Policy Methods and PPO

Matteo Cozzi

April 15, 2025

A Step Back: What is our Goal in RL?

Reinforcement Learning (RL)

Reinforcement Learning is a framework where an agent learns to make decisions through trial and error, by interacting with an environment and receiving feedback in the form of rewards.

- The agent takes actions in an environment.
- The environment responds with a new situation and a reward signal.
- Over time, the agent aims to learn a strategy (policy) that chooses actions to maximize long-term reward.

Core Objective

Learn a policy that leads to the most rewarding behavior over time.

A Step Back: Value-based vs Policy-based Methods

Value-based Methods

- Learn value function $Q(s, a)$.
- The policy is *indirectly* derived from the learned value function:

$$\pi(s) = \arg \max_{a \in \mathcal{A}(s)} Q(s, a).$$

- Examples: **Q-Learning**, **Deep Q-Network (DQN)**.

Policy-based Methods

- Directly parameterize the policy $\pi(a \mid s; \theta)$ and optimize it with respect to expected returns.
- Examples: **REINFORCE**, **Actor-Critic**, **PPO**.

Why Move to Policy-Based Methods?

- **Direct Optimization:**
  - Policy-based methods directly optimize the decision-making strategy.
  - This often leads to more stable convergence in complex environments.
- **Handling Continuous Actions:**
  - Unlike value-based methods, policy-based methods naturally extend to continuous or high-dimensional action spaces.
  - No need to perform maximization over actions (e.g., $\max_a Q(s, a)$), which may be intractable.
- **Stochastic Policies:**
  - Can represent and learn stochastic behaviors, useful in partially observable or multi-modal settings.
- **Foundation for Advanced Methods:**
  - Forms the basis for powerful algorithms like Actor-Critic and Proximal Policy Optimization (PPO).

From Policy-Based to Gradient-Based Optimization

- **Policy-based methods** directly parameterize a policy $\pi_\theta(a \mid s)$ and optimize a performance measure $J(\theta)$.
- Several optimization approaches exist:
  - **Gradient-based methods:** Optimize using the gradient $\nabla_\theta J(\theta)$. This is the most common approach in deep reinforcement learning.
  - **Gradient-free methods:** Such as evolutionary strategies or other black-box optimizers.
- In modern deep RL, **policy gradient methods** are predominantly used due to their scalability and efficiency with high-dimensional function approximators.

We will focus on the gradient-based derivation that underlies many policy optimization algorithms.

Derivation of the Policy Gradient Theorem (I)

- Define the performance objective:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\big[G(\tau)\big] = \int_\tau P(\tau;\theta)\, G(\tau)\, d\tau,$$

where $\tau = (s_0, a_0, s_1, a_1, \dots)$ denotes a trajectory, and

$$P(\tau;\theta) = \rho_0(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t \mid s_t)\, P(s_{t+1} \mid s_t, a_t).$$

- Our goal is to compute the gradient:

$$\nabla_\theta J(\theta) = \nabla_\theta \int_\tau P(\tau;\theta)\, G(\tau)\, d\tau.$$

Derivation of the Policy Gradient Theorem (II)

- We apply the likelihood ratio trick. Recognize that

$$\nabla_\theta P(\tau;\theta) = P(\tau;\theta)\nabla_\theta \log P(\tau;\theta).$$

- Noting that $P(\tau;\theta)$ factors through the policy, we have:

$$\log P(\tau;\theta) = \log \rho_0(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t \mid s_t) + \sum_{t=0}^{T-1} \log P(s_{t+1} \mid s_t, a_t).$$

- Since the environment dynamics $P(s_{t+1} \mid s_t, a_t)$ and initial state distribution $\rho_0(s_0)$ do not depend on θ, their gradients vanish. Therefore,

$$\nabla_\theta \log P(\tau;\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t).$$

- Plugging this back into our gradient, we obtain:

$$\nabla_\theta J(\theta) = \int_\tau P(\tau;\theta) \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right) G(\tau)\, d\tau.$$

Derivation of the Policy Gradient Theorem (III)

- Express the gradient as an expectation:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G(\tau) \right].$$

- In practice, it is common to use the *return from time t* rather than $G(\tau)$ for each time step:

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k.$$

Then we write:

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

Introduction to REINFORCE

- **REINFORCE** is a Monte Carlo policy gradient algorithm.
- It directly uses complete episodes to estimate returns.
- The update rule is derived from the policy gradient theorem:

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

  where

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k.$$

- In words, we increase the likelihood of actions that yield higher returns.

1. **Sampling a Trajectory:** Execute the current policy π_θ to generate an episode:

$$\tau = \big(s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T\big).$$

   This trajectory is sampled according to $P(\tau; \theta)$, as in the gradient theorem.

2. **Computing the Return:** For each time step t, calculate the return:

$$G_t = \sum_{k=t}^{T-1} \gamma^{\,k-t} r_k.$$

   Link to Theorem: These G_t are the reward signals weighting the log-probability gradients in $\nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t$.

REINFORCE: Gradient Estimation & Policy Update

1. **Gradient Estimation:**
   - For each time step t in the sampled trajectory, compute:
     $$\nabla_\theta \log \pi_\theta(a_t \mid s_t).$$
   - Multiply by the corresponding return G_t to form:
     $$g_t = \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t.$$
   - **Recall:** This directly implements the term from the policy gradient theorem.
2. **Policy Update:** Aggregate the gradients over the episode and update parameters with a learning rate α:
   $$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T-1} g_t.$$
3. **Iteration:** Repeat by collecting new trajectories, ensuring the policy continually improves.

Variance Reduction & Baselines in REINFORCE

- **Problem:** The REINFORCE update

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t$$

  can have high variance, slowing down learning.
- **Solution: Baselines** We subtract a baseline $b(s_t)$ from the return:

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \left(G_t - b(s_t)\right).$$

- **Why it helps:** $\left(G_t - b(s_t)\right)$ represents the *advantage* of taking action a_t in state s_t. Using a baseline reduces the variance of the gradient estimate without introducing bias.
- **Common Baseline:** A natural choice for $b(s_t)$ is the state-value function $V^\pi(s_t)$.

Policy Methods: A Distinction

Actor-Only Methods

- Learn the policy directly from experience.
- Do not estimate any value function.
- Example: **REINFORCE**.
- Simple to implement, but gradients have **high variance**.

Actor-Critic Methods

- Combine a policy (Actor) with a value function (Critic).
- The Critic helps reduce variance by estimating how good actions are.
- Example: **PPO (Proximal Policy optimization)**.
- More stable and sample-efficient than Actor-Only.

Actor-Critic Methods: Detailed Overview

- **Actor:**
  - Parameterizes the policy as $\pi_\theta(a \mid s)$ — it decides which action to take.
  - Updated by ascending the gradient of the expected return.
- **Critic:**
  - Estimates the state-value function $V_w(s)$, which approximates the expected return from state s.
  - Provides feedback to the actor by evaluating the quality of its actions.
- **Interaction:**
  - The Critic's evaluation is used as a *baseline* to compute the **advantage**:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s).$$

  - In practice, a sample-based form is often used:

$$A_t = G_t - V_w(s_t),$$

    where G_t is the return computed from time step t.
- **Benefit:** By using the critic as a baseline, we reduce the variance of the policy gradient estimate.

Deriving the Actor's Gradient (Part I)

- **Performance Objective:**

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[G(\tau)]$$

where τ is a trajectory generated by the policy.

- **Policy Gradient Theorem:**

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

with

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k.$$

- This expression indicates that we adjust θ to increase the probability of actions leading to higher returns.

Deriving the Actor's Gradient (Part II)

- **Introducing a Baseline:** Subtracting a baseline, such as the state-value function $V_w(s_t)$, does not change the expected value but reduces variance:

$$A_t = G_t - V_w(s_t).$$

- **Final Actor Update:** The gradient estimate becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, A_t\right],$$

leading to the parameter update:

$$\theta \leftarrow \theta + \alpha\, \nabla_\theta J(\theta).$$

- This update reinforces actions whose outcomes surpass the expected value.

Critic Update – TD Error and Loss (Part I)

- **Objective for the Critic:** Learn an approximation $V_w(s)$ to the expected return from state s.
- **Temporal Difference (TD) Error:**

$$\delta_t = r_t + \gamma V_w(s_{t+1}) - V_w(s_t).$$

- δ_t measures the discrepancy between the current value estimate and a bootstrap estimate from the next state.

Critic Update – Loss Minimization (Part II)

- **Loss Function:** To train the critic, we minimize the mean squared error (MSE) of the TD error:

$$L(w) = \frac{1}{2}\delta_t^2 = \frac{1}{2}\Big(r_t + \gamma\, V_w(s_{t+1}) - V_w(s_t)\Big)^2.$$

- **Parameter Update:** Update the critic parameters via gradient descent:

$$w \leftarrow w - \alpha_w \nabla_w L(w).$$

- A well-trained critic provides accurate value estimates, leading to reliable advantage computations.

Actor-Critic Process (Part I): Data Collection & Advantage Estimation

1. **Sampling an Episode:** Execute the current policy π_θ to obtain a trajectory:
$$\tau = \{s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T\}.$$
2. **Compute Returns:** For each time step t:
$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k.$$
3. **Estimate Advantage:** Use the critic to compute:
$$A_t = G_t - V_w(s_t).$$
4. This estimation bridges the actor's policy update with the critic's evaluation.

Actor-Critic Process (Part II): Updates and Iteration

1. **Update the Critic:** Minimize the loss:

$$L(w) = \frac{1}{2}\Big(r_t + \gamma\, V_w(s_{t+1}) - V_w(s_t)\Big)^2,$$

   updating w via gradient descent.

2. **Update the Actor:** Adjust θ using:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

3. **Iteration:** Collect new episodes and repeat the process to continually refine both the policy and value estimates.

PPO Motivation (Part I)

- **Instability in Naive Policy Gradients:**
  - Large, unconstrained policy updates can lead to erratic and unstable learning.
  - Sudden shifts in policy often degrade performance in complex environments.
- **Trust Region Policy Optimization (TRPO):**
  - Introduces a KL-divergence constraint:

$$D_{\mathrm{KL}}\left(\pi_\theta, \pi_{\theta_{\text{old}}}\right) \leq \delta.$$

  - Limits policy updates to remain within a "trust region."

PPO Motivation (Part II)

- **Practicality of PPO:**
  - PPO aims to achieve the stability of TRPO without the need for complex second-order optimization.
- **Key Idea:** Use a *clipped* objective to restrict the extent of policy updates.
  - This prevents large deviations from the old policy without explicitly enforcing a KL constraint.
- **Outcome:** More stable updates, improved sample efficiency, and ease of implementation.

From Standard Actor-Critic to PPO

- **Standard Actor-Critic Update:**

$$\theta \leftarrow \theta + \alpha \, \nabla_\theta \log \pi_\theta(a_t \mid s_t) \, \hat{A}_t.$$

- **Limitation:** Direct updates can lead to large changes in the policy if not controlled.
- **Introducing the Probability Ratio:**

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}.$$

- **Interpretation:**
  - $r_t(\theta)$ measures the change in probability for the taken action between the new and old policies.
  - Ideally, $r_t(\theta)$ should remain near 1 to ensure moderate updates.

PPO Clipped Objective

Clipped Surrogate Objective

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

- $\hat{A}_t$ is the advantage estimate.
- The objective takes the minimum of the unclipped and clipped terms to restrict large policy updates.
- **Role of Clipping:**
  - Prevents $r_t(\theta)$ from straying far from 1 (i.e., a small change from the old policy).
  - If $r_t(\theta)$ exceeds the range $[1-\epsilon, 1+\epsilon]$, the corresponding term is clipped.
- The hyperparameter ϵ controls how much the new policy is allowed to deviate from the old policy during each update.
- **Outcome:** The update remains *proximal* to the prior policy, ensuring stable learning.

Actor and Critic Loss in PPO (Part I)

Actor Loss:

$$L_{\text{actor}}(\theta) \approx \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right].$$

- This objective encourages policies that improve advantage estimates while avoiding excessive updates.

Critic Loss:

$$L_{\text{critic}}(w) \approx \mathbb{E}_t\left[\left(V_w(s_t) - G_t^{\text{target}}\right)^2\right],$$

where G_t^{target} is a return estimate (e.g., one-step or n-step bootstrapped return).

Actor and Critic Loss in PPO (Part II)

- **Combined Objective:**

$$L^{\text{PPO}}(\theta, w) = L_{\text{actor}}(\theta) - c_1 L_{\text{critic}}(w) + c_2 H(\pi_\theta),$$

  where:
  - c_1 is the coefficient balancing the critic loss.
  - c_2 is the coefficient for the entropy bonus.
- **Entropy Bonus $H(\pi_\theta)$:**
  - Encourages exploration by penalizing overly deterministic policies.
- **Purpose:** Jointly optimizes the policy and value function, ensuring both accurate value predictions and stable policy improvement.

Generalized Advantage Estimation (GAE): Fundamentals

- **Motivation:**
  - Naively using $G_t - V(s_t)$ yields a high-variance advantage estimate.
  - A reliable advantage estimate is crucial for stable policy updates.
- **Temporal Difference (TD) Residual:**

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t).$$

  - δ_t captures the immediate error of our value prediction.
- **GAE Core Idea:**
  - Use a weighted sum of future TD residuals to compute the advantage.

Generalized Advantage Estimation (GAE): Computation and Benefits

- **GAE Formula:**

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{t+l},$$

  where $\lambda \in [0, 1]$ is a hyperparameter that controls the trade-off:
  - $\lambda = 0$: Uses only the immediate TD error (high bias, low variance).
  - $\lambda = 1$: Approximates the full Monte Carlo return (low bias, high variance).
- **Benefits:**
  - **Variance Reduction:** Bootstrapping from $V(s)$ reduces fluctuations.
  - **Timely Updates:** Utilizes multi-step information without waiting for episode termination.
  - **Tunable Bias-Variance Trade-Off:** λ can be adjusted to meet the needs of the environment.

Full PPO Algorithm: Data Collection & Advantage Estimation

- **Step 1: Data Collection**
  - Run the current policy θ_{old} to collect a batch of trajectories.
  - Gather tuples $(s_t,\ a_t,\ r_t,\ s_{t+1})$ for each time step.
- **Step 2: Compute Returns**
  - For each time step, calculate the return:

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k.$$

  - Returns summarize the future rewards from the current state, but using full returns can introduce high variance.
- **Step 3: Advantage Estimation**
  - Estimate the advantages $\hat{A}_t$ using methods like GAE.

Full PPO Algorithm: Policy & Critic Updates

- **Step 4: Policy Update**
  - Compute the probability ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}.$$

  - Optimize the clipped objective:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\Big(r_t(\theta)\hat{A}_t,\ \text{clip}\big(r_t(\theta),\ 1-\epsilon,\ 1+\epsilon\big)\hat{A}_t\Big)\Big].$$

- **Step 5: Critic Update**
  - Update the critic parameters w by minimizing:

$$L_{\text{critic}}(w) = \mathbb{E}_t\Big[\big(V_w(s_t) - G_t^{\text{target}}\big)^2\Big].$$

  - G_t^{target} can use bootstrapping (e.g., one-step or multi-step returns) to further stabilize learning.
- **Final Step: Iteration**
  - After updating, set $\theta_{\text{old}} \leftarrow \theta$ and repeat for the next data batch.

Benefits of PPO

- **Stable Updates:** Clipping keeps policy changes moderate and avoids catastrophic updates.
- **Sample Efficiency:** Multiple epochs over the same batch improve the efficiency of data usage.
- **Simplicity:** Avoids the complex second-order computations required in TRPO.
- **Wide Applicability:** Effective in continuous control and high-dimensional RL tasks.

Appendix

Additional Material and Derivations

Derivation of the Likelihood Ratio Trick (Part I)

Objective:

$$J(\theta) = \mathbb{E}_{x \sim p_\theta}[f(x)] = \int p_\theta(x) f(x)\, dx.$$

Step 1: Compute the Gradient of $J(\theta)$

$$\nabla_\theta J(\theta) = \nabla_\theta \int p_\theta(x) f(x)\, dx = \int \nabla_\theta p_\theta(x) f(x)\, dx.$$

Step 2: Log-Derivative Identity

$$\nabla_\theta \log p_\theta(x) = \frac{1}{p_\theta(x)} \nabla_\theta p_\theta(x) \quad \Rightarrow \quad \nabla_\theta p_\theta(x) = p_\theta(x)\, \nabla_\theta \log p_\theta(x).$$

Derivation of the Likelihood Ratio Trick (Part II)

Step 3: Substitute Back into the Gradient Expression

$$\nabla_\theta J(\theta) = \int p_\theta(x) \, \nabla_\theta \log p_\theta(x) \, f(x) \, dx.$$

Step 4: Express as an Expectation

$$\nabla_\theta J(\theta) = \mathbb{E}_{x \sim p_\theta} \left[f(x) \, \nabla_\theta \log p_\theta(x) \right].$$

Intuition: We rewrite the derivative of the probability $p_\theta(x)$ as a product of $p_\theta(x)$ and the gradient of its log — allowing us to express the full gradient as an expectation, which can be estimated from samples.