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

Taha Majlesi

July 17, 2025

Abstract

This document provides a comprehensive theoretical exploration of policy gradient methods in reinforcement learning, charting a course from the foundational REINFORCE algorithm to the state-of-the-art Soft Actor-Critic. It is structured to build understanding progressively, starting with the basic principles of policy gradients, analyzing their inherent limitations, and detailing the successive innovations designed to overcome these challenges. The exposition culminates in a detailed analysis of the maximum entropy framework and the theoretical guarantees underpinning the Soft Actor-Critic algorithm.

Contents

Chapter 1

Foundations of Policy Gradients: The REINFORCE Algorithm

This chapter lays the essential groundwork for all policy gradient methods. We will derive the fundamental theorem that makes direct policy optimization possible and introduce its most direct implementation, REINFORCE. We will then critically analyze its primary weakness—high variance—and explore the foundational techniques used to mitigate it.

1.1 The Objective Function in Reinforcement Learning

In the framework of reinforcement learning (RL), an agent interacts with an environment over a sequence of discrete time steps. The agent's goal is to learn a behavior, or **policy**, that maximizes a cumulative measure of reward. Policy gradient methods approach this problem by directly parameterizing the policy and optimizing its parameters to achieve this goal.

Let the policy be denoted by $\pi_\theta(a_t|s_t)$, which represents the probability of taking action a_t in state s_t under the control of parameters θ. These parameters typically correspond to the weights of a neural network. The interaction between the agent and the environment generates a sequence of states and actions known as a **trajectory**, denoted by $\tau = (s_0, a_0, s_1, a_1, \dots)$. The probability of a particular trajectory occurring under policy π_θ is given by $p_\theta(\tau)$.

The fundamental objective in RL is to maximize the expected total reward. The total reward for a trajectory, $R(\tau)$, is the sum of rewards received over that trajectory, often discounted by a factor $\gamma \in [0, 1)$. Formally, this is expressed as:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[R(\tau)] = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right] \tag{1.1}$$

Here, T is the horizon of the episode, and $r(s_t, a_t)$ is the reward received at time step t. This objective function, $J(\theta)$, represents the performance of the policy π_θ. The core task of policy gradient methods is to find the parameters θ^* that maximize this function, $\theta^* = \arg\max_\theta J(\theta)$. This is typically achieved through **gradient ascent**, where the parameters are iteratively updated in the direction of the gradient of the objective function, $\nabla_\theta J(\theta)$.

The probabilistic nature of this objective is critical. The expectation $\mathbb{E}_{\tau \sim p_\theta(\tau)}$ depends on the distribution of trajectories, $p_\theta(\tau)$, which is itself a function of the policy parameters

θ. This dependency complicates the computation of the gradient, as changing θ alters the very distribution over which the expectation is taken. The next section addresses this central challenge.

1.2 The Policy Gradient Theorem: Derivation and the Log-Derivative Trick

The primary challenge in optimizing $J(\theta)$ is computing its gradient, $\nabla_\theta J(\theta)$. A naive approach would involve differentiating through the dynamics of the environment, which are often unknown and complex. The **Policy Gradient Theorem** provides an elegant way to compute this gradient without needing to know the environment's transition model, by transforming the gradient of an expectation into an expectation of a gradient. This is made possible by a mathematical sleight of hand known as the **log-derivative trick**.

The derivation begins by expressing the objective function as an integral (or sum, in the discrete case) over the space of all possible trajectories:

$$J(\theta) = \int p_\theta(\tau) R(\tau) d\tau \tag{1.2}$$

Taking the gradient with respect to θ, and assuming we can swap the gradient and integral operators (a condition that generally holds under mild assumptions via the Leibniz integral rule), we have:

$$\nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) R(\tau) d\tau \tag{1.3}$$

At this stage, we are left with the term $\nabla_\theta p_\theta(\tau)$, which is difficult to work with directly. The log-derivative trick provides a way to reintroduce $p_\theta(\tau)$ into the expression. The trick is based on a simple identity from calculus for the derivative of a logarithm: $\nabla_x \log f(x) = \frac{\nabla_x f(x)}{f(x)}$. Rearranging this gives $\nabla_x f(x) = f(x) \nabla_x \log f(x)$. Applying this identity to our gradient, we get:

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) \tag{1.4}$$

Substituting this back into the gradient of the objective function yields:

$$\nabla_\theta J(\theta) = \int p_\theta(\tau) (\nabla_\theta \log p_\theta(\tau)) R(\tau) d\tau \tag{1.5}$$

This expression is now in the form of an expectation. Specifically, it is the expectation of the quantity $(\nabla_\theta \log p_\theta(\tau)) R(\tau)$ under the distribution $p_\theta(\tau)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [(\nabla_\theta \log p_\theta(\tau)) R(\tau)] \tag{1.6}$$

The final step is to simplify the term $\nabla_\theta \log p_\theta(\tau)$. The probability of a trajectory τ is the product of the initial state probability and the probabilities of all subsequent state-action transitions:

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t) \tag{1.7}$$

Taking the logarithm transforms this product into a sum:

$$\log p_\theta(\tau) = \log p(s_0) + \sum_{t=0}^{T-1} (\log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)) \tag{1.8}$$

When we take the gradient with respect to θ, the terms that do not depend on θ—namely, the initial state distribution $p(s_0)$ and the environment dynamics $p(s_{t+1}|s_t, a_t)$—vanish. This leaves us with:

$$\nabla_\theta \log p_\theta(\tau) = \nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{1.9}$$

Substituting this final piece back into our expectation gives the canonical form of the **Policy Gradient Theorem**:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) R(\tau) \right] \tag{1.10}$$

This result is powerful because it provides a way to estimate the policy gradient using only samples from the policy's interaction with the environment, without any knowledge of the environment's model. The term $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is often called the "score function" and is readily computable for most parameterized policies, such as those represented by neural networks.

1.3 REINFORCE: Monte Carlo Policy Gradients

The **REINFORCE** algorithm, also known as Monte Carlo Policy Gradients, is the most direct and foundational implementation of the Policy Gradient Theorem. It operates by approximating the expectation in the policy gradient expression with a sample mean calculated from a batch of trajectories collected by executing the current policy.

Given the policy gradient expression:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) R(\tau) \right] \tag{1.11}$$

REINFORCE approximates this gradient by generating a set of N trajectories, $\{\tau_1, \tau_2, \ldots, \tau_N\}$, and computing the empirical average:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) R(\tau_i) \right] \tag{1.12}$$

The term $R(\tau_i)$ is the total return of the i-th trajectory. This formulation can be re-arranged by swapping the sums, leading to a more common representation where the gradient update is considered at each time step:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) R(\tau_i) \tag{1.13}$$

The core idea of REINFORCE is to "reinforce" actions based on the total outcome of the episode. If a trajectory τ_i resulted in a high total reward $R(\tau_i)$, the algorithm increases the probabilities of all actions taken in that trajectory by adjusting θ in the direction of the score function $\nabla_\theta \log \pi_\theta$. Conversely, if the total reward was low (or negative), the probabilities of those actions are decreased.

The complete REINFORCE algorithm can be summarized in the following steps:

1. **Initialization:** Initialize the policy parameters θ randomly.

2. **Sampling:** For a number of episodes, run the current policy π_θ in the environment to collect a batch of N trajectories $\{\tau_1, \ldots, \tau_N\}$. For each trajectory, store the sequence of states, actions, and rewards.

3. **Gradient Estimation:** For each trajectory τ_i in the batch, compute the total return $R(\tau_i) = \sum_{t=0}^{T-1} r(s_{i,t}, a_{i,t})$. Then, compute the policy gradient estimate using the formula above.

4. **Parameter Update:** Update the policy parameters using a step of gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{1.14}$$

   where α is a learning rate.

5. **Repeat:** Go back to step 2 and repeat the process until the policy converges.

While REINFORCE is theoretically sound and provides an unbiased estimate of the policy gradient, it suffers from a major practical drawback: extremely high variance. The next section delves into the sources of this variance and the progressive refinements developed to create more stable and efficient algorithms.

1.4 Addressing High Variance

The vanilla REINFORCE estimator, while unbiased, is notoriously noisy. The high variance stems from the fact that the return $R(\tau)$ is a random variable that depends on a long sequence of stochastic actions and state transitions. A single high-reward (or low-reward) trajectory can drastically swing the gradient estimate, leading to unstable training. The following sections describe a logical progression of techniques designed to strip away irrelevant information from the reward signal, thereby reducing variance and improving the stability and sample efficiency of policy gradient methods.

1.4.1 Causality and Reward-to-Go

A primary source of variance in the REINFORCE algorithm is its flawed credit assignment. The estimator $(\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t))R(\tau)$ assigns the same credit, the total return of the entire episode $R(\tau)$, to every action taken within that episode. This is intuitively incorrect. An action a_t taken at time step t can only influence rewards that occur at or after time t. It cannot possibly affect rewards that have already been received $(r_0, r_1, \ldots, r_{t-1})$. This principle is known as **causality**. Including past rewards in the credit for action a_t contributes nothing but noise to the gradient estimate.

To address this, we can modify the policy gradient estimator to only include rewards that occur after the action is taken. This modified return is known as the **reward-to-go**, denoted G_t or $\hat{Q}^\pi(s_t, a_t)$. It is defined as the sum of future rewards from time step t onwards:

$$\hat{Q}^\pi(s_t, a_t) = G_t = \sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \tag{1.15}$$

The policy gradient formulation is then updated to use this reward-to-go as the weighting factor for the score function at each time step:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t \right] \tag{1.16}$$

The corresponding Monte Carlo estimator becomes:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) \right) \tag{1.17}$$

It can be formally proven that this modification does not introduce any bias into the gradient estimate. The expected value of the terms we removed is zero. For any time step t, the expectation of the gradient contribution from past rewards can be shown to be zero due to the independence of past rewards from the current action, given the current state. By removing these zero-expectation, high-variance terms, the reward-to-go estimator achieves a significant variance reduction compared to the vanilla REINFORCE estimator, leading to more stable and faster learning.

1.4.2 Baselines and the Advantage Function

While reward-to-go improves credit assignment, it does not solve all sources of variance. Consider an environment where all rewards are positive, for instance, always between 100 and 110. In this scenario, every action will receive a positive reward-to-go, and thus every action will be reinforced (i.e., its probability will be increased). The learning signal is not about whether an action was "good" in an absolute sense, but whether it was *better than average* for the situation it was in.

This motivates the introduction of a **baseline**, $b(s_t)$, which is a function that depends only on the state s_t and is subtracted from the reward-to-go. The policy gradient becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)(G_t - b(s_t)) \right] \tag{1.18}$$

Subtracting a baseline does not introduce bias into the gradient estimate. This can be proven by showing that the expectation of the subtracted term is zero:

$$\begin{aligned} \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t) \right] &= \sum_{t=0}^{T-1} \mathbb{E}_{s_t \sim p_\theta(s_t)} \left[\mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)} [\nabla_\theta \log \pi_\theta(a_t|s_t)] b(s_t) \right] \\ &= \sum_{t=0}^{T-1} \mathbb{E}_{s_t \sim p_\theta(s_t)} \left[b(s_t) \mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)} [\nabla_\theta \log \pi_\theta(a_t|s_t)] \right] \end{aligned}$$

The inner expectation is $\mathbb{E}_{a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t)] = \int \pi_\theta(a_t|s_t) \frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)} da_t = \int \nabla_\theta \pi_\theta(a_t|s_t) da_t = \nabla_\theta \int \pi_\theta(a_t|s_t) da_t = \nabla_\theta(1) = 0$. Thus, the entire expression is zero, and the estimator remains unbiased.

While any function of state can serve as a baseline, the optimal choice for variance reduction is the true **state-value function**, $V^\pi(s_t)$. The state-value function is defined as the expected reward-to-go from state s_t: $V^\pi(s_t) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[G_t|s_t]$. Using $V^\pi(s_t)$ as

the baseline centers the returns around zero, reinforcing actions that lead to better-than-average outcomes and penalizing those that lead to worse-than-average outcomes.

The quantity $G_t - V^\pi(s_t)$ is an estimate of the **Advantage Function**, $A^\pi(s_t, a_t)$. The advantage function is formally defined as the difference between the action-value function and the state-value function:

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \tag{1.19}$$

where $Q^\pi(s_t, a_t) = \mathbb{E}[G_t | s_t, a_t]$. Using a Monte Carlo estimate where G_t is a sample of $Q^\pi(s_t, a_t)$, we arrive at the advantage-based policy gradient formulation:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) A^\pi(s_t, a_t) \right] \tag{1.20}$$

In practice, $A^\pi(s_t, a_t)$ is estimated, often using a learned value function approximator (a **critic**), which leads to the family of **Actor-Critic** algorithms. This progression—from total reward to reward-to-go to the advantage function—represents a systematic effort to refine the learning signal, reducing variance at each step to enable more stable and efficient policy optimization.

Chapter 2

Towards Monotonic Improvement: Policy Gradients as Policy Iteration

This chapter reframes policy gradient methods through the lens of classic policy iteration, providing a deeper theoretical understanding of how they achieve policy improvement. This perspective reveals a critical challenge—the state distribution mismatch—and motivates the development of trust-region methods that offer theoretical guarantees on performance improvement.

2.1 A New Perspective on Policy Updates

Classic **Policy Iteration (PI)** is a fundamental dynamic programming algorithm that finds an optimal policy by alternating between two steps: policy evaluation and policy improvement. In policy evaluation, the value function of the current policy is computed. In policy improvement, a new, better policy is formed by acting greedily with respect to the computed value function.

Policy gradient methods, particularly those employing an advantage function (Actor-Critic methods), can be viewed as a stochastic, approximate version of this process. The main steps of a typical policy gradient algorithm align closely with the PI framework:

- **Policy Evaluation (Approximate):** The algorithm estimates the advantage function, $\hat{A}^{\pi}(s_t, a_t)$, for the current policy π. This is analogous to the policy evaluation step in PI. In practice, this is done by generating samples (running the policy) and fitting a model (a critic) to estimate the returns or advantages.

- **Policy Improvement (Approximate):** The algorithm uses the estimated advantage function $\hat{A}^{\pi}(s_t, a_t)$ to compute a policy gradient and take a step to update the policy parameters from θ to θ', resulting in a new policy π'. This is analogous to the policy improvement step.

This perspective is more than just an analogy; it provides a theoretical foundation for understanding why and how policy gradient updates lead to better policies. It shifts the focus from simple gradient ascent on an objective function to a more structured, iterative process of evaluating and improving the policy, which allows for formal guarantees on performance.

2.2 The Performance Difference Lemma: A Formal Guarantee

To formalize the connection to policy iteration, we need to show that a policy update step actually leads to a better policy. A key result, sometimes called the **Performance Difference Lemma**, provides an exact expression for the change in expected return when moving from an old policy π_θ to a new policy $\pi_{\theta'}$.

The claim is that the difference in performance is equal to the expected (discounted) sum of advantages of the old policy, where the expectation is taken over the state-action distribution induced by the *new* policy:

$$J(\theta') - J(\theta) = \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_\theta}(s_t, a_t) \right] \tag{2.1}$$

This identity is central to understanding policy improvement. It states that if, on average, the new policy selects actions that have a positive advantage according to the old policy's value function, then the new policy is guaranteed to have a higher expected return.

Proof of the Performance Difference Lemma. We begin by expressing the performance difference. The objective $J(\theta)$ is the expected value of the initial state, $J(\theta) = \mathbb{E}_{s_0 \sim p(s_0)}[V^{\pi_\theta}(s_0)]$. Since the initial state distribution $p(s_0)$ is independent of the policy, we can write:

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

Next, we use the telescoping sum property of the value function. For any trajectory, the value at the start can be expressed as the sum of single-step changes in value: $V(s_0) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t (V(s_t) - \gamma V(s_{t+1}))]$. Substituting this into our expression gives:

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

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

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

Combining the expectations and grouping terms by the summation over t:

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

The term inside the parentheses is precisely the definition of the advantage function for the old policy, $A^{\pi_\theta}(s_t, a_t)$. This completes the proof:

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

□

2.3 The State Distribution Mismatch Problem

The Performance Difference Lemma provides a clear target for optimization: find a new policy $\pi_{\theta'}$ that maximizes the expected advantage. However, there is a fundamental practical challenge embedded in this expression. The expectation, $\mathbb{E}_{\tau\sim p_{\theta'}(\tau)}$, is taken over trajectories generated by the *new* policy $\pi_{\theta'}$, while the advantage function, A^{π_θ}, is calculated with respect to the *old* policy π_θ.

This creates a "moving target" problem. To find the optimal new parameters θ', we need to maximize an objective that depends on the state visitation distribution $p_{\theta'}(s_t)$. But this distribution is itself a function of θ'. We cannot simply sample from $\pi_{\theta'}$ to estimate the expectation, because $\pi_{\theta'}$ is the very thing we are trying to find.

This issue is known as the **state distribution mismatch**. It is a core theoretical challenge in reinforcement learning, particularly for off-policy methods. On-policy algorithms like REINFORCE circumvent this by collecting new samples from the latest policy for every update, which is highly sample-inefficient. Off-policy methods, which aim to reuse old data, must find a way to correct for this mismatch. The discrepancy between the state distribution of the data-generating (behavior) policy and the state distribution of the policy being evaluated (target policy) can lead to biased and high-variance gradient estimates if not properly handled.

2.4 Importance Sampling and the Surrogate Objective

A standard statistical technique for handling a mismatch between sampling and target distributions is **importance sampling (IS)**. The core idea is to re-weight samples from a proposal distribution $p(x)$ to estimate an expectation under a target distribution $q(x)$, using the identity:

$$\mathbb{E}_{x\sim q(x)}[f(x)] = \int q(x)f(x)dx = \int p(x)\frac{q(x)}{p(x)}f(x)dx = \mathbb{E}_{x\sim p(x)}\left[\frac{q(x)}{p(x)}f(x)\right] \tag{2.2}$$

The ratio $q(x)/p(x)$ is the importance weight.

We can apply this to the performance difference expression. The full expectation is over trajectories, but we can decompose it into expectations over states and actions. The mismatch arises from both the action selection distribution ($\pi_{\theta'}$ vs. π_θ) and the state visitation distribution ($p_{\theta'}(s_t)$ vs. $p_\theta(s_t)$). We can use IS to change the expectation over actions from $\pi_{\theta'}$ to π_θ:

$$\mathbb{E}_{a_t\sim\pi_{\theta'}(a_t|s_t)}[\gamma^t A^{\pi_\theta}(s_t,a_t)] = \mathbb{E}_{a_t\sim\pi_\theta(a_t|s_t)}\left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)}\gamma^t A^{\pi_\theta}(s_t,a_t)\right] \tag{2.3}$$

This leaves us with an expectation over states visited by the new policy, $\mathbb{E}_{s_t\sim p_{\theta'}(s_t)}$. To make the optimization problem tractable, we make a critical approximation: we assume that the state visitation distributions are approximately equal, $p_{\theta'}(s_t) \approx p_\theta(s_t)$. This approximation is reasonable only if the new policy $\pi_{\theta'}$ is not too different from the old policy π_θ.

By ignoring the change in state distribution, we arrive at a **surrogate objective function**, $L_{\pi_\theta}(\pi_{\theta'})$, which approximates the true performance improvement but is com-

putable using data from the old policy π_θ:

$$J(\theta') - J(\theta) \approx L_{\pi_\theta}(\pi_{\theta'}) = \sum_t \mathbb{E}_{s_t \sim p_\theta(s_t)} \mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)} \left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t) \right] \tag{2.4}$$

Maximizing this surrogate objective forms the basis of many modern policy gradient algorithms. The gradient of this objective at $\theta' = \theta$ is exactly the policy gradient we derived earlier. However, taking a large step by maximizing L can violate the assumption that $p_{\theta'}(s_t) \approx p_\theta(s_t)$, potentially leading to a decrease in the true performance $J(\theta)$.

2.5 Bounding the Error for Guaranteed Improvement

To transform the approximation into a rigorous guarantee, we must bound the error introduced by ignoring the state distribution mismatch. This requires formalizing the notion of "closeness" between the old policy π_θ and the new policy $\pi_{\theta'}$ and analyzing how this difference propagates to the state distributions.

Let's define the maximum pointwise difference between the policies as $\max_{s,a} |\pi_{\theta'}(a|s) - \pi_\theta(a|s)| \leq \epsilon$. This condition implies that at any given state, the probability of the new policy taking a different action than the old policy is bounded. This difference accumulates over time. For a deterministic policy, if the probability of making a "mistake" (deviating from the old policy) at any step is ϵ, the probability of having followed the old policy's path for t steps is $(1-\epsilon)^t$. The probability of having deviated at least once is $1 - (1-\epsilon)^t$. This leads to a bound on the total variation distance between the state distributions:

$$|p_{\theta'}(s_t) - p_\theta(s_t)| \leq 2(1 - (1-\epsilon)^t) \tag{2.5}$$

Using the useful inequality $(1-\epsilon)^t \geq 1 - \epsilon t$ for $\epsilon \in [0, 1]$, we can establish a simpler, albeit looser, bound:

$$|p_{\theta'}(s_t) - p_\theta(s_t)| \leq 2\epsilon t \tag{2.6}$$

This bound on the state distribution allows us to connect the true performance improvement, $J(\theta') - J(\theta)$, to the surrogate objective, $L_{\pi_\theta}(\pi_{\theta'})$. The difference between the two is an error term that can be bounded. The analysis shows this error is proportional to the divergence between the policies. This leads to a lower bound on the performance improvement, which is the cornerstone of trust region methods:

$$J(\theta') - J(\theta) \geq L_{\pi_\theta}(\pi_{\theta'}) - C \cdot D_{\text{KL}}^{\max}(\pi_\theta, \pi_{\theta'}) \tag{2.7}$$

Here, $D_{\text{KL}}^{\max}(\pi_\theta, \pi_{\theta'})$ is the maximum KL-divergence between the policies over all states, and C is a constant that depends on the discount factor and maximum reward.

This inequality is profound. It guarantees that if we find a policy $\pi_{\theta'}$ that maximizes the right-hand side, we are guaranteed to achieve a monotonic improvement in the true objective $J(\theta)$. This transforms the optimization problem into:

$$\max_{\theta'} \left(L_{\pi_\theta}(\pi_{\theta'}) - C \cdot D_{\text{KL}}^{\max}(\pi_\theta, \pi_{\theta'}) \right) \tag{2.8}$$

This is equivalent to maximizing the surrogate objective subject to a penalty or constraint on the KL-divergence. This is the core idea behind **Trust Region Policy Optimization (TRPO)**, which seeks to maximize the surrogate objective within a "trust region" where the KL-divergence is bounded by a small constant δ: $D_{\text{KL}}(\pi_\theta || \pi_{\theta'}) \leq \delta$. By carefully controlling the size of the policy update, these methods can make principled, guaranteed steps toward the optimal policy.

Chapter 3

Soft Actor-Critic: An Off-Policy Algorithm for Maximum Entropy RL

Soft Actor-Critic (SAC) is a state-of-the-art, off-policy actor-critic algorithm that has demonstrated remarkable performance in terms of sample efficiency and stability, particularly in continuous control tasks. Its innovation lies in integrating the **maximum entropy reinforcement learning** framework with a stable actor-critic architecture. This chapter details the theoretical underpinnings of SAC, its architecture, and the specific loss functions that drive its learning process.

3.1 The Maximum Entropy Reinforcement Learning Framework

Standard reinforcement learning aims to find a policy that maximizes the expected cumulative reward. In contrast, the maximum entropy RL framework modifies this objective to include a term that encourages policy randomness, or entropy. The objective function in this framework is to maximize the sum of both the expected reward and the policy's entropy at each time step, weighted by a **temperature parameter** α:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{3.1}$$

Here, ρ_π is the state-action marginal distribution induced by policy π, and $\mathcal{H}(\pi(\cdot|s_t))$ is the entropy of the policy distribution at state s_t, defined as $\mathcal{H}(P) = \mathbb{E}_{x\sim P}[-\log P(x)]$.

The temperature parameter α controls the relative importance of the entropy term versus the reward. A higher α encourages more exploration and stochasticity, while an α of zero recovers the standard RL objective.

The intuition behind this objective is multifaceted:

- **Enhanced Exploration:** By rewarding entropy, the agent is incentivized to explore more widely. It discourages the policy from collapsing prematurely to a deterministic, suboptimal strategy, helping it to discover better solutions.

- **Robustness and Transferability:** A policy that maintains high entropy is less committed to a single mode of behavior. This can make it more robust to perturbations in the environment and allow it to adapt more quickly if the task changes. If multiple actions or paths yield similar high rewards, a maximum entropy policy will learn to be capable of taking all of them.
- **Improved Stability:** The entropy term acts as a regularizer, which can lead to smoother and more stable training dynamics.

3.2 Soft Value Functions

The introduction of the entropy term into the objective function necessitates a modification of the standard Bellman equations and value functions. These are referred to as "soft" value functions.

The **soft state-value function**, $V^{\text{soft}}(s_t)$, is defined as the expected future return from state s_t, including the entropy bonuses at all future steps. It is related to the soft Q-value via the following expectation:

$$V^{\text{soft}}(s_t) = \mathbb{E}_{a_t \sim \pi}[Q^{\text{soft}}(s_t, a_t) - \alpha \log \pi(a_t|s_t)] \tag{3.2}$$

The term $-\alpha \log \pi(a_t|s_t)$ can be seen as the entropy contribution for a single action sample.

The **soft action-value function**, $Q^{\text{soft}}(s_t, a_t)$, represents the expected return after taking action a_t in state s_t, with all subsequent actions also chosen to maximize the entropy-augmented objective. It is defined by the **soft Bellman equation**:

$$Q^{\text{soft}}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p}[V^{\text{soft}}(s_{t+1})] \tag{3.3}$$

This equation is structurally identical to the standard Bellman equation, but it uses the soft state-value function V^{soft} as its target. By substituting the definition of V^{soft} into the Bellman equation, we can write the **soft Bellman backup operator**, $\mathcal{T}^\pi$, which maps a Q-function to an updated Q-function:

$$\mathcal{T}^\pi Q(s_t, a_t) \triangleq r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p, a_{t+1} \sim \pi}[Q(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1}|s_{t+1})] \tag{3.4}$$

This operator forms the basis for the policy evaluation step in the theoretical framework of Soft Policy Iteration.

3.3 SAC Algorithm and Architecture

Soft Actor-Critic is an off-policy, model-free actor-critic algorithm designed to optimize the maximum entropy objective. Being off-policy allows it to reuse past experiences stored in a replay buffer, leading to significant improvements in sample efficiency. The practical implementation of SAC involves several function approximators, typically neural networks, that are trained concurrently.

The key components are:

- **Policy Network (Actor)** $\pi_\phi(a|s)$**:** Parameterized by ϕ, this network outputs the parameters of a stochastic policy (e.g., the mean and standard deviation of a Gaussian distribution for continuous actions).

- **Two Soft Q-Function Networks (Critics) $Q_{\theta_1}(s,a)$ and $Q_{\theta_2}(s,a)$:** Parameterized by θ_1 and θ_2. SAC employs two separate Q-networks and uses the minimum of their predictions during the policy update. This technique, inspired by Clipped Double Q-Learning, helps to counteract the tendency of Q-learning methods to overestimate values, which can lead to instability and poor performance.

- **A Soft Value Function Network $V_\psi(s)$:** Parameterized by ψ. This network's primary role is to be the target for the policy update, as will be detailed in the loss functions.

- **A Target Value Function Network $V_{\bar{\psi}}(s)$:** This is a copy of the value network V_ψ whose parameters $\bar{\psi}$ are updated slowly via an exponential moving average (EMA) of the main value network's parameters: $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$. Using this "target" network provides a stable, slowly changing target for the Q-function updates, which is a crucial ingredient for stable training in off-policy learning.

The overall algorithm proceeds by collecting experience (state, action, reward, next state tuples) into a replay buffer $\mathcal{D}$. In each training step, a mini-batch of experiences is sampled from the buffer, and gradient descent is performed on the loss functions for the actor, critics, and value function networks.

3.4 The Loss Functions of SAC

The learning process in SAC is driven by minimizing a set of specific loss functions for each of the network components. These loss functions are derived directly from the soft Bellman equations and the maximum entropy objective.

Value Function Loss $J_V(\psi)$: This is a mean-squared error (MSE) loss that trains the value network V_ψ. The target for the value function at state s_t is derived directly from the definition of the soft V-value. It is the expected value of the soft Q-function minus the entropy term, where the expectation is over actions sampled from the current policy π_ϕ. The Q-value here is taken from the critic networks. In practice, the minimum of the two Q-networks is used to be consistent with the policy update.

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\frac{1}{2} \left(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\phi} \left[\min_{i=1,2} Q_{\theta_i}(s_t, a_t) - \alpha \log \pi_\phi(a_t|s_t) \right] \right)^2 \right] \tag{3.5}$$

Q-Function Loss $J_Q(\theta)$: This is also an MSE loss, training the two critic networks Q_{θ_1} and Q_{θ_2}. The target, $\hat{Q}(s_t, a_t)$, is constructed using the soft Bellman equation. It is the sum of the immediate reward $r(s_t, a_t)$ and the discounted soft value of the next state, $\gamma V_{\bar{\psi}}(s_{t+1})$. Crucially, the value of the next state is provided by the slowly-updating target value network, $V_{\bar{\psi}}$, which stabilizes the learning process.

$$J_Q(\theta_i) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}} \left[\frac{1}{2} \left(Q_{\theta_i}(s_t, a_t) - (r(s_t, a_t) + \gamma V_{\bar{\psi}}(s_{t+1})) \right)^2 \right] \quad \text{for } i = 1, 2 \tag{3.6}$$

Policy Loss $J_\pi(\phi)$: The policy improvement step aims to update the policy to act more optimally according to the current estimate of the soft Q-function. The objective is to find a new policy that minimizes the KL-divergence to the exponentiated Q-function (a Boltzmann distribution). This objective can be written as:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}, a_t \sim \pi_\phi} \left[\alpha \log(\pi_\phi(a_t|s_t)) - \min_{i=1,2} Q_{\theta_i}(s_t, a_t) \right] \tag{3.7}$$

Maximizing this objective is equivalent to minimizing the KL divergence. The gradient of this objective is computed using the **reparameterization trick**. Instead of sampling an action a_t directly from π_ϕ, the action is expressed as a deterministic function of the state and some independent noise, e.g., $a_t = f_\phi(\epsilon; s_t)$ where $\epsilon \sim \mathcal{N}(0, I)$. This allows the gradient to be backpropagated through the actor network and the critic networks, resulting in a low-variance gradient estimate.

The interplay of these components—off-policy updates, entropy maximization, and stable target networks—is what grants SAC its impressive performance and stability.

Chapter 4

Theoretical Guarantees for Soft Actor-Critic

This final chapter provides the theoretical backbone for SAC, demonstrating through a series of proofs that the algorithm is guaranteed to converge to the optimal policy under the maximum entropy objective. This is achieved through the framework of Soft Policy Iteration, which mirrors classical policy iteration but is adapted for the entropy-regularized setting.

4.1 Soft Policy Iteration

Soft Policy Iteration is a theoretical procedure that serves as the foundation for proving the convergence of Soft Actor-Critic. Analogous to classical Policy Iteration, it is an idealized algorithm that alternates between two core steps until convergence:

1. **Soft Policy Evaluation:** For a fixed policy π, compute its corresponding soft Q-function, Q^π. This is achieved by iterating the soft Bellman backup operator to convergence.

2. **Soft Policy Improvement:** Given the soft Q-function Q^π, find a new policy π_{new} that is an improvement over the old one.

The subsequent sections will prove that each of these steps is guaranteed to either improve or maintain the policy's performance, leading to a monotonic convergence towards the optimal policy within the maximum entropy framework.

4.2 Lemma 1: Soft Policy Evaluation

The first step in establishing the convergence of Soft Policy Iteration is to show that the evaluation step is well-defined and converges to the correct value function.

Lemma 4.1 (Soft Policy Evaluation)**.** *Consider the soft Bellman backup operator $\mathcal{T}^\pi$ for a given policy π. For any initial Q-function mapping $Q_0 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, the sequence defined by $Q_{k+1} = \mathcal{T}^\pi Q_k$ will converge to the true soft Q-value of policy π, denoted Q^π, as $k \to \infty$. This holds assuming a finite action space $|\mathcal{A}| < \infty$ to ensure bounded entropy.*

The soft Bellman backup operator is defined as:

$$\mathcal{T}^{\pi}Q(s_t, a_t) \triangleq r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p}[V^{\pi}(s_{t+1})] \tag{4.1}$$

where the soft state-value function is $V^{\pi}(s_t) = \mathbb{E}_{a_t\sim\pi}[Q(s_t, a_t) - \alpha \log \pi(a_t|s_t)]$.

Proof of Lemma 1. The proof relies on showing that the soft Bellman operator is a contraction mapping, which, by the Banach fixed-point theorem, guarantees convergence to a unique fixed point. The provided material offers an elegant way to demonstrate this by recasting the problem into the standard policy evaluation framework.

Define an Entropy-Augmented Reward: We can absorb the future entropy term into a modified reward function. Let the soft reward be defined as:

$$r_{\text{soft}}(s_t, a_t) \triangleq r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p}[\alpha \mathcal{H}(\pi(\cdot|s_{t+1}))] \tag{4.2}$$

where $\mathcal{H}(\pi(\cdot|s_{t+1})) = \mathbb{E}_{a_{t+1}\sim\pi}[-\log \pi(a_{t+1}|s_{t+1})]$.

Rewrite the Soft Bellman Update: We can rewrite the soft Bellman backup operator using this soft reward.

$$\begin{aligned}
\mathcal{T}^{\pi}Q(s_t, a_t) &= r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p, a_{t+1}\sim\pi}[Q(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1}|s_{t+1})] \\
&= r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p}[\alpha \mathcal{H}(\pi(\cdot|s_{t+1}))] + \gamma \mathbb{E}_{s_{t+1}\sim p, a_{t+1}\sim\pi}[Q(s_{t+1}, a_{t+1})] \\
&= r_{\text{soft}}(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p, a_{t+1}\sim\pi}[Q(s_{t+1}, a_{t+1})]
\end{aligned}$$

Apply Standard Convergence Results: This final expression is identical in form to the standard Bellman backup operator, but with the reward function r replaced by the entropy-augmented reward r_{soft}. The standard policy evaluation operator is a well-known γ-contraction in the infinity norm, provided the rewards are bounded. The assumption of a finite action space ensures that the entropy term $\mathcal{H}$ is bounded, and thus r_{soft} is also bounded. Therefore, the soft Bellman backup operator $\mathcal{T}^{\pi}$ is also a γ-contraction, which guarantees that repeated application will converge to its unique fixed point, which is the soft Q-function Q^{π}. □

4.3 Lemma 2: Soft Policy Improvement

The second step is to prove that the policy improvement step results in a new policy that is at least as good as, or better than, the old policy in terms of its soft Q-value.

Lemma 4.2 (Soft Policy Improvement)**.** *Let $\pi_{old} \in \Pi$ be a policy, and let π_{new} be the policy obtained by the soft policy improvement step:*

$$\pi_{new} = \arg\min_{\pi'\in\Pi} D_{KL}\left(\pi'(\cdot|s_t) \middle\| \frac{\exp(\frac{1}{\alpha}Q^{\pi_{old}}(s_t, \cdot))}{Z^{\pi_{old}}(s_t)}\right) \tag{4.3}$$

where Z is the partition function that normalizes the distribution. Then, $Q^{\pi_{new}}(s_t, a_t) \geq Q^{\pi_{old}}(s_t, a_t)$ for all $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$ (assuming $|\mathcal{A}| < \infty$).

Proof of Lemma 2. The proof proceeds by first establishing a key inequality and then using it iteratively within the soft Bellman equation.

Establish a Key Inequality: The policy π_{new} is defined as the minimizer of the KL divergence objective. Since we can always choose $\pi' = \pi_{\text{old}}$, it must be that the objective

value for π_{new} is less than or equal to the objective value for π_{old}. Expanding the KL divergence definition $D_{\text{KL}}(P\|Q) = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)]$, we get:

$$\mathbb{E}_{a_t\sim\pi_{\text{new}}}\left[\log \pi_{\text{new}}(a_t|s_t) - \left(\frac{1}{\alpha}Q^{\pi_{\text{old}}}(s_t,a_t) - \log Z^{\pi_{\text{old}}}(s_t)\right)\right] \leq \mathbb{E}_{a_t\sim\pi_{\text{old}}}\left[\log \pi_{\text{old}}(a_t|s_t) - \left(\frac{1}{\alpha}Q^{\pi_{\text{old}}}(s_t,a_t)\right.\right.$$

Since the partition function $Z^{\pi_{\text{old}}}(s_t)$ and temperature α are constant with respect to the action, we can simplify and rearrange this inequality to obtain:

$$\mathbb{E}_{a_t\sim\pi_{\text{new}}}\left[Q^{\pi_{\text{old}}}(s_t,a_t) - \alpha\log\pi_{\text{new}}(a_t|s_t)\right] \geq \mathbb{E}_{a_t\sim\pi_{\text{old}}}\left[Q^{\pi_{\text{old}}}(s_t,a_t) - \alpha\log\pi_{\text{old}}(a_t|s_t)\right]$$

The right-hand side is the definition of the soft state-value function $V^{\pi_{\text{old}}}(s_t)$. Thus, we have the crucial inequality:

$$\mathbb{E}_{a_t\sim\pi_{\text{new}}}\left[Q^{\pi_{\text{old}}}(s_t,a_t) - \alpha\log\pi_{\text{new}}(a_t|s_t)\right] \geq V^{\pi_{\text{old}}}(s_t) \tag{4.4}$$

Apply the Inequality Iteratively: Now, consider the soft Bellman equation for $Q^{\pi_{\text{old}}}(s_t,a_t)$:

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

We can apply the inequality derived in step 1 to the $V^{\pi_{\text{old}}}(s_{t+1})$ term:

$$Q^{\pi_{\text{old}}}(s_t,a_t) \leq r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim p}\left[\mathbb{E}_{a_{t+1}\sim\pi_{\text{new}}}[Q^{\pi_{\text{old}}}(s_{t+1},a_{t+1}) - \alpha\log\pi_{\text{new}}(a_{t+1}|s_{t+1})]\right]$$

The right-hand side of this inequality is precisely the soft Bellman backup operator for the new policy, $\mathcal{T}^{\pi_{\text{new}}}$, applied to the old Q-function, $Q^{\pi_{\text{old}}}$. So, we have shown:

$$Q^{\pi_{\text{old}}} \leq \mathcal{T}^{\pi_{\text{new}}}Q^{\pi_{\text{old}}}$$

Monotonicity and Convergence: Since $\mathcal{T}^{\pi_{\text{new}}}$ is a monotone operator (if $Q_1 \leq Q_2$, then $\mathcal{T}Q_1 \leq \mathcal{T}Q_2$), we can apply it repeatedly to both sides of the inequality:

$$Q^{\pi_{\text{old}}} \leq \mathcal{T}^{\pi_{\text{new}}}Q^{\pi_{\text{old}}} \leq (\mathcal{T}^{\pi_{\text{new}}})^2Q^{\pi_{\text{old}}} \leq \dots$$

From Lemma 1, we know that this sequence of applications converges to the fixed point of the operator, which is $Q^{\pi_{\text{new}}}$. Therefore, we can conclude that:

$$Q^{\pi_{\text{old}}}(s_t,a_t) \leq Q^{\pi_{\text{new}}}(s_t,a_t)$$

This completes the proof that each soft policy improvement step yields a policy with a soft Q-function that is pointwise greater than or equal to the previous one. □

4.4 Theorem 1: Convergence of Soft Policy Iteration

By combining the results of the two preceding lemmas, we can establish the main convergence theorem for Soft Policy Iteration.

Theorem 4.3 (Soft Policy Iteration)**.** *Repeated application of soft policy evaluation and soft policy improvement to any initial policy $\pi \in \Pi$ converges to an optimal policy π^* such that $Q^{\pi^*}(s_t,a_t) \geq Q^{\pi}(s_t,a_t)$ for all other policies $\pi \in \Pi$ and all state-action pairs (s_t,a_t), assuming a finite action space and bounded rewards.*

Proof of Theorem 1. The proof brings together the guarantees from the evaluation and improvement steps to show overall convergence to optimality.

Monotonicity: Let π_i be the policy at iteration i. By Lemma 2 (Soft Policy Improvement), we know that the corresponding soft Q-function is monotonically non-decreasing:

$$Q^{\pi_0} \leq Q^{\pi_1} \leq Q^{\pi_2} \leq \dots$$

Boundedness and Convergence: The rewards are assumed to be bounded. For a finite action space, the policy entropy is also bounded. Consequently, the soft Q-function Q^{π} is bounded from above for any policy $\pi \in \Pi$. A monotonically non-decreasing sequence that is bounded from above is guaranteed to converge to a limit. Therefore, the sequence of Q-functions Q^{π_i} converges to some limit Q-function, Q^{π^*}, and the sequence of policies converges to a corresponding limit policy π^*.

Optimality of the Limit Policy: We need to show that this converged policy π^* is indeed optimal. At convergence, the policy improvement step can no longer find a better policy. This means that π^* must be the minimizer of the KL-divergence objective with respect to its own Q-function:

$$\pi^* = \arg\min_{\pi' \in \Pi} D_{\mathrm{KL}} \left(\pi'(\cdot|s_t) \middle\| \frac{\exp(\frac{1}{\alpha} Q^{\pi^*}(s_t, \cdot))}{Z^{\pi^*}(s_t)} \right)$$

This implies that for any other policy $\pi \in \Pi$ where $\pi \neq \pi^*$, we must have $D_{\mathrm{KL}}(\pi^* \| \dots) < D_{\mathrm{KL}}(\pi \| \dots)$. Following the same argument as in the proof of Lemma 2, this strict inequality in the KL-divergence objective leads to a strict inequality in the soft Q-values:

$$Q^{\pi^*}(s_t, a_t) > Q^{\pi}(s_t, a_t)$$

for all (s_t, a_t). This shows that the soft value of any other policy in the set Π is strictly lower than that of the converged policy π^*.

Therefore, π^* is the optimal policy within the class Π for the maximum entropy objective. This theorem provides a strong theoretical foundation for Soft Actor-Critic, showing that its underlying algorithmic structure is principled and guaranteed to converge to an optimal solution in the tabular case. While the practical algorithm uses function approximation and does not run the evaluation and improvement steps to convergence, these theoretical guarantees provide confidence in its design and help explain its empirical stability and success. □