highlight the fundamental bias-variance trade-off.

- **Monte Carlo (MC) Evaluation:** The Critic is trained via supervised regression where the target label is the full, empirical discounted return from that state: $y_{i,t} = \sum_{t'=t}^{T} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'})$. This target is an *unbiased* estimate of $V^\pi(s_t)$ but suffers from *high variance.*
- **Bootstrap (TD) Evaluation:** A more common approach is to use bootstrapping, where the target value is constructed from the immediate reward plus the discounted value of the next state, as estimated by the Critic itself: $y_{i,t} = r(s_{i,t}, a_{i,t}) + \gamma \hat{V}^\pi_\phi(s_{i,t+1})$. This target is *biased* but has significantly *lower variance* and allows for online learning.

In both cases, the Critic's parameters ϕ are typically updated by minimizing the mean squared error: $\mathcal{L}(\phi) = \frac{1}{2}\sum_i ||\hat{V}^\pi_\phi(s_i) - y_i||^2$.

3.5 Actor-Critic Algorithm and Design Choices

The general Actor-Critic algorithm can be implemented in two main flavors: **batch** and **online**. Furthermore, there are two primary architectural designs for the Actor and Critic networks:

- **Two Separate Networks:** The Actor and Critic are independent neural networks. This is simple and stable but may be less data-efficient.
- **Shared Network:** A single network with a shared feature extractor processes the state, followed by two separate "heads"—one for the policy output (Actor) and one for the value output (Critic) [24, 1]. This can be more sample-efficient but can introduce more complex training dynamics [24].

The following table summarizes the evolution of the learning signal in policy gradient methods.

Method	**Learning Signal (Weighting Term)**	**Bias**	**Relative Variance**
REINFORCE	Total Trajectory Return: $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$	Unbiased	High
REINFORCE with Baseline	Advantage over average: $G_t - b(s_t)$	Unbiased	Medium
Actor-Critic	Advantage Estimate (TD Error): $\hat{A}_t = r_t + \gamma \hat{V}_\phi(s_{t+1}) - \hat{V}_\phi(s_t)$	Biased	Low

Table 1: Evolution of the learning signal, highlighting the bias-variance trade-off.

4 Proximal Policy Optimization (PPO)

While Actor-Critic methods effectively address high variance, they do not inherently solve the instability caused by excessively large policy updates. Proximal Policy Optimization (PPO) is a state-of-the-art algorithm designed specifically to address this instability by constraining how much the policy can change at each iteration [13, 14].

4.1 The Peril of Large Policy Updates

The central question motivating PPO is: how can we take the biggest possible improvement step on a policy without causing performance collapse? [19]. In standard on-policy methods, data is collected, an update is performed, and then the old data is discarded [15]. This is because after a significant policy update, the old data is no longer representative, and using it would introduce a harmful bias [12]. This process is highly sample-inefficient [12]. The core problem is the extreme sensitivity to the step size.

- If the step size is too small, learning is stable but painfully slow [16].
- If the step size is too large, a single bad batch can lead to a destructive update, a catastrophic failure [7, 12].

4.2 Trust Region Methods and the Motivation for PPO

Trust Region Policy Optimization (TRPO) was a landmark algorithm that provided a principled solution to this problem [17]. TRPO's core idea is to maximize performance subject to a constraint on how much the policy is allowed to change, measured by the Kullback-Leibler (KL) divergence [12]. The TRPO objective can be conceptualized as:

$$\text{Maximize } \mathcal{L}(\theta) \text{ subject to } D_{KL}(\pi_{\theta_{new}}(.|s)||\pi_{\theta_{old}}(.|s)) \leq \delta \tag{9}$$

This ensures the new policy stays within a "trust region" of the old policy [17]. However, TRPO's main drawback is its complexity and computational cost, as it requires second-order optimization [13, 17]. PPO was developed to capture the benefits of TRPO using only first-order optimization, making it far simpler and more general [14, 18].

4.3 The PPO-Clip Algorithm

The most popular variant of PPO, PPO-Clip, replaces the hard KL constraint with a novel clipped surrogate objective function [19]. First, we define the **probability ratio** between the new and old policies [13, 20]:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{10}$$

The PPO-Clip objective function is then defined as [13, 14]:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{11}$$

Here, $\hat{A}_t$ is the advantage estimate, and ϵ is a small hyperparameter (e.g., 0.2) that defines the clipping range [21]. The clipping mechanism works as follows [22, 23]:

- **When Advantage $\hat{A}_t > 0$:** The objective becomes $\min(r_t(\theta)\hat{A}_t, (1+\epsilon)\hat{A}_t)$. This creates a ceiling, removing the incentive for overly large policy updates once the ratio exceeds $1+\epsilon$.
- **When Advantage $\hat{A}_t < 0$:** The objective becomes $\max(r_t(\theta)\hat{A}_t, (1-\epsilon)\hat{A}_t)$. This creates a floor, limiting the penalty even if the policy ratio becomes very small.

4.4 The Full PPO Algorithm and Objective

In practice, the PPO algorithm optimizes a combined objective function that includes the policy loss, a value function loss, and an entropy bonus to encourage exploration [14, 20, 24]. The full objective is:

$$L(\theta) = \hat{\mathbb{E}}_t \left[L^{CLIP}(\theta) - c_1 L^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right] \tag{12}$$

Where $L^{VF}(\theta) = (\hat{V}_\phi(s_t) - V_t^{targ})^2$ is the squared-error loss for the Critic, and $S[\pi_\theta](s_t)$ is an entropy bonus. A key feature of PPO is that it can perform multiple epochs of minibatch updates on the same batch of data, significantly improving sample efficiency [14].

5 Synthesis and Conclusion

The study of advanced policy gradient methods reveals a clear and compelling intellectual narrative of identifying fundamental challenges—high variance and update instability—and systematically developing more sophisticated techniques to overcome them.

5.1 The Evolutionary Path of Policy Gradients

The progression of algorithms discussed illustrates a logical journey toward more effective RL agents.

1. **The Starting Point: REINFORCE.** This algorithm provides the foundational insight of directly optimizing a policy but suffers from extremely high variance.

2. **Improving Credit Assignment.** The first advancements focused on refining the learning signal. This was achieved by introducing **causality** (reward-to-go), **discounting**, and **baselines**, culminating in the realization that the state-value function, $V^\pi(s)$, is the optimal baseline.

3. **Learned Credit Assignment: Actor-Critic.** This gave rise to the Actor-Critic framework, where a **Critic** learns the value function to compute a low-variance advantage estimate, providing the **Actor** with a much cleaner learning signal.

4. **Ensuring Stable Updates: PPO.** While Actor-Critic methods solve the variance problem, they do not inherently prevent destructive updates. **Proximal Policy Optimization (PPO)** addresses this by introducing a clipped surrogate objective that constrains the magnitude of policy changes, ensuring stable and robust training.

5.2 Concluding Remarks and Future Directions

Proximal Policy Optimization has established itself as a default algorithm for a wide range of RL problems [13, 27]. Its combination of strong empirical performance, simplicity, and robustness has made it a benchmark against which new algorithms are measured. However, the field continues to evolve. Active areas of investigation include:

- Developing algorithms with even greater sample efficiency.
- Designing more sophisticated exploration strategies.
- Creating algorithms that are even more robust to hyperparameter choices.
- Proposing direct alternatives to PPO's clipping mechanism that might offer more principled KL-divergence control without sacrificing simplicity [25].

Ultimately, the journey through advanced policy gradients demonstrates a core lesson in AI: progress is often achieved by creating mechanisms that allow an agent to learn effectively from noisy and ambiguous feedback.

References

[1] Mnih, V., et al. (2016). Asynchronous Methods for Deep Reinforcement Learning. *ICML.*

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

[3] Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-Critic Algorithms. *NIPS.*

[4] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning.*

[5] Schulman, J. (2016). Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs. *PhD Thesis, UC Berkeley.*

[6] Greensmith, E., Bartlett, P. L., & Baxter, J. (2004). Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning. *Journal of Machine Learning Research.*

[7] Henderson, P., et al. (2018). Deep Reinforcement Learning that Matters. *AAAI.*

[8] Tucker, G., et al. (2018). The Mirage of Action-Dependent Baselines in Reinforcement Learning. *ICML.*

[9] Islam, R., et al. (2017). Reproducibility of Benchmarked Deep Reinforcement Learning Tasks. *ICML Workshop.*

[10] Mania, H., et al. (2018). Simple random search provides a competitive baseline for reinforcement learning. *arXiv preprint.*

[11] Engstrom, L., et al. (2020). Implementation Matters in Deep RL: A Case Study on PPO and TRPO. *ICLR.*

[12] Schulman, J., et al. (2015). Trust Region Policy Optimization. *ICML.*

[13] Schulman, J., et al. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint.*

[14] OpenAI. (2018). "PPO." *OpenAI Blog.*

[15] Wang, Z., et al. (2017). Sample Efficient Actor-Critic with Experience Replay. *ICLR.*

[16] Kakade, S. (2002). A Natural Policy Gradient. *NIPS.*

[17] Arjovsky, M., et al. (2017). Towards Principled Methods for Training Generative Adversarial Networks. *ICLR.*

[18] Andrychowicz, M., et al. (2021). What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study. *ICLR.*

[19] Schulman, J. (2020). "A Deeper Look at PPO." *BAIR Blog.*

[20] Hill, A., et al. (2018). Stable Baselines. *GitHub Repository.*

[21] Dhariwal, P., et al. (2017). OpenAI Baselines. *GitHub Repository.*

[22] Weng, L. (2018). ”Policy Gradient Algorithms.” *Lil'Log Blog.*

[23] Kostrikov, I. (2018). ”Pytorch-a2c-ppo-acktr-gail.” *GitHub Repository.*

[24] Schulman, J., et al. (2016). High-Dimensional Continuous Control Using Generalized Advantage Estimation. *ICLR.*

[25] Queeney, J., et al. (2023). Simple Policy Optimization. *ICLR.*

[26] Raffin, A., et al. (2021). Stable-Baselines3: A Reliable Reinforcement Learning Implementation. *Journal of Machine Learning Research.*

[27] Berner, C., et al. (2019). Dota 2 with Large Scale Deep Reinforcement Learning. *arXiv preprint.*