Week 7: Policy Gradient

Ardhito Nurhadyansah[1], Mohamad Arvin Fadriansyah[2], and Ian Suryadi Timothy H[3]

[1]2106750206
[2]2006596996
[3]2106750875

1 Policy Improvement in an Inexact Way

In reinforcement learning (RL), the ultimate goal is to find a policy π that maximizes the agent's reward, such as the discounted sum of rewards, average reward per time step, or total reward. A common approach is to iteratively improve an existing policy π^k to obtain a new policy π^{k+1} such that

$$v(\pi^{k+1}) \; \geq \; v(\pi^k),$$

where $v(\pi)$ denotes the value (under some optimality criterion) of policy π.

In many practical scenarios, we do not fully solve for an optimal policy at each iteration; instead, we perform *inexact* or partial improvements. As long as these updates monotonically increase or maintain the policy value, the overall process can converge to a better policy over time.

1.1 Policy Gradient Methods

In RL, one popular realization of inexact policy improvement is via *policy gradient methods*. The idea is to perform gradient *ascent* on the objective function $v_x(\pi)$ where x indicates the chosen optimality criterion (e.g., discounted return, average reward, total reward, or bias). Concretely, we seek

$$\nabla_\pi \, v_x(\pi) \quad \text{(or its approximation)},$$

and then update the policy in the direction of this gradient to increase performance:

$$\pi^{k+1} \; \leftarrow \; \pi^k \; + \; \alpha \, \nabla_\pi \, v_x(\pi^k),$$

where α is a step size (learning rate).

A direct application of this concept, called *direct policy parameterization*, attempts to compute partial derivatives of $v_x(\pi)$ with respect to the *policy* π itself. However, this approach can be difficult to use in practice because:

- Policies may not have a simple, differentiable form if they are defined implicitly (e.g., large tables or non-differentiable structures).
- Directly working in the space of all possible policies can be extremely high-dimensional and unstructured, making optimization unwieldy.

These challenges motivate an alternative known as *explicit policy parameterization*.

2 Motivation for Explicit Policy Parameterization

To address the difficulties of direct policy parameterization, we introduce a parameter vector

$$\theta \; \in \; \Theta \subseteq \mathbb{R}^d,$$

where d is the dimension of the parameter space. This parameter vector defines a *family* of policies:

$$\pi_\theta(a \mid s),$$

so that the value of the policy can be written as

$$v(\pi_\theta) \; = \; v(\theta).$$

By focusing on θ, we transform the problem of improving π into the (often more tractable) problem of optimizing θ via gradient ascent:

$$\theta^{k+1} \; \leftarrow \; \theta^k \; + \; \alpha \nabla_\theta \, v(\theta^k).$$

This shift from direct policy parameterization to exact policy parameterization has several advantages:

1. High-Dimensional Function Approximation: Using neural networks or other flexible approximators for π_θ allows us to represent complex policies that can generalize well across large state spaces.
2. Systematic Exploration Control: Stochastic parameterizations can naturally incorporate an exploration mechanism (similar in spirit to ε-greedy), enabling more principled control over the exploration-exploitation trade-off.
3. Compatibility with Gradient-Based Optimization: Once the policy is explicitly parameterized and differentiable, powerful optimization algorithms can be applied. Methods such as Proximal Policy Optimization (PPO), Trust Region Policy Optimization (TRPO), and other variants can be readily used.
4. Empirical Success in State-of-the-Art (SOTA) RL: Many of the best-performing RL algorithms in continuous control and high-dimensional settings rely on explicit policy parameterization. These algorithms have shown remarkable success in benchmarks such as robotic control tasks and complex games.

3 Policy Parameterization for Discrete Action Spaces

In reinforcement learning with discrete action spaces, we often represent the policy $\pi(a \mid s; \boldsymbol{\theta})$ using a parameterized categorical distribution. One commonly used approach is the **softmax (Gibbs/Boltzmann) parameterization**, where the probability of selecting an action a in state s is defined as:

$$\pi(a \mid s; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta}^\top \phi(s,a))}{\sum_{a' \in A} \exp(\boldsymbol{\theta}^\top \phi(s,a'))},$$

where:

- $\phi(s,a) \in \mathbb{R}^d$ is a feature vector for state-action pair (s,a),
- $\boldsymbol{\theta} \in \mathbb{R}^d$ is the parameter vector,
- $\mathscr{A}$ is the set of all possible discrete actions.

This softmax function ensures that:

- Each action probability is positive: $\pi(a \mid s; \boldsymbol{\theta}) > 0$,
- The probabilities sum to one: $\sum_{a \in A} \pi(a \mid s; \boldsymbol{\theta}) = 1$.

The distribution $\pi(\cdot \mid s; \boldsymbol{\theta})$ is also known as a **categorical distribution** over actions, and this formulation aligns with the Gibbs distribution in statistical physics, which favors higher-scoring actions (higher $\boldsymbol{\theta}^\top \phi(s,a)$).

Alternative Interpretation: Preference-Based Parameterization. Sutton (2018) also presents a preference-based view of softmax policy parameterization, where each action a is assigned a scalar preference $H_t(a) \in \mathbb{R}$. These preferences determine the policy via:

$$\pi_t(a) = \frac{\exp(H_t(a))}{\sum_{b \in A} \exp(H_t(b))}.$$

Only the relative differences between preferences matter. For example, adding a constant to all $H_t(a)$ values has no effect on the resulting action probabilities. This highlights the invariance property of the softmax formulation. Initially, all preferences are often set equally (e.g., $H_1(a) = 0$) to induce uniform exploration.

Connection to Logistic Function. In the case of two actions a_1 and a_2, this softmax reduces to a logistic (sigmoid) function:

$$\pi(a_1) = \frac{1}{1 + \exp(-(H(a_1) - H(a_2)))},$$

which is widely used in statistics and neural networks. This emphasizes the connection between policy gradients and logistic regression models.

Gradient of the Softmax Policy. When applying policy gradient methods, we need the gradient of the log-policy:

$$\nabla_{\boldsymbol{\theta}} \log \pi(a \mid s; \boldsymbol{\theta}) = \phi(s, a) - \sum_{a' \in A} \pi(a' \mid s; \boldsymbol{\theta}) \phi(s, a').$$

This expression follows from the quotient rule and is essential for computing the policy gradient in REINFORCE and actor-critic methods.

Interpretation. This softmax parameterization encourages exploration: even suboptimal actions have non-zero probability of being selected. The scale of $\boldsymbol{\theta}^\top \phi(s, a)$ can influence how deterministic or stochastic the policy behaves.

4 Gradient Ascent for Policy Improvement

As mentioned in the previous chapters, one of the way to achieve inexact policy improvement is via *gradient ascent*. Concretely, if $v(\theta)$ denotes the performance measure (e.g., expected return) of a parameterized policy π_θ, then a gradient ascent step can be written as:

$$\theta \;\leftarrow\; \theta \;+\; \alpha \, \nabla_\theta \, v(\theta), \tag{1}$$

where α is a step-size (learning rate) and $\nabla_\theta v(\theta)$ is the gradient of the performance measure with respect to the policy parameters θ. By moving in the direction of the gradient, we *increase* $v(\theta)$, thus improving the policy in an inexact yet systematic manner.

4.1 Policy Gradient Formula

A key question is how to compute $\nabla_\theta \, v(\theta)$ when the policy $\pi_\theta(a \mid s)$ is stochastic. Under certain regularity conditions, the *policy gradient theorem* states:

$$\nabla_\theta \, v(\theta) = \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) \sum_{a \in \mathcal{A}} q^{\pi_\theta}(s, a) \, \nabla_\theta \, \pi_\theta(a \mid s) \tag{2}$$

$$= \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) \sum_{a \in \mathcal{A}} q^{\pi_\theta}(s, a) \, \pi_\theta(a \mid s) \, \nabla_\theta \log \pi_\theta(a \mid s), \tag{3}$$

where:

- $d^{\pi_\theta}(s)$ is the (stationary) distribution of states when following policy π_θ.
- $q^{\pi_\theta}(s, a)$ is the action-value function under π_θ, i.e., the expected return starting from state s, taking action a, and thereafter following π_θ.
- $\log \pi_\theta(a \mid s)$ is used to transform $\nabla_\theta \, \pi_\theta(a \mid s)$ into a form more amenable to sampling.

In words, to compute the gradient of the policy performance, we weight each action a in state s by both its likelihood under π_θ and its corresponding value $q^{\pi_\theta}(s, a)$.

4.2 Sampling-Based (Unbiased) Estimation

Directly summing over all states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$ is often intractable, especially in large or continuous spaces. Instead, we typically approximate (3) via samples from the current policy π_θ. That is, we collect trajectories $\{(s_t, a_t, r_t)\}$ by interacting with the environment using π_θ, and then estimate:

$$\nabla_\theta v(\theta) \;\approx\; \frac{1}{N}\sum_{i=1}^{N}\Big[q^{\pi_\theta}\big(s_i, a_i\big)\,\nabla_\theta \log \pi_\theta\big(a_i \mid s_i\big)\Big], \tag{4}$$

where N is the number of sampled state-action pairs (or episodes).

If $q^{\pi_\theta}(s_i, a_i)$ is itself replaced by an unbiased return estimate (e.g., Monte Carlo returns, bootstrapped estimates, or advantage functions), then (4) is an *unbiased* stochastic gradient of $v(\theta)$. The entire procedure is thus referred to as *stochastic gradient ascent (SGA)*.

4.3 Biased vs. Semi-Stochastic Approaches

In practice, some approximations introduce bias. For instance, $q^{\pi_\theta}(s, a)$ may be approximated by a learned critic or truncated returns. Moreover, during learning, θ may change faster than we can gather fully on-policy samples. This mismatch between the policy used to gather data and the updated policy parameters can create a *semi-stochastic* gradient approach. While theoretically these approximations may introduce some bias, in many real-world tasks they are sufficiently accurate to yield strong performance and stable learning.

4.4 Putting It All Together

The gradient ascent update rule for inexact policy improvement can be summarized as:

$$\theta_{k+1} \;=\; \theta_k \;+\; \alpha\,\widehat{\nabla_\theta v(\theta_k)}, \tag{5}$$

where $\widehat{\nabla_\theta v(\theta_k)}$ is an approximation of the true gradient, computed via sampled trajectories or batches of experience. This approach has the following key characteristics:

- Inexact but Monotonically Improving: Each update step aims to increase $v(\theta)$, though not necessarily by an exact or optimal amount.
- Unbiased or Semi-Unbiased Estimates: Depending on how $q^{\pi_\theta}(s, a)$ is estimated, the gradient update may be unbiased (e.g., Monte Carlo returns) or partially biased (e.g., bootstrapped estimates, off-policy data).
- Computational Feasibility: Sampling-based methods are often the only viable approach in high-dimensional or continuous RL problems.
- Compatibility with Practical Hyperparameters: Even if the procedure is not strictly on-policy or if there is some mismatch in distributions, these methods still tend to work well in practice, especially with appropriate hyperparameter tuning.

5 Discounted Reward and the Policy Gradient Theorem

In reinforcement learning, when working with continuing tasks, we often aim to maximize the expected *discounted return*. This leads to defining the performance objective as:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{\infty}\gamma^t R_{t+1}\right] = \mathbf{P}_\pi^* \mathbf{r}_\pi,$$

where:

- $\boldsymbol{\theta} \in \mathbb{R}^d$ is the policy parameter,
- $\gamma \in [0, 1)$ is the discount factor,
- $\pi_{\boldsymbol{\theta}}$ is the parameterized policy,
- $\mathbf{P}_\pi^* = (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1}$ is the discounted resolvent (inverse Bellman operator),
- $\mathbf{r}_\pi \in \mathbb{R}^{|\mathcal{S}|}$ is the reward vector under policy π.

However, the distribution over states under this discounted formulation is no longer a proper probability distribution. Instead, it is a weighted occupancy distribution that reflects how often states are visited, discounted over time. Formally, this state distribution is:

$$\mu_\gamma(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \Pr(S_t = s),$$

which corresponds to a *geometric distribution* over state visitation (trial until first success). This reflects that recent states are weighted more heavily than distant ones.

5.1 How to Sample a State from the Discounted Distribution

Let $\widetilde{P}_{\widetilde{\pi}}^{\gamma} \propto (1-\gamma)P_\pi^\gamma$ denote the normalized discounted state distribution. To sample a single state $S \sim \widetilde{P}_\pi^\gamma$, follow this procedure:

1. Sample a length $L \sim \text{Geo}(1-\gamma)$, representing the number of steps to run the episode. This reflects the discounted weighting of time steps.
2. Run an episode (trajectory) from $t = 0$ to $t = L - 1$ using the current policy $\pi_{\boldsymbol{\theta}}$.
3. Let the sampled state be:
$$S_{L-1}.$$
All previous states can be discarded. However, in practice, people often use *all* states $\{S_0, \ldots, S_{L-1}\}$ to construct an unbiased gradient estimate more efficiently.

This method allows sampling from the discounted state distribution using a simple rejection-free procedure based on the geometric distribution.

5.2 Gradient of the Discounted Performance

When differentiating the performance objective with respect to parameters $\boldsymbol{\theta}$, a challenge arises:

- The derivative of the state distribution $\mu_\gamma(s)$ with respect to $\boldsymbol{\theta}$ is unknown.
- This term is not sampling friendly, which makes gradient estimation difficult.

To illustrate the problem, consider the gradient of the value function from a start state s_0:

$$\nabla_{\boldsymbol{\theta}} v_\gamma^\pi(s_0) = \sum_{s\in\mathcal{S}} (\mathbf{P}_\pi^\gamma)(s \mid s_0) \sum_{a\in\mathcal{A}} q_\gamma^\pi(s,a)\nabla_{\boldsymbol{\theta}} \log \pi(a \mid s; \boldsymbol{\theta}).$$

This formulation shows two key difficulties:

- The discounted transition matrix $\mathbf{P}_\pi^\gamma$ is not a proper probability distribution over states because its rows do not sum to 1.
- The dependency on future state visitation makes $\nabla_{\boldsymbol{\theta}} v_\gamma^\pi(s_0)$ nontrivial to estimate via sampling, since it relies on **off-policy-like expectations** over future trajectories.

Despite this, we can still derive a useful result. The **Policy Gradient Theorem** states:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu_\gamma(s) \sum_a \nabla_{\boldsymbol{\theta}} \pi(a|s;\boldsymbol{\theta}) q^\pi(s,a),$$

where the gradient does *not* involve the derivative of the state distribution $\mu_\gamma(s)$, and the proportionality constant depends on γ.

This result allows us to construct stochastic estimates of the policy gradient using sample trajectories.

5.3 Sampling-Friendly Estimation

Using the identity:
$$\nabla_{\boldsymbol{\theta}} \pi(a \mid s; \boldsymbol{\theta}) = \pi(a \mid s; \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log \pi(a \mid s; \boldsymbol{\theta}),$$
we can rewrite the gradient as:

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi[q^\pi(s,a)\nabla_{\boldsymbol{\theta}} \log \pi(a \mid s; \boldsymbol{\theta})],$$

which forms the basis of REINFORCE and other policy gradient algorithms.

5.4 REINFORCE with Discounted Return

In practice, for episodic tasks, we define the update rule as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t \mid S_t; \boldsymbol{\theta}_t),$$

where $G_t = \sum_{k=t}^{T} \gamma^{k-t} R_{k+1}$ is the sampled discounted return. This form emphasizes the time-weighted contribution of each reward.

5.5 Geometric Distribution Intuition

The discounted state distribution $\mu_\gamma(s)$ resembles a geometric distribution with success probability $1-\gamma$, as it defines the likelihood of visiting a state within discounted trials. This interpretation helps justify the form of the weighting in the gradient expression.

5.6 Conclusion

Although the discounted reward setting does not induce a proper probability distribution over states, the policy gradient theorem provides a powerful result that avoids this difficulty by leveraging the structure of the expected return and enabling sampling-based learning algorithms like REINFORCE.

6 Difference

6.1 Difference in State Distribution and Type of Reward

In many decision-making problems, there is an important distinction between two approaches for evaluating rewards:

- *Average reward*: This approach is often associated with a stationary state distribution. Over a long time horizon, the state distribution may converge to a stationary distribution, and the policy's performance is measured by the average reward achieved under that distribution.
- *Discounted reward*: This method applies a discount factor γ to future rewards. It can be interpreted as a process where the number of steps follows a Geometric distribution with parameter $(1-\gamma)$. In other words, there is a probability $(1-\gamma)$ of terminating at each step, thus placing more emphasis on recent states.

Additionally, the reward function itself can depend on either the state (s) or the state-action pair (s, a):

$$r(s,a) \quad \text{or} \quad r(s).$$

Some algorithms assume that the reward depends solely on the state, while others allow it to depend on both state and action. These differences impact how the policy π is evaluated.

6.2 Dependency of the Optimal Policy on the Initial State Distribution

The optimal policy π^* is often defined by taking into account the expectation over the **initial state distribution** p:

$$\pi^*(p) \;=\; \underset{\pi}{\operatorname{argmax}} \; \mathbb{E}_{S_0\sim p}\big[V^\pi(S_0)\big],$$

where $V^\pi(S_0)$ is the value of the policy π starting from state S_0.

For instance, if the initial state distribution is *uniform*:

$$p_{\text{uniform}}(s) \;=\; \frac{1}{|\mathcal{S}|},$$

the resulting optimal policy, π^*_{uniform}, may be different from π^*_p when a different initial distribution is used. For example, consider an initial distribution such as

$$p = [1,\, 0,\, 0,\, \dots] \quad \text{or} \quad p = [0,\, 1,\, 0,\, \dots],$$

which represents a probability of 1 for starting in one specific state and 0 for all others. These variations lead to different optimal policies because the weight or emphasis on each initial state changes.

It is common to denote the optimal policy according to the initial distribution used:

$$\pi_0^* \quad \text{(for a particular initial distribution)}, \qquad \pi_p^* \quad \text{(for the initial distribution } p).$$

If p is modified, the optimization criterion changes accordingly, and the corresponding optimal policy π_p^* may also change.

7 Variance Reduction in Policy Gradient Methods

This set of notes addresses a key issue in policy gradient methods: while the standard estimator of

$$\mathbb{E}\big[q(S,A)\,\nabla \log \pi(A \mid S)\big]$$

is unbiased, it can have high variance. Two common strategies to reduce variance without introducing bias are:

1. Using a **baseline** (or control variate) that is independent of the action.
2. Employing an **actor-critic** approach, which leverages learned value functions.

Below is a more detailed breakdown of the main ideas from the notes:

7.1 High Variance in Policy Gradient Estimators

In policy gradient methods (e.g., REINFORCE), an update often takes the form

$$\nabla J(\theta) \approx \big(G_t\big)\,\nabla \log \pi(A_t \mid S_t;\theta),$$

where G_t is an unbiased sample of the return (or an estimate of $Q^\pi(S_t, A_t)$). Although this estimator is unbiased, it may have large variance, making convergence slow or unstable.

7.2 Baseline as a Control Variate

Motivation. A *control variate* is a technique from statistics used to reduce the variance of an estimator without adding bias. In the context of policy gradients, the most common control variate is a *baseline* $b(S)$ that does not depend on the action A. Because it is independent of the action, subtracting it from $Q^\pi(S,A)$ leaves the expected gradient unbiased:

$$\mathbb{E}\big[(Q^\pi(S,A)-b(S))\,\nabla \log \pi(A \mid S)\big] \;=\; \mathbb{E}\big[Q^\pi(S,A)\,\nabla \log \pi(A \mid S)\big] - \mathbb{E}\big[b(S)\,\nabla \log \pi(A \mid S)\big].$$

Since $\mathbb{E}_{A\sim\pi}[\nabla \log \pi(A \mid S)] = 0$, the second term disappears, thus leaving the estimator unbiased. However, the variance can be significantly reduced by a good choice of $b(S)$.

Example: Advantage Function. One common baseline is $V^\pi(S)$, the state value function. Substituting $b(S) = V^\pi(S)$ yields:

$$Q^\pi(S,A) - V^\pi(S) \;=\; A^\pi(S,A),$$

the *advantage function*. The advantage $A^\pi(S,A)$ indicates how much better or worse an action A is relative to the average value of state S.

7.3 Actor-Critic and Temporal-Difference (TD) Learning

Actor-Critic. An *actor-critic* method combines:

- An *actor* that updates the policy parameters (θ) via gradient ascent.
- A *critic* that estimates value functions (e.g., V^π or Q^π) used as baselines to reduce variance.

Because the critic can learn $V^\pi(S)$ (or $Q^\pi(S,A)$) online using data, it provides an adaptive baseline that improves over time, helping the actor's updates become more stable.

Temporal-Difference (TD) as an Unbiased Estimator. In many actor-critic methods, the critic uses *temporal-difference learning* to update its value function. A TD target for $V^\pi(S)$ might be:

$$V^\pi(S_t) \leftarrow V^\pi(S_t) + \alpha\big(R_{t+1} + \gamma V^\pi(S_{t+1}) - V^\pi(S_t)\big).$$

Although the single-step TD target $\big(R_{t+1} + \gamma V^\pi(S_{t+1})\big)$ can introduce bias in certain settings, under typical assumptions (e.g., continuing tasks with a fixed policy), it converges to the true value function and thus serves as an *unbiased estimator* in the limit. In practice, it also helps reduce variance compared to full Monte Carlo returns.

7.4 Putting It All Together

- **Baseline/Control Variate (BCV)**: Subtracting a baseline that does not depend on A keeps the estimator unbiased and can significantly reduce variance. The most popular baseline is the state value $V^\pi(S)$, leading to the advantage function $A^\pi(S, A)$.
- **Actor-Critic**: Combines the idea of a parameterized policy (actor) with a learned value function (critic). The critic's estimates serve as a baseline, and temporal-difference methods often keep the critic's estimate updated efficiently.
- **Variance-Bias Trade-off**: While Monte Carlo returns provide unbiased estimates of $Q^\pi(S, A)$, they can have high variance. TD methods reduce variance (and sometimes introduce small bias), but typically converge faster in practice. Carefully chosen baselines and TD updates strike a balance between variance and bias, improving overall performance of policy gradient algorithms.

7.5 Using a Parametric Critic to Reduce Variance Further

Beyond the basic idea of subtracting a baseline $b(S)$ (which does not depend on the action A), many policy gradient methods incorporate a *parametric critic* to approximate the value function or the action-value function. Let w be the parameters of the critic, which could approximate:

$$Q_{\text{critic}}(S, A; w) \quad \text{or} \quad V_{\text{critic}}(S; w).$$

This parametric approximation further refines the variance-reduction approach:

1. **Baseline via a Parametric Value Function.** If we choose $b(S) = V_{\text{critic}}(S; w)$ as the baseline, then the policy gradient update becomes

$$\nabla_\theta J(\theta) \approx \big(Q^\pi(S, A) - V_{\text{critic}}(S; w)\big)\, \nabla_\theta \log \pi_\theta(A \mid S),$$

   where $\big(Q^\pi(S, A) - V_{\text{critic}}(S; w)\big)$ is an approximate advantage. In practice, $Q^\pi(S, A)$ itself is often replaced by the critic's estimate $Q_{\text{critic}}(S, A; w)$ or by a sample return. Substituting this yields

$$\big(Q_{\text{critic}}(S, A; w) - V_{\text{critic}}(S; w)\big) \;=\; A_{\text{critic}}(S, A; w),$$

   an *advantage* estimate based on the critic.
2. **Actor-Critic: Single-Sample Updates.** In an *actor-critic* setup, each transition (S, A, R, S') can be used to:
   - Update the critic parameters w (e.g., via temporal-difference learning), aiming to improve the approximation $Q_{\text{critic}}(S, A; w)$ or $V_{\text{critic}}(S; w)$.
   - Update the actor parameters θ, using the critic's estimates to form a low-variance gradient update:

$$\delta_t \;=\; R + \gamma Q_{\text{critic}}(S', A'; w) \;-\; Q_{\text{critic}}(S, A; w)$$

     (for an action-value-based critic), or a similar TD error for a state-value-based critic. This TD error δ_t helps correct both the critic's estimate and provides a local advantage-like signal for the actor.

   Since the critic is updated continually, it progressively provides more accurate estimates of $Q^\pi(S, A)$ or $V^\pi(S)$, which in turn reduces the variance of the actor's policy gradient updates.
3. **Combining Baseline and Actor-Critic.** We often employ *both* a baseline and an actor-critic approach:
   - The **baseline** concept ensures that subtracting something like $V_{\text{critic}}(S; w)$ does not introduce bias but can substantially reduce variance.
   - The **actor-critic** framework ensures that this baseline (or the action-value approximation) is adaptively learned via single-sample updates (e.g., using TD methods).

   Hence, instead of only subtracting a static baseline, we can directly learn and refine $Q_{\text{critic}}(S, A; w)$ (or $V_{\text{critic}}(S; w)$) to generate the advantage term δ. This approach typically converges faster and is more stable than using Monte Carlo returns for the baseline.
4. **Overall Benefit of Parametric Critic.** By maintaining a parameterized function $Q_{\text{critic}}(S, A; w)$ or $V_{\text{critic}}(S; w)$, the method can:
   - Incorporate single-step or multi-step TD updates, which often have lower variance compared to full returns.

- Provide continuous improvement in baseline quality, thus continually reducing variance in the policy gradient.
- Combine seamlessly with neural network approximations, allowing scalability to large state-action spaces.

8 Citations and References