A Theoretical Exposition of Modern Policy Gradient Methods: From First Principles to Soft Actor-Critic

by Taha Majlesi

July 17, 2025

Contents

Introduction

This report provides a comprehensive theoretical exploration of policy-based methods in reinforcement learning (RL), charting a course from the foundational principles of direct policy optimization to the sophisticated, entropy-regularized framework of Soft Actor-Critic (SAC). The central objective of reinforcement learning is to develop an agent that can learn an optimal strategy, or **policy**, for interacting with an environment to maximize a cumulative reward signal. Policy-based methods form a distinct and powerful class of algorithms within RL that directly parameterize and optimize this policy, in contrast to value-based methods that first learn a value function and then derive a policy from it.

The thematic arc of this report traces the evolution of these methods as a direct response to a series of fundamental theoretical and practical challenges. We will begin by deriving the **Policy Gradient Theorem**, the cornerstone of this entire subfield, which provides a way to compute the gradient of the expected return with respect to the policy parameters without needing a model of the environment's dynamics. We will then examine its most direct implementation, the **REINFORCE** algorithm, a Monte Carlo approach that, while theoretically sound, suffers from high variance in its gradient estimates, leading to slow and unstable learning.

This critical issue of variance motivates the introduction of state-dependent baselines and, ultimately, the **Advantage Function**, a refined learning signal that measures the relative quality of an action compared to the policy's average behavior in a given state. We will then bridge the conceptual gap between value-based and policy-based RL by re-interpreting policy gradients as an infinitesimal form of the classic **Policy Iteration** algorithm. This perspective is enabled by the crucial **Performance Difference Lemma**, which precisely relates the change in policy performance to the expected advantage function.

This unification, however, reveals a subtle but critical flaw in naive implementations: the problem of **state distribution mismatch**. This issue arises because the performance of a new policy is evaluated using data generated by the old policy, an approximation that is only valid for small policy changes. We will analyze the theoretical bounds on this mismatch and contextualize it within the framework of trust-region methods, such as **Trust Region Policy Optimization (TRPO)** and **Proximal Policy Optimization (PPO)**, which were developed specifically to ensure stable learning by constraining the magnitude of policy updates.

Finally, this report will culminate in a deep dive into **Soft Actor-Critic (SAC)**, a state-of-the-art off-policy algorithm that represents a paradigm shift towards maximum entropy reinforcement learning. We will present its theoretical guarantees, which are rooted in a novel "Soft Policy Iteration" framework, and dissect its practical implementation, including its unique loss functions and architecture that contribute to its remarkable sample efficiency and stability. This journey from first principles to advanced algorithms will illuminate the key innovations that have established policy gradient methods as among the most powerful and widely used tools for solving complex sequential decision-making problems in modern artificial intelligence.

1 The Foundations of Policy Gradient Optimization

This section establishes the theoretical and practical groundwork for all policy gradient methods. The core idea is to directly adjust the parameters of a policy in the direction that increases the expected cumulative reward. We will derive the fundamental theorem that makes this direct optimization possible, introduce its simplest algorithmic form (REINFORCE), and then address its primary practical limitation—high variance—by developing the concept of the advantage function.

1.1 The Policy Gradient Theorem: A First-Principles Derivation

The primary goal of policy-based reinforcement learning is to find the optimal parameters, denoted by a vector θ, for a stochastic policy $\pi_\theta(a|s)$. This policy defines a probability distribution over actions a given a state s. The quality of a policy is measured by an objective function $J(\theta)$, which is typically the expected total discounted reward obtained by following the policy.

For an episodic task, this objective can be defined as the value of the starting state s_0:

$$J(\theta) = V^{\pi_\theta}(s_0) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] \tag{1}$$

where τ represents a full trajectory of states, actions, and rewards, $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T, a_T, r_{T}$ and $R(\tau) = \sum_{t=0}^{T} \gamma^t r_{t+1}$ is the cumulative discounted reward for that trajectory. The expectation $\mathbb{E}_{\tau \sim \pi_\theta}$ is taken over the distribution of trajectories induced by the policy π_θ. This distribution, $p_\theta(\tau)$, can be expressed as:

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{2}$$

Here, $p(s_0)$ is the initial state distribution and $p(s_{t+1}|s_t, a_t)$ represents the environment's transition dynamics, which are typically unknown to the agent.

The objective function can be written as an integral over all possible trajectories:

$$J(\theta) = \int p_\theta(\tau) R(\tau) d\tau \tag{3}$$

To optimize the policy using gradient ascent, we need to compute the gradient of this objective function with respect to the policy parameters, $\nabla_\theta J(\theta)$. The primary challenge is that the policy parameters θ influence the trajectory distribution $p_\theta(\tau)$ in a complex, environment-dependent way. A naive attempt to compute the gradient would involve differentiating the trajectory distribution:

$$\nabla_\theta J(\theta) = \nabla_\theta \int p_\theta(\tau) R(\tau) d\tau = \int [\nabla_\theta p_\theta(\tau)] R(\tau) d\tau \tag{4}$$

This form is intractable because calculating $\nabla_\theta p_\theta(\tau)$ requires differentiating the environment's dynamics, which are unknown.

To overcome this, we employ a fundamental mathematical tool known as the **log-derivative trick** or the score function identity. For any differentiable function $f(x)$, its derivative can be expressed as $\nabla f(x) = f(x) \nabla \log f(x)$. Applying this to our trajectory distribution $p_\theta(\tau)$, we get:

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) \tag{5}$$

This identity is powerful because it allows us to reintroduce the original probability distribution $p_\theta(\tau)$ into the gradient expression. Substituting this back into the gradient of the objective function, we have:

$$\nabla_\theta J(\theta) = \int p_\theta(\tau)[\nabla_\theta \log p_\theta(\tau)]R(\tau)d\tau \tag{6}$$

This integral is now in the form of an expectation, which can be estimated by sampling trajectories from the environment:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[(\nabla_\theta \log p_\theta(\tau))R(\tau)] \tag{7}$$

The next crucial step is to simplify the term $\nabla_\theta \log p_\theta(\tau)$. By taking the logarithm of the trajectory probability, the product becomes a sum:

$$\log p_\theta(\tau) = \log p(s_0) + \sum_{t=0}^{T}(\log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t)) \tag{8}$$

When we take the gradient of this expression with respect to θ, the terms that do not depend on the policy parameters—namely, the initial state distribution $p(s_0)$ and the environment dynamics $p(s_{t+1}|s_t, a_t)$—vanish, as their gradients are zero. This leaves us with a much simpler expression:

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{9}$$

This is a remarkable result. The gradient of the log-probability of a trajectory depends only on the gradient of the log-probabilities of the actions taken under the policy, completely independent of the environment's dynamics.

Substituting this back into our expectation form gives the final expression for the **Policy Gradient Theorem**:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right) R(\tau)\right] \tag{10}$$

This theorem is the cornerstone of all policy gradient methods. It provides a way to estimate the gradient of the performance objective by simply running the policy in the environment, collecting trajectories, and averaging the results. It transforms an intractable optimization problem into a tractable stochastic estimation problem.

1.2 The REINFORCE Algorithm: A Monte Carlo Realization

The REINFORCE algorithm, also known as Monte Carlo Policy Gradient, is the most direct algorithmic implementation of the Policy Gradient Theorem. It operates by collecting full episodes (trajectories) of experience and using these to compute a stochastic estimate of the gradient, which is then used to update the policy parameters via gradient ascent.

The basic REINFORCE algorithm proceeds as follows:

1. **Sample Trajectories:** Using the current policy π_θ, generate a batch of N trajectories, $\{\tau_1, \tau_2, \ldots, \tau_N\}$. Each trajectory consists of a sequence of states, actions, and rewards from an initial state to a terminal state.

2. **Estimate the Gradient:** For each trajectory τ_i, compute the total return $R(\tau_i)$. Then, estimate the policy gradient using the sample average of the expression from the Policy Gradient Theorem:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) R(\tau_i) \right] \tag{11}$$

3. **Update Policy Parameters:** Update the policy parameters θ in the direction of the estimated gradient:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{12}$$

   where α is a learning rate hyperparameter.

While this formulation is a direct application of the theorem, it contains a significant inefficiency related to causality. The term $R(\tau_i)$ is the total return for the entire episode. This means that the update for an action a_t at time step t is scaled by rewards received before time t. However, an action taken at time t can only influence future rewards, not past ones. This observation leads to a more refined and lower-variance gradient estimator.

We can rewrite the gradient estimator by swapping the order of summations and recognizing that the rewards from before time t can be dropped without introducing bias. This leads to the use of the **reward-to-go**, which is the sum of rewards from a specific time step t until the end of the episode. The reward-to-go is defined as:

$$\hat{Q}^\pi(s_t, a_t) = \sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'}) \tag{13}$$

This term is an empirical Monte Carlo estimate of the action-value function $Q^\pi(s_t, a_t)$, representing the expected return after taking action a_t in state s_t and following policy π thereafter.

Using the reward-to-go, the policy gradient estimator becomes:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \hat{Q}^\pi_{i,t} \tag{14}$$

This is the most common form of the REINFORCE algorithm. The intuition is straightforward: for each state-action pair in a trajectory, we compute the gradient of the log-probability of that action, $\nabla_\theta \log \pi_\theta(a_t|s_t)$. This gradient vector points in the direction in parameter space that increases the probability of selecting action a_t in state s_t. We then scale this vector by the reward-to-go, $\hat{Q}^\pi_{i,t}$. If the subsequent rewards were high (a large positive $\hat{Q}$), we push the policy parameters significantly in that direction. If the subsequent rewards were low (or negative), we push the parameters in the opposite direction, making that action less likely in the future.

While this refined estimator respects causality, it still suffers from a major practical drawback: **high variance**. The reward-to-go $\hat{Q}^\pi_{i,t}$ is a Monte Carlo estimate based on a single trajectory. Due to the stochasticity of both the policy and the environment, this value can fluctuate wildly for the same state-action pair across different episodes. This high variance in the gradient estimate leads to noisy updates, which can slow down the learning process and make it unstable.

1.3 Mitigating Variance: Baselines and the Advantage Function

The high variance of the REINFORCE gradient estimator is a significant practical hurdle. Consider an environment where all rewards are positive, for instance, ranging from +100 to +110. An action leading to a reward-to-go of +105 will be reinforced, while an action leading to +102 will also be reinforced, just slightly less. However, if an action leads to a reward-to-go of +90, the policy will be updated to make that action less likely, even though it may still be a very good outcome in absolute terms. The learning signal is sensitive to the arbitrary zero-point of the reward function. What matters is not the absolute value of the return, but whether it was **better or worse than what was expected**.

To address this, we can introduce a state-dependent **baseline**, $b(s_t)$, into the policy gradient formula. The gradient estimator becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(\hat{Q}_t^\pi - b(s_t))\right] \tag{15}$$

A remarkable property of this formulation is that as long as the baseline $b(s_t)$ does not depend on the action a_t, it can be subtracted from the reward-to-go without introducing any bias into the gradient estimate. This can be proven formally by showing that the expectation of the added term is zero:

$$\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)\right] = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)]$$

Focusing on a single timestep t, the expectation over actions is:

$$\mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)}[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)] = b(s_t)\int \pi_\theta(a_t|s_t)\nabla_\theta \log \pi_\theta(a_t|s_t) da_t$$

Using the log-derivative trick in reverse ($\nabla f(x) = f(x)\nabla \log f(x)$):

$$\begin{aligned} &= b(s_t)\int \nabla_\theta \pi_\theta(a_t|s_t) da_t \\ &= b(s_t)\nabla_\theta \int \pi_\theta(a_t|s_t) da_t \end{aligned}$$

Since the integral of a probability distribution over its domain is 1, this becomes:

$$= b(s_t)\nabla_\theta(1) = 0$$

Because the expected value of the baseline term is zero, subtracting it does not change the expected gradient, meaning the estimator remains unbiased.

The question then becomes: what is the optimal choice for the baseline $b(s_t)$? The ideal baseline for variance reduction is one that is highly correlated with the reward-to-go $\hat{Q}_t^\pi$. The natural choice is the **state-value function**, $V^\pi(s_t)$, which is defined as the expected reward-to-go from state s_t:

$$V^\pi(s_t) = \mathbb{E}_{\tau\sim\pi_\theta}[\hat{Q}_t^\pi|s_t] \tag{16}$$

By choosing $b(s_t) = V^\pi(s_t)$, we are centering the learning signal. The term $(\hat{Q}_t^\pi - V^\pi(s_t))$ will be positive for actions that led to a better-than-average outcome for that state, and negative for actions that led to a worse-than-average outcome. This subtraction isolates the incremental benefit of choosing a specific action over the "default" or average behavior of the policy in that state.

This leads to the definition of the **Advantage Function**, $A^\pi(s_t, a_t)$:

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \tag{17}$$

The advantage function quantifies precisely how much better or worse a specific action a_t is compared to the average action in a given state s_t. The policy gradient can now be expressed in its most common and effective form:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^\pi(s_t, a_t)\right] \tag{18}$$

This formulation is at the heart of most modern policy gradient algorithms, including actor-critic methods. In practice, the advantage function $A^\pi(s_t, a_t)$ is not known exactly and must be estimated. Actor-critic methods accomplish this by training a separate neural network, the "critic," to approximate the value function $V^\pi(s_t)$, which is then used to compute an estimate of the advantage. This progression—from total reward to reward-to-go to the advantage function—represents a principled refinement of the learning signal to create more stable and efficient policy gradient algorithms.

2 Unifying Perspectives: Policy Gradient as Policy Iteration

While policy gradient methods and value-based methods like Q-learning are often presented as distinct branches of reinforcement learning, a deeper theoretical analysis reveals a profound connection. This section frames policy gradients within the classic Generalized Policy Iteration (GPI) framework, demonstrating that direct policy optimization can be understood as an infinitesimal version of the evaluation-improvement loop that underpins value-based methods. This perspective not only provides a stronger theoretical foundation for policy gradients but also clarifies the fundamental role of the advantage function.

2.1 The Generalized Policy Iteration (GPI) Framework

Generalized Policy Iteration is a high-level concept that describes the general structure of most reinforcement learning algorithms. It involves the interaction of two competing and complementary processes:

- **Policy Evaluation:** For a given policy π, this process aims to compute its corresponding value function, either the state-value function $V^\pi(s)$ or the action-value function $Q^\pi(s, a)$. This step answers the question: "How good is the current policy?".
- **Policy Improvement:** For a given value function V (or Q), this process aims to find a new, better policy π' by making it "greedy" with respect to the value

function. This step answers the question: "How can I improve my policy based on my current evaluation of the world?".

In classical dynamic programming, these two steps are performed iteratively and to completion. One first evaluates the policy completely, then improves it completely, and repeats. The interaction between these two processes is guaranteed to converge to the optimal policy π^* and the optimal value function V^*. The "generalized" aspect of GPI refers to the fact that these processes can be interleaved in many ways. For example, value iteration performs only one step of evaluation (a single Bellman backup) before improving the policy, and asynchronous methods can update values and policies in an even more fine-grained manner.

2.2 The Policy Improvement Theorem

The guarantee that the GPI scheme converges rests on the Policy Improvement Theorem. This theorem provides the formal justification for the "improvement" step, ensuring that making a policy greedy with respect to its own value function leads to a new policy that is at least as good as, and usually better than, the original.

For any pair of deterministic policies π and π', the theorem states that if for all states $s \in S$, the following condition holds:

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s) \tag{19}$$

then the policy π' must be as good as, or better than, π. That is:

$$V^{\pi'}(s) \geq V^\pi(s) \quad \text{for all } s \in S \tag{20}$$

Furthermore, if the inequality is strict for any state s, then the new policy π' is strictly better for at least one state.

The proof of this theorem is instructive. It proceeds by unrolling the definition of $V^{\pi'}(s)$ and repeatedly applying the initial condition.

$$\begin{aligned} V^{\pi'}(s) &= \mathbb{E}_{\pi'}[r_{t+1} + \gamma V^{\pi'}(s_{t+1}) | s_t = s] \\ &= \mathbb{E}_{\pi'}[r_{t+1} + \gamma \mathbb{E}_{\pi'}[r_{t+2} + \gamma V^{\pi'}(s_{t+2}) | s_{t+1}] | s_t = s] \end{aligned}$$

By repeatedly expanding $V^{\pi'}$ and using the fact that at each step, $Q^\pi(s, \pi'(s)) \geq V^\pi(s)$, we can show that the value of the new policy is bounded below by the value of the old policy. The core logic is that if taking the improved action once and then following the old policy is better, then taking the improved action every time it's available must be even better. This theorem is the bedrock of policy iteration, as it guarantees monotonic improvement until optimality is reached.

2.3 Derivation of the Performance Difference Lemma

To connect policy gradients to this GPI framework, we need a way to express the change in performance, $J(\theta') - J(\theta)$, when we update a policy from π_θ to $\pi_{\theta'}$. The Performance Difference Lemma provides exactly this expression, and its derivation is a cornerstone of modern policy gradient theory.

The central claim is as follows:

$$J(\theta') - J(\theta) = \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_\theta}(s_t, a_t) \right] \tag{21}$$

This equation states that the improvement in performance from switching to a new policy $\pi_{\theta'}$ is equal to the expected sum of advantages of the new policy's actions, where the advantages are calculated with respect to the old policy π_θ.

The proof, as detailed in the lecture slides, is an elegant application of algebraic manipulation and the definitions of value functions:

1. Begin with the performance difference:

   $$J(\theta') - J(\theta) = J(\theta') - \mathbb{E}_{s_0 \sim p(s_0)}[V^{\pi_\theta}(s_0)]$$

   The value of the initial state, $V^{\pi_\theta}(s_0)$, is an expectation over the initial state distribution. Since this distribution is independent of the policy, we can change the outer expectation to be over trajectories sampled from the new policy, $p_{\theta'}(\tau)$, without changing the value:

   $$J(\theta') - J(\theta) = J(\theta') - \mathbb{E}_{\tau \sim p_{\theta'}(\tau)}[V^{\pi_\theta}(s_0)]$$

2. Express $V^{\pi_\theta}(s_0)$ using a telescoping sum. For any sequence, $x_0 = \sum_{t=0}^{\infty}(x_t - x_{t+1})$. Applying this to the value function sequence, where the discount factor γ is included:

   $$V^{\pi_\theta}(s_0) = \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t V^{\pi_\theta}(s_t) - \sum_{t=1}^{\infty} \gamma^t V^{\pi_\theta}(s_t) \right]$$

   Note that $\sum_{t=1}^{\infty} \gamma^t V^{\pi_\theta}(s_t) = \sum_{t=0}^{\infty} \gamma^{t+1} V^{\pi_\theta}(s_{t+1})$.

3. Substituting this back and rearranging terms:

   $$J(\theta') - J(\theta) = J(\theta') + \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t (\gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)) \right]$$

4. Substitute the definition of the new policy's performance, $J(\theta') = \mathbb{E}_{\tau \sim p_{\theta'}(\tau)}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$:

   $$= \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] + \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t (\gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)) \right]$$

5. Combine the expectations:

   $$= \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t) + \gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)) \right]$$

   The term inside the parentheses is the definition of the temporal-difference (TD) error, which is also an unbiased, single-sample estimate of the advantage function $A^{\pi_\theta}(s_t, a_t)$. Thus, we arrive at the final lemma:

   $$J(\theta') - J(\theta) = \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_\theta}(s_t, a_t) \right] \tag{22}$$

This completes the derivation.

2.4 Interpreting Policy Gradient as Approximate Policy Iteration

The Performance Difference Lemma provides the crucial link to interpret policy gradient methods as a form of GPI. The overall process of a policy gradient algorithm can be broken down into two main steps that directly mirror the evaluation and improvement stages of policy iteration.

- **Policy Evaluation (Approximate):** The first step in a policy gradient algorithm is to estimate the advantage function, $A^{\pi_\theta}(s,a)$, for the current policy π_θ.
  - In REINFORCE, this is done via Monte Carlo rollouts, where $\hat{A}_t^\pi = \hat{Q}_t^\pi - b(s_t)$.
  - In actor-critic methods, this is done by training a critic network to approximate $V^{\pi_\theta}(s)$ or $Q^{\pi_\theta}(s,a)$, from which the advantage is computed.

  This step is analogous to the policy evaluation phase of GPI. It is an *approximate* evaluation because we are using sampled data and function approximators rather than solving the Bellman equations exactly.

- **Policy Improvement (Infinitesimal):** The second step is to update the policy parameters θ to improve performance. The policy gradient update rule is $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$. From Section 1, we know that $\nabla_\theta J(\theta)$ is proportional to $\mathbb{E}[\nabla_\theta \log \pi_\theta(a|s) A^\pi(s,a)]$. This update explicitly pushes the policy to increase the probability of actions with positive advantages and decrease the probability of actions with negative advantages.

  This step is analogous to the policy improvement phase of GPI. It seeks to find a new policy $\pi_{\theta'}$ that maximizes the expected advantage, which the Performance Difference Lemma tells us will maximize the performance improvement $J(\theta') - J(\theta)$. The update is *infinitesimal* because it takes a small step in the direction of the gradient rather than making a large, discrete jump to a fully greedy policy.

This framing is not merely an analogy; it is a formal equivalence in the limit of small updates. It provides a powerful theoretical lens through which to understand the behavior of policy gradient methods. It explains why the advantage function is the correct quantity to use as the learning signal—it is precisely the quantity that drives the improvement step in classical policy iteration. This unification lends the strong theoretical foundations of GPI to policy gradient methods, connecting two seemingly disparate branches of reinforcement learning.

3 The Challenge of Distribution Mismatch and Trust Regions

The interpretation of policy gradients as a form of policy iteration reveals a subtle but critical issue that lies at the heart of on-policy learning. The Performance Difference Lemma, $J(\theta') - J(\theta) = \mathbb{E}_{\tau \sim p_{\theta'}(\tau)}[\dots]$, states that the improvement is an expectation over trajectories generated by the *new* policy, $\pi_{\theta'}$. However, in practice, we only have access to data collected with the *old* policy, π_θ. This discrepancy is known as the **state distribution mismatch** problem. This section dissects this problem, introduces importance sampling as the theoretical solution, analyzes the bounds on this mismatch, and connects this theoretical challenge to the practical solutions offered by Trust Region methods like TRPO and PPO.

3.1 Identifying the Mismatch: A Theoretical Flaw in Practice

The core of the state distribution mismatch problem is a practical contradiction. The theoretical formula for policy improvement requires us to evaluate the expected advantage under the new policy's state visitation distribution, $p_{\theta'}(s_t)$. But to generate samples from this distribution, we would need to run the new policy $\pi_{\theta'}$. We cannot determine the new policy until we have computed the update, and we cannot compute the update without samples from the new policy. This creates a chicken-and-egg problem.

To break this cycle, on-policy algorithms like REINFORCE make a crucial, and technically incorrect, approximation. They substitute the expectation over the new policy's distribution with an expectation over the old policy's distribution:

$$\mathbb{E}_{\tau\sim p_{\theta'}(\tau)}\left[\sum_t \gamma^t A^{\pi_\theta}(s_t,a_t)\right] \approx \mathbb{E}_{\tau\sim p_{\theta}(\tau)}\left[\sum_t \gamma^t A^{\pi_\theta}(s_t,a_t)\right] \tag{23}$$

This approximation is only valid if the new state distribution $p_{\theta'}$ is very close to the old state distribution p_θ. This, in turn, implies that the new policy $\pi_{\theta'}$ must be very close to the old policy π_θ. Therefore, this approximation holds only when the policy updates are kept small. This introduces a fundamental tension: we desire large, efficient learning steps to speed up convergence, but large steps can cause the new policy to diverge significantly from the old one, invalidating the data used for the update and potentially leading to a catastrophic drop in performance, a phenomenon known as **policy collapse**.

3.2 Correction via Importance Sampling

The statistically sound method for correcting this distribution mismatch is **Importance Sampling (IS)**. The principle of IS allows one to estimate the expectation of a function $f(x)$ under a target distribution $p(x)$ using samples drawn from a different behavior distribution $q(x)$. The fundamental formula is:

$$\mathbb{E}_{x\sim p}[f(x)] = \int p(x)f(x)dx = \int q(x)\frac{p(x)}{q(x)}f(x)dx = \mathbb{E}_{x\sim q}\left[\frac{p(x)}{q(x)}f(x)\right] \tag{24}$$

The term $\frac{p(x)}{q(x)}$ is called the importance weight or importance ratio.

Applying this to the performance difference lemma, we can correct for the mismatch in the action selection probabilities at each state. The expectation over actions from the new policy, $\mathbb{E}_{a_t\sim\pi_{\theta'}}$, can be rewritten as an expectation over actions from the old policy, $\mathbb{E}_{a_t\sim\pi_\theta}$, by introducing an importance ratio:

$$\mathbb{E}_{a_t\sim\pi_{\theta'}(a_t|s_t)}[A^{\pi_\theta}(s_t,a_t)] = \mathbb{E}_{a_t\sim\pi_{\theta}(a_t|s_t)}\left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)}A^{\pi_\theta}(s_t,a_t)\right] \tag{25}$$

This correction addresses the difference in action probabilities but does not yet fix the mismatch in the state visitation distribution, $p(s_t)$. A full off-policy correction would require multiplying importance ratios across the entire trajectory, which is known to suffer from extremely high variance, especially in long-horizon problems, making it practically unusable in many cases. This high variance is why on-policy methods, which aim to keep the policy change small to minimize the mismatch, are often preferred for their stability.

3.3 Bounding the Mismatch Error: A Theoretical Justification

To provide a theoretical justification for the on-policy approximation ($\mathbb{E}_{p_{\theta'}} \approx \mathbb{E}_{p_\theta}$), we must analyze the error introduced by the state distribution mismatch. The lecture slides provide a derivation that bounds this error, showing that it is manageable if the policy update is small.

The derivation proceeds in two main stages:

1. **Bounding the State Distribution Change:** First, we bound the difference between the state distributions, $|p_{\theta'}(s_t) - p_\theta(s_t)|$. We start by defining policy "closeness" as the maximum difference in action probabilities: $|\pi_{\theta'}(a|s) - \pi_\theta(a|s)| \leq \epsilon$ for all states and actions. A key lemma states that if two distributions are ϵ-close, there exists a coupling such that they agree with probability $1 - \epsilon$. This implies that at any step, the new policy will take a "mistake" action (an action different from what the old policy would have likely chosen) with a probability of at most ϵ.

   The state distribution at time t under the new policy, $p_{\theta'}(s_t)$, can be expressed as a mixture of the distribution where no mistakes have occurred and a distribution resulting from at least one mistake:

   $$p_{\theta'}(s_t) = (1-\epsilon)^t p_\theta(s_t) + (1 - (1-\epsilon)^t) p_{\text{mistake}}(s_t)$$

   The absolute difference is then:

   $$|p_{\theta'}(s_t) - p_\theta(s_t)| = (1 - (1-\epsilon)^t)|p_\theta(s_t) - p_{\text{mistake}}(s_t)|$$

   Since the total variation distance between any two probability distributions is at most 2, we have $|p_\theta(s_t) - p_{\text{mistake}}(s_t)| \leq 2$. Using the inequality $(1-\epsilon)^t \geq 1 - \epsilon t$ for $\epsilon \in [0, 1]$, the bound simplifies to:

   $$|p_{\theta'}(s_t) - p_\theta(s_t)| \leq 2(1 - (1-\epsilon)^t) \leq 2\epsilon t \tag{26}$$

   This bound, while not tight, formally shows that the divergence between state distributions grows linearly with time and is proportional to the "closeness" of the policies, ϵ.

2. **Bounding the Objective Value Error:** This bound on the state distribution is then used to bound the error in the objective function. Let $L(\theta')$ be the approximate objective calculated using the old policy's state distribution. The error between the true improvement and the approximate one is bounded by:

   $$|(J(\theta') - J(\theta)) - L(\theta')| \leq \sum_{t=0}^{T} \gamma^t |p_{\theta'}(s_t) - p_\theta(s_t)| \max_{s_t, a_t} |A^{\pi_\theta}(s_t, a_t)|$$

   Substituting our bound for the distribution mismatch and letting C be a constant upper bound on the advantage function (e.g., $C = O(r_{\max}/(1-\gamma))$), we get:

   $$|(J(\theta') - J(\theta)) - L(\theta')| \leq \sum_{t=0}^{T} \gamma^t (2\epsilon t) C$$

   This leads to a lower bound on the true performance improvement:

   $$J(\theta') - J(\theta) \geq L(\theta') - \sum_{t=0}^{T} 2\epsilon t \gamma^t C \tag{27}$$

The implication is profound: maximizing the approximate objective $L(\theta')$ is equivalent to maximizing a lower bound on the true performance improvement. This provides a theoretical justification for using the on-policy approximation, but it hinges critically on the policy update being small (i.e., ϵ being small) to keep the error term from dominating.

3.4 Contextualizing with Trust Region Methods (TRPO & PPO)

The analysis of the distribution mismatch reveals the core challenge of on-policy learning: how to make learning steps as large as possible for efficiency, without making them so large that they destabilize the learning process. **Trust Region Methods** are a class of algorithms designed to explicitly manage this trade-off.

3.4.1 Trust Region Policy Optimization (TRPO)

TRPO formalizes the idea of keeping the new policy "close" to the old one by solving a constrained optimization problem at each step. It aims to maximize the approximate performance improvement, $L(\theta')$, subject to a hard constraint on the average KL-divergence between the old and new policies:

$$\text{maximize}_{\theta'} \quad \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta}\left[\frac{\pi_{\theta'}(a|s)}{\pi_\theta(a|s)} A^{\pi_\theta}(s,a)\right] \tag{28}$$

$$\text{subject to} \quad \mathbb{E}_{s\sim\rho_\theta}[\mathrm{D}_{\mathrm{KL}}(\pi_\theta(\cdot|s)||\pi_{\theta'}(\cdot|s))] \leq \delta \tag{29}$$

Here, δ defines the size of the "trust region." By explicitly constraining the policy change, TRPO ensures that the on-policy approximation remains valid and guarantees monotonic policy improvement. While powerful, TRPO is a second-order optimization method that involves complex calculations (like the Fisher-vector product via conjugate gradients), making it computationally expensive and difficult to implement.

3.4.2 Proximal Policy Optimization (PPO)

PPO was developed as a simpler, first-order approximation of TRPO that achieves similar stability and performance and has become a default algorithm in deep RL. Instead of a hard KL-divergence constraint, PPO uses a novel **clipped surrogate objective function** to discourage large policy updates:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right] \tag{30}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the importance ratio. The 'clip' function limits the ratio $r_t(\theta)$ to the range $[1-\epsilon, 1+\epsilon]$. The 'min' operator then takes the smaller of the normal policy gradient objective and this clipped objective. The effect is that if a policy update would push the ratio $r_t(\theta)$ far from 1, the clipping mechanism activates and removes the incentive for such a large change. This effectively creates a soft "trust region" without the computational overhead of TRPO, making it much simpler to implement and often more sample-efficient. Some PPO implementations also incorporate an early stopping mechanism based on a KL-divergence threshold as an additional safeguard against overly large updates.

In summary, the state distribution mismatch is the central theoretical problem that motivates the development of modern, stable on-policy algorithms. The theoretical bounds justify the need for small updates, and TRPO and PPO provide principled, practical methods for managing the size of these updates to ensure stable and monotonic improvement.

4 Soft Actor-Critic: Reinforcement Learning with Maximum Entropy

This final section details the Soft Actor-Critic (SAC) algorithm, a state-of-the-art off-policy method that introduces a new paradigm: maximum entropy reinforcement learning. This framework fundamentally alters the optimization objective by rewarding the policy not just for maximizing cumulative reward, but also for maintaining high entropy, or randomness. We will explore its theoretical foundation in "Soft Policy Iteration" as presented in the lecture slides, and then dissect its practical architecture and the loss functions that drive its remarkable stability and sample efficiency.

4.1 The Maximum Entropy RL Objective

In standard reinforcement learning, the objective is to find a policy π that maximizes the expected cumulative reward:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t)] \tag{31}$$

where ρ_π is the state-action visitation distribution induced by policy π.

Soft Actor-Critic modifies this objective by adding a policy entropy term at each timestep, weighted by a "temperature" parameter, α. The new objective is to maximize the entropy-augmented return:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{32}$$

where $\mathcal{H}(\pi(\cdot|s_t))$ is the entropy of the policy distribution at state s_t, defined as:

$$\mathcal{H}(\pi(\cdot|s_t)) = \mathbb{E}_{a_t\sim\pi(\cdot|s_t)}[-\log\pi(a_t|s_t)] \tag{33}$$

The intuition behind this objective is that the agent is incentivized to succeed at the task while acting as randomly as possible. This has several profound benefits:

- **Improved Exploration:** The entropy bonus directly encourages the agent to explore. By rewarding randomness, it prevents the policy from prematurely converging to a sharp, deterministic, and potentially suboptimal peak in the policy landscape. This allows the agent to discover more diverse and potentially more rewarding behaviors.
- **Improved Robustness and Stability:** Entropy maximization makes the learning process more robust to model inaccuracies and estimation errors. The resulting policies are less "brittle" because they do not commit fully to a single course of action. If multiple actions have similar high values, the maximum entropy policy will assign them roughly equal probability, allowing it to adapt if one action becomes unavailable or less effective. This also helps to smooth the learning landscape, contributing to more stable convergence.

- **Learning Multi-modal Behaviors:** In tasks where multiple distinct strategies are equally optimal, a standard RL agent might arbitrarily converge to one. A maximum entropy agent, however, can learn to represent all of them, committing probability mass to each optimal mode of behavior.

4.2 The Theoretical Framework: Soft Policy Iteration

The SAC algorithm is not merely an ad-hoc addition of an entropy term; it is derived from a rigorous, modified version of the policy iteration framework, termed **Soft Policy Iteration**. This framework redefines the core operations of policy evaluation and improvement to be consistent with the maximum entropy objective.

4.2.1 Soft Policy Evaluation (Lemma 1)

The first step is to define how to evaluate a policy in this new "soft" setting. This involves defining a soft value function and a corresponding soft Bellman backup operator.

The **soft state-value function** $V^\pi(s)$ and **soft action-value function** $Q^\pi(s,a)$ are defined as the expected return plus the expected future discounted entropy:

$$V^\pi(s_t) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{k=t}^{\infty}\gamma^{k-t}(r(s_k,a_k)+\alpha\mathcal{H}(\pi(\cdot|s_k)))|s_t\right] \tag{34}$$

These two functions are related by the following equations:

$$V^\pi(s_t) = \mathbb{E}_{a_t\sim\pi}[Q^\pi(s_t,a_t)-\alpha\log\pi(a_t|s_t)] \tag{35}$$

$$Q^\pi(s_t,a_t) = r(s_t,a_t)+\gamma\mathbb{E}_{s_{t+1}\sim p}[V^\pi(s_{t+1})] \tag{36}$$

Combining these gives the **soft Bellman expectation equation**:

$$Q^\pi(s_t,a_t) = r(s_t,a_t)+\gamma\mathbb{E}_{s_{t+1}\sim p,a_{t+1}\sim\pi}[Q^\pi(s_{t+1},a_{t+1})-\alpha\log\pi(a_{t+1}|s_{t+1})] \tag{37}$$

The **soft Bellman backup operator**, $\mathcal{T}^\pi$, is defined based on this relationship:

$$\mathcal{T}^\pi Q(s_t,a_t) \triangleq r(s_t,a_t)+\gamma\mathbb{E}_{s_{t+1}\sim p}\left[\mathbb{E}_{a_{t+1}\sim\pi}[Q(s_{t+1},a_{t+1})-\alpha\log\pi(a_{t+1}|s_{t+1})]\right] \tag{38}$$

Lemma 4.1 (Soft Policy Evaluation). *For any initial Q-function Q_0, repeated application of the soft Bellman backup operator, $Q_{k+1}=\mathcal{T}^\pi Q_k$, is guaranteed to converge to the true soft Q-value of the policy π, i.e., $\lim_{k\to\infty}Q_k=Q^\pi$.*

The proof of this lemma relies on showing that the operator $\mathcal{T}^\pi$ is a contraction mapping with respect to the maximum norm, which guarantees a unique fixed point.

4.2.2 Soft Policy Improvement (Lemma 2)

The second step is to improve the policy based on the soft Q-function. In contrast to standard policy iteration's greedy 'argmax' update, soft policy improvement updates the policy towards a distribution derived from the exponential of the soft Q-function. This new policy, π_{new}, is found by solving the following optimization problem:

$$\pi_{\text{new}} = \arg\min_{\pi'\in\Pi}\mathrm{D}_{\mathrm{KL}}\left(\pi'(\cdot|s_t)\middle\|\frac{\exp(\frac{1}{\alpha}Q^{\pi_{\text{old}}}(s_t,\cdot))}{Z^{\pi_{\text{old}}}(s_t)}\right) \tag{39}$$

Here, Π is a family of tractable policies (e.g., Gaussians), $\mathrm{D_{KL}}$ is the Kullback-Leibler divergence, and Z is the partition function that normalizes the distribution. This update projects the new policy to be as close as possible to the Boltzmann distribution defined by the old policy's Q-function.

Lemma 4.2 (Soft Policy Improvement)**.** *This policy update guarantees a monotonic improvement in the soft Q-value: for the new policy π_{new} derived from π_{old}, we have $Q^{\pi_{new}}(s,a) \geq Q^{\pi_{old}}(s,a)$ for all (s,a).*

The proof is elegant and proceeds by showing that the expected soft value under the new policy is at least as good as the old soft value function.

4.2.3 Convergence of Soft Policy Iteration (Theorem 1)

Theorem 4.3 (Soft Policy Iteration)**.** *Repeated application of soft policy evaluation and soft policy improvement will cause the sequence of policies to converge to an optimal policy π^* such that $Q^{\pi^*}(s,a) \geq Q^{\pi}(s,a)$ for all other policies $\pi \in \Pi$ and all (s,a).*

The proof relies on the monotonic improvement established in Lemma 2. The sequence of soft Q-functions, Q^{π_i}, is monotonically increasing. Since the rewards and entropy are bounded, the soft Q-function is also bounded above. A monotonically increasing and bounded sequence is guaranteed to converge.

4.3 The Practical SAC Algorithm: Architecture and Loss Functions

The theoretical framework of Soft Policy Iteration provides the foundation for a practical algorithm that uses function approximators (deep neural networks). The modern SAC architecture typically consists of:

- A stochastic **Actor** network, $\pi_\phi(a|s)$, parameterized by ϕ.
- Two **Q-Critic** networks, $Q_{\theta_1}(s,a)$ and $Q_{\theta_2}(s,a)$, parameterized by θ_1 and θ_2.
- Two **Target Q-Critic** networks, $Q_{\theta_{1,\text{targ}}}$ and $Q_{\theta_{2,\text{targ}}}$, which are time-delayed copies of the critic networks.
- An **experience replay buffer** $\mathcal{D}$ to store past transitions for off-policy learning.

The algorithm alternates between collecting experience and updating the network parameters by minimizing three distinct loss functions.

4.3.1 Q-Function Loss (J_Q)

The two Q-critics are trained to minimize the soft Bellman error. The loss function for each critic Q_{θ_i} is the mean squared error between its prediction and a target value y:

$$J_Q(\theta_i) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\frac{1}{2}(Q_{\theta_i}(s_t,a_t) - y)^2\right] \tag{40}$$

The target value y is where the key components of SAC are integrated:

$$y = r_t + \gamma\left(\min_{j=1,2} Q_{\theta_{j,\text{targ}}}(s_{t+1},a_{t+1}) - \alpha\log\pi_\phi(a_{t+1}|s_{t+1})\right) \tag{41}$$

where the next action a_{t+1} is sampled from the current policy, $a_{t+1} \sim \pi_\phi(\cdot|s_{t+1})$.

- **Target Networks:** The target value y is computed using the slowly-updated target Q-networks. This provides a stable learning target.
- **Clipped Double-Q Learning:** The 'min' operator is taken over the two target Q-networks. This is a trick borrowed from TD3 to combat the tendency of Q-learning algorithms to overestimate values.
- **Entropy Regularization:** The target includes the entropy bonus for the next state, consistent with the soft Bellman equation.

4.3.2 Policy Loss (J_π)

The actor (policy) is updated to maximize the expected future return plus entropy. This is achieved by training the policy to output actions that maximize the soft Q-function. The policy loss is derived from the KL-divergence objective in the soft policy improvement step:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}, \epsilon_t \sim \mathcal{N}} \left[\alpha \log \pi_\phi(f_\phi(\epsilon_t; s_t)|s_t) - \min_{j=1,2} Q_{\theta_j}(s_t, f_\phi(\epsilon_t; s_t)) \right] \tag{42}$$

Here, the action is computed using the **reparameterization trick**, $a_t = f_\phi(\epsilon_t; s_t)$, where ϵ_t is a noise vector sampled from a standard distribution (e.g., a Gaussian). This allows the gradient to be backpropagated through the stochastic policy network.

4.3.3 Entropy Temperature Loss (J_α) (Optional but common)

Instead of treating the temperature α as a fixed hyperparameter, SAC can learn it automatically by defining a separate loss function to enforce an entropy constraint:

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi_t}[-\alpha(\log \pi_t(a_t|s_t) + \bar{\mathcal{H}})] \tag{43}$$

where $\bar{\mathcal{H}}$ is a target entropy hyperparameter (often set to a heuristic like $-\dim(\mathcal{A})$). This objective adjusts α via gradient descent such that the policy's average entropy remains close to the target entropy.

Conclusion

This report has provided a detailed theoretical journey through the landscape of modern policy gradient methods, tracing their evolution from foundational principles to state-of-the-art algorithms. The analysis reveals a clear intellectual lineage where each major development serves as a principled solution to the limitations of its predecessors.

The journey began with the **Policy Gradient Theorem**, a fundamental result that makes direct policy optimization tractable. Its most direct implementation, **REINFORCE**, while theoretically sound, was shown to be practically limited by high variance.

This challenge of variance was systematically addressed by incorporating value functions, culminating in the **Advantage Function**. This crucial step also unveiled a deep connection to classical reinforcement learning theory, allowing us to reinterpret policy gradients as an infinitesimal form of **Generalized Policy Iteration**, grounded in the **Performance Difference Lemma**.

However, this perspective also brought to light the critical problem of **state distribution mismatch**. We analyzed the theoretical bounds on the error introduced by this mismatch, which underscored the necessity of constraining policy updates. This theoretical need for stability directly motivates the development of algorithms like **Trust Region Policy Optimization (TRPO)** and its more practical successor, **Proximal Policy Optimization (PPO)**.

Finally, we explored **Soft Actor-Critic (SAC)**, which represents a paradigm shift towards maximum entropy reinforcement learning. By augmenting the standard RL objective with a policy entropy term, SAC achieves superior exploration and robustness. Its convergence is guaranteed by the **Soft Policy Iteration** framework. The practical SAC algorithm, with its sophisticated architecture incorporating twin critics, target networks, and automatic temperature adjustment, stands as a powerful synthesis of these theoretical advancements.

The evolution from REINFORCE to SAC is a compelling narrative of how the field of reinforcement learning has systematically identified and overcome fundamental theoretical and practical challenges—variance, stability, and sample efficiency—to produce algorithms capable of solving increasingly complex sequential decision-making problems.

Comparative Analysis of Modern Policy Gradient Algorithms

Table 1: Comparative Analysis of Modern Policy Gradient Algorithms

Feature	**REINFORCE**	**TRPO**	**PPO**	**SA**
Core Idea	Monte Carlo Policy Gradient	Trust Region Constrained PG	Clipped Surrogate Objective PG	Max Off-
On/Off-Policy	On-Policy	On-Policy	On-Policy	Off-
Policy Type	Stochastic	Stochastic	Stochastic	Stoc
Key Innovation	Direct application of the PG Theorem	KL-divergence trust region constraint for monotonic improvement	Clipped surrogate objective as a 1st-order approx. of TRPO	Entr and for s
Stability	Low (high variance)	High	Good	High
Sample Efficiency	Low	Low	Low	High