

PPO: A Complete Derivation and Its Design Motivations

Including How Actors and Critics Learn: From Policy Gradients, PPO-Clip, and GAE to Robot Control

October 18, 2026

Contents

Part I. From Expected Return to Policy Gradients	2
Part II. Baselines, Value Functions, and Advantage	4
Part III. Performance Differences, Surrogate Objectives, and Importance Sampling	5
Part IV. TRPO: Controlling Policy Updates with a Trust Region	6
Part V. Deriving the PPO-Clipped Objective	8
Part VI. GAE: How Do We Estimate the Advantage?	10
Part VII. The Complete Actor–Critic Loss	13
Part VIII. Continuous-Action PPO and Robot Control	16
Part IX. A Complete PPO Iteration	18
Part X. Hyperparameters and Diagnostics	20
Part XI. Connections to REINFORCE, Actor–Critic, and TRPO	21
Part XII. Final Formula Sheet and Research Takeaways	22

[Course home](#) · [简体中文](#) · [PDF](#)

Item	Details
Topic	PPO
Revision	2026-10-19
License	CC BY-NC-SA 4.0 · Xiong Zeng and contributors

Part I. From Expected Return to Policy Gradients

Learning objective. Understand not just the PPO loss, but why every component was introduced. This tutorial develops the mathematics step by step, provides numerical examples, addresses implementation pitfalls, and connects PPO to continuous robot control and residual reinforcement learning.

Central idea: Increase the probability of advantageous actions and decrease the probability of disadvantageous actions, without changing the policy so aggressively that an improvement estimate constructed from old-policy data becomes unreliable.

Roadmap: Expected return $\rightarrow$ Policy gradient $\rightarrow$ Baselines and advantage $\rightarrow$ Performance differences and importance sampling $\rightarrow$ TRPO $\rightarrow$ PPO-Clip $\rightarrow$ GAE $\rightarrow$ Complete actor-critic objective $\rightarrow$ Continuous control and implementation.

Notation. We mainly consider a discounted infinite-horizon MDP with $\gamma \in [0, 1]$, and $J(\theta)$ is the discounted return from the initial-state distribution. We explicitly distinguish the rigorous discounted-state-occupancy formulation from the time-step averages commonly used in PPO implementations. Within each PPO optimization phase, the advantage estimates $\hat{A}_t$ are held fixed after being computed from an old-policy rollout.

Learning thread: The critic learns to predict long-term return from fixed bootstrapped/GAE targets $\rightarrow$ TD errors and GAE estimate the relative quality of old actions $\rightarrow$ the actor changes action probabilities through advantage-weighted policy gradients $\rightarrow$ PPO-Clip discourages overly large policy changes $\rightarrow$ the updated policy collects new samples. **The critic performs regression; the actor optimizes a clipped surrogate over action distributions. Fixed advantage estimates connect them.**

1. Formulating the reinforcement-learning objective

Consider a Markov decision process (MDP)

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho_0),$$

where $s_t \in \mathcal{S}$ denotes a state, $a_t \in \mathcal{A}$ an action, $P(s_{t+1} \mid s_t, a_t)$ the environment transition kernel, r_t the reward, ρ_0 the initial-state distribution, and $\pi_\theta(a \mid s)$ a stochastic policy parameterized by θ . The goal is

$$\boxed{\theta^* = \arg \max_{\theta} J(\theta)}, \quad \boxed{J(\theta) = \mathbb{E}_{\tau \sim p_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]}. \quad (1)$$

A trajectory is $\tau = (s_0, a_0, s_1, a_1, \dots)$, with discounted return $R(\tau) = \sum_t \gamma^t r_t$. Consequently,

$$J(\theta) = \int p_\theta(\tau) R(\tau) d\tau.$$

Motivation and difficulty. Policy parameters influence actions, which affect future states and subsequent rewards:

$$\theta \longrightarrow a_0 \longrightarrow s_1 \longrightarrow a_1 \longrightarrow \dots \longrightarrow R(\tau).$$

With contact, collision, friction, or unknown dynamics, differentiating the return through the physical environment may be unavailable or impractical. We need a gradient estimator that **does not require differentiable dynamics**.

2. Deriving policy gradients using the log-derivative trick

Differentiate the expected return:

$$\nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) R(\tau) d\tau.$$

Apply the score-function identity

$$\boxed{\nabla_{\theta} p_{\theta}(\tau) = p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau)}. \quad (2)$$

We obtain

$$\boxed{\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}} [R(\tau) \nabla_{\theta} \log p_{\theta}(\tau)]}. \quad (3)$$

This is the **likelihood-ratio (score-function) trick**. It converts the gradient of a probability density into a derivative of its log density. The second key observation is that an MDP trajectory probability factors:

$$p_{\theta}(\tau) = \rho_0(s_0) \prod_{t=0}^{\infty} \pi_{\theta}(a_t | s_t) P(s_{t+1} | s_t, a_t).$$

Taking logarithms, and assuming neither the initial-state distribution nor the transition kernel depends on θ ,

$$\begin{aligned} \log p_{\theta}(\tau) &= \log \rho_0(s_0) + \sum_t \log \pi_{\theta}(a_t | s_t) + \sum_t \log P(s_{t+1} | s_t, a_t), \\ \boxed{\nabla_{\theta} \log p_{\theta}(\tau)} &= \boxed{\sum_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)}. \end{aligned} \quad (4)$$

Thus, formally,

$$\boxed{\nabla_{\theta} J(\theta) = \mathbb{E} \left[R(\tau) \sum_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]}. \quad (5)$$

A rigorous infinite-horizon derivation requires the usual integrability and interchange-of-limits assumptions; alternatively, derive the result in finite horizon and take a suitable limit.

What did this accomplish? We differentiate only the policy network's $\log \pi_{\theta}(a_t | s_t)$. The environment is used in the forward direction to generate trajectories and rewards, without requiring its derivatives.

3. Causality and reward-to-go

Equation (5) multiplies every action's score gradient by the entire trajectory return. But the action a_t cannot change rewards that occurred before t . The zero-mean score identity makes the expected contribution of those past-reward cross terms vanish. Define reward-to-go

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

Then

$$\boxed{\nabla_{\theta} J(\theta) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) G_t \right]}. \quad (6)$$

Motivation. Credit an action for outcomes that it can actually influence. Discarding past rewards often reduces variance while preserving the expected gradient. The explicit γ^t is needed for the return objective in Eq. (1); implementations commonly average sampled time steps without this explicit factor, which amounts to a particular sampling/weighting convention rather than an identical expression for every infinite-horizon formulation.

Part II. Baselines, Value Functions, and Advantage

4. Why can we subtract a baseline?

Reward-to-go still fluctuates because future outcomes are random. Introduce a state-dependent baseline $b(s_t)$:

$$g_t = \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) [G_t - b(s_t)].$$

For a fixed state, the additional term has zero expectation:

$$\begin{aligned} & \mathbb{E}_{a \sim \pi_{\theta}(\cdot | s)} [\nabla_{\theta} \log \pi_{\theta}(a | s) b(s)] \\ &= b(s) \int \pi_{\theta}(a | s) \nabla_{\theta} \log \pi_{\theta}(a | s) da \\ &= b(s) \int \nabla_{\theta} \pi_{\theta}(a | s) da \\ &= b(s) \nabla_{\theta} \int \pi_{\theta}(a | s) da \\ &= b(s) \nabla_{\theta} 1 = 0. \end{aligned} \tag{7}$$

Therefore,

$$\boxed{\mathbb{E}[\nabla \log \pi_{\theta} G_t] = \mathbb{E}[\nabla \log \pi_{\theta} (G_t - b(s_t))]} \tag{8}$$

Motivation. Remove a common, action-independent component of future return, leaving a more informative relative-action signal. Not every baseline necessarily reduces variance; the minimum-variance baseline can depend on the squared norm of the score gradient. Using $V^{\pi}(s)$ is a natural and highly practical choice.

5. Why use the value function as the baseline?

Define the state-value function, action-value function, and advantage function:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}[G_t | s_t = s], \\ Q^{\pi}(s, a) &= \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a], \\ \boxed{A^{\pi}(s, a)} &= \boxed{Q^{\pi}(s, a) - V^{\pi}(s)}. \end{aligned} \tag{9}$$

$V^{\pi}(s)$ measures expected return when continuing with the current policy. $Q^{\pi}(s, a)$ measures expected return when first taking action a , and then following the policy. Hence:

- $A^{\pi}(s, a) > 0$: action a performs better than the policy's average action at s ; it should become more likely.
- $A^{\pi}(s, a) < 0$: the action is worse than average; it should become less likely.
- $A^{\pi}(s, a) = 0$: this estimate supplies no relative preference signal.

Using the exact value baseline yields the advantage policy-gradient formula:

$$\boxed{\nabla_{\theta} J(\theta) = \mathbb{E} \left[\sum_t \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) A^{\pi_{\theta}}(s_t, a_t) \right]} \tag{10}$$

This motivates the **actor-critic architecture**: the actor π_{θ} generates actions; the critic V_{ϕ} approximates state values; the advantage connects the two. An imperfect state-only baseline does not itself change the ideal score-function expectation, but bootstrapped advantage estimators constructed from an imperfect critic can be biased.

Part III. Performance Differences, Surrogate Objectives, and Importance Sampling

6. Why can't we reuse old data indefinitely?

The policy-gradient theorem describes expectations under the current policy. After data is collected with π_{θ_0} and the policy changes to π_{θ_1} or π_{θ_2} , the same rollout is still distributed according to π_{θ_0} , not the new policy. Recollecting trajectories after every optimizer step, however, is expensive.

One goal of PPO is to **perform several relatively safe first-order optimization steps on one batch of old-policy data**. Achieving this requires separating changes in the state distribution from changes in action probabilities at a given state.

7. Step-by-step proof of the performance difference lemma

Define the normalized discounted state-visitation distribution

$$d_\pi(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t \Pr(s_t = s \mid \pi). \quad (11)$$

For continuous state spaces, this notation refers to the corresponding density or probability measure. For any new and old policies, the **performance difference lemma** gives the exact identity

$$J(\pi_\theta) - J(\pi_{\text{old}}) = \frac{1}{1 - \gamma} \mathbb{E}_{s \sim d_{\pi_\theta}} [A^{\pi_{\text{old}}}(s, a)]. \quad (12)$$

Proof. The Bellman equation for the old policy implies

$$A^{\pi_{\text{old}}}(s_t, a_t) = \mathbb{E}[r_t + \gamma V^{\pi_{\text{old}}}(s_{t+1}) - V^{\pi_{\text{old}}}(s_t) \mid s_t, a_t].$$

Sum the discounted advantages along a trajectory generated by the **new** policy:

$$\begin{aligned} & \mathbb{E}_{\pi_\theta} \left[\sum_t \gamma^t A^{\pi_{\text{old}}}(s_t, a_t) \right] \\ &= \mathbb{E}_{\pi_\theta} \left[\sum_t \gamma^t r_t + \sum_t \gamma^{t+1} V^{\pi_{\text{old}}}(s_{t+1}) - \sum_t \gamma^t V^{\pi_{\text{old}}}(s_t) \right]. \end{aligned}$$

The last two sums telescope, leaving $-V^{\pi_{\text{old}}}(s_0)$ when the limiting tail vanishes. Thus

$$\mathbb{E}_{\pi_\theta} \left[\sum_t \gamma^t A^{\pi_{\text{old}}}(s_t, a_t) \right] = J(\pi_\theta) - J(\pi_{\text{old}}).$$

Re-expressing the discounted sum using the occupancy distribution gives Eq. (12).

Interpretation. A new policy improves performance when, in the states it *actually visits* and with the actions it *actually chooses*, it achieves positive advantage relative to the old policy.

8. Approximating true improvement by a surrogate objective

The challenge in Eq. (12) is the new-policy state distribution:

$$s \sim d_{\pi_\theta}.$$

We only possess trajectories from the old policy. If the policies remain close, we can make the **local approximation**

$$d_{\pi_\theta}(s) \approx d_{\pi_{\text{old}}}(s).$$

Ignoring the constant factor $1/(1 - \gamma)$, define

$$L_{\text{sur}}(\theta) = \mathbb{E}_{\substack{s \sim d_{\pi_{\text{old}}} \\ a \sim \pi_\theta}} [A^{\pi_{\text{old}}}(s, a)]. \quad (13)$$

We have frozen the state distribution and need only reason about changes in action probabilities conditional on state. With exact advantages, this surrogate has the correct local first-order direction at the old policy (up to the discarded positive constant).

Crucial condition. The approximation is credible only while π_θ remains reasonably close to π_{old} . This is precisely the motivation behind trust regions and clipping.

9. Deriving the probability ratio by importance sampling

Equation (13) still takes actions from the new policy, whereas our stored actions were sampled by the old policy. Conditional on state s ,

$$\begin{aligned} \mathbb{E}_{a \sim \pi_\theta} [A^{\pi_{\text{old}}}(s, a)] &= \int \pi_\theta(a | s) A^{\pi_{\text{old}}}(s, a) da \\ &= \int \pi_{\text{old}}(a | s) \frac{\pi_\theta(a | s)}{\pi_{\text{old}}(a | s)} A^{\pi_{\text{old}}}(s, a) da. \end{aligned}$$

Define the probability ratio

$$r_\theta(s, a) = \frac{\pi_\theta(a | s)}{\pi_{\text{old}}(a | s)}. \quad (14)$$

We arrive at a conservative-policy-iteration (CPI) surrogate:

$$L^{\text{CPI}}(\theta) = \mathbb{E}_{\substack{s \sim d_{\pi_{\text{old}}} \\ a \sim \pi_{\text{old}}}} [r_\theta(s, a) A^{\pi_{\text{old}}}(s, a)]. \quad (15)$$

In practice this is written $\mathbb{E}_t[r_t(\theta) \hat{A}_t]$.

For example, if the old policy assigns probability 0.2 to an action and the new policy assigns 0.3, then $r = 1.5$; a new probability of 0.1 gives $r = 0.5$. Positive advantage encourages a larger ratio; negative advantage encourages a smaller one.

Limits. This importance-sampling identity requires appropriate support / absolute continuity: an action receiving positive probability under the new policy cannot have zero probability under the old policy. Standard nondegenerate Gaussians typically satisfy this support condition. Importantly, the ratio corrects the **conditional action distribution**, not the discrepancy between d_{π_θ} and $d_{\pi_{\text{old}}}$. It therefore does not eliminate the state-distribution approximation in Eq. (13).

Part IV. TRPO: Controlling Policy Updates with a Trust Region

10. Why is unconstrained maximization of CPI dangerous?

For an action with positive advantage, $r_t \hat{A}_t$ increases as r_t grows. With $\hat{A} = 5$, ratios 1.2, 2, 10 yield contributions 6, 10, 50. It seems attractive to drive the probability of such an action arbitrarily high. But its advantage was estimated under the old policy, and may no longer predict performance after substantial policy changes.

A robot's changed action distribution can trigger a different contact mode, drop an object, or induce states almost absent from the old dataset. Hence increasing L^{CPI} does **not** guarantee an increase in actual return J .

11. KL divergence and the TRPO constraint

Measure changes between the action distributions at a fixed state using KL divergence:

$$D_{\text{KL}}(\pi_{\text{old}} \parallel \pi_{\theta}) = \mathbb{E}_{a \sim \pi_{\text{old}}} \left[\log \frac{\pi_{\text{old}}(a \mid s)}{\pi_{\theta}(a \mid s)} \right].$$

The practical TRPO optimization problem is

$$\begin{array}{ll} \max_{\theta} & \mathbb{E}_t[r_t(\theta) \hat{A}_t] \\ \text{s.t.} & \mathbb{E}_{s \sim d_{\pi_{\text{old}}}} [D_{\text{KL}}(\pi_{\text{old}}(\cdot \mid s) \parallel \pi_{\theta}(\cdot \mid s))] \leq \delta. \end{array} \quad (16)$$

The parameter δ controls the trust-region size. The motivation is not “always make the smallest change”; rather, **seek improvements inside a local region where the performance surrogate is comparatively reliable**.

The TRPO analysis yields performance bounds that penalize the worst-state policy discrepancy; a representative structural form is

$$J(\pi_{\theta}) \geq J(\pi_{\text{old}}) + \frac{L_{\text{sur}}(\theta)}{1 - \gamma} - C \max_s D_{\text{KL}}(\pi_{\text{old}} \parallel \pi_{\theta}). \quad (17)$$

Here C depends on bounds on advantages / returns and on γ ; precise constants and KL orientation depend on the chosen inequality. The theoretical control of a **worst-state** discrepancy is stronger than the **average KL** constraint commonly implemented. Actual neural-network TRPO therefore has no unconditional monotonic-improvement guarantee.

12. TRPO's local second-order approximation and natural gradient

Write $\Delta\theta = \theta - \theta_{\text{old}}$. Taylor-expand the surrogate:

$$L^{\text{CPI}}(\theta) \approx L^{\text{CPI}}(\theta_{\text{old}}) + g^{\top} \Delta\theta, \quad g = \nabla_{\theta} L^{\text{CPI}}|_{\theta_{\text{old}}}.$$

KL divergence is zero, and has zero first derivative, at identical distributions. Its second-order expansion is

$$\bar{D}_{\text{KL}} \approx \frac{1}{2} \Delta\theta^{\top} F \Delta\theta,$$

where F is the Fisher information matrix evaluated at the old parameters. To make the evaluation point unambiguous, define $u_{\text{old}}(s, a) = \nabla_{\theta} \log \pi_{\theta}(a \mid s)|_{\theta=\theta_{\text{old}}}$. Then

$$F = \mathbb{E}_{s \sim d_{\pi_{\text{old}}}, a \sim \pi_{\text{old}}} [u_{\text{old}}(s, a) u_{\text{old}}(s, a)^{\top}].$$

The local approximation is

$$\begin{array}{ll} \max_{\Delta\theta} & g^{\top} \Delta\theta \\ \text{s.t.} & \frac{1}{2} \Delta\theta^{\top} F \Delta\theta \leq \delta. \end{array} \quad (18)$$

Its Lagrangian is

$$\mathcal{L}(\Delta\theta, \eta) = g^{\top} \Delta\theta - \eta \left(\frac{1}{2} \Delta\theta^{\top} F \Delta\theta - \delta \right).$$

Stationarity $g - \eta F \Delta\theta = 0$ yields $\Delta\theta = \eta^{-1} F^{-1} g$. Enforcing the active constraint, assuming F positive definite and $g \neq 0$, gives

$$\Delta\theta^* = \sqrt{\frac{2\delta}{g^\top F^{-1} g}} F^{-1} g. \quad (19)$$

The direction $F^{-1}g$ is the hallmark of **natural policy gradient**. A large policy network cannot directly invert its Fisher matrix, so TRPO relies on Fisher-vector products, conjugate gradients, line search, and often damping. **PPO's design motivation** is to retain conservative policy improvement while dispensing with this complicated second-order constrained optimization.

Part V. Deriving the PPO-Clipped Objective

13. From a KL penalty to a clipped surrogate

A straightforward first-order alternative is PPO-Penalty:

$$L^{\text{KL PEN}}(\theta) = \mathbb{E}_t[r_t(\theta)\hat{A}_t] - \beta \mathbb{E}_t[D_{\text{KL}}(\pi_{\text{old}}\|\pi_\theta)]. \quad (20)$$

A small β may under-regularize updates, while a large β can make learning unnecessarily slow; adaptive tuning is commonly needed. PPO-Clip instead restricts the **additional surrogate gain** obtained from overly large changes in action probability. Define

$$\text{clip}(r, 1 - \epsilon, 1 + \epsilon) = \min\{1 + \epsilon, \max\{r, 1 - \epsilon\}\}.$$

The familiar PPO objective is

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t \right) \right]. \quad (21)$$

Values around $\epsilon = 0.1$ – 0.2 are common starting points, not universal prescriptions. The key is to understand **why the min and clip operations must be used together**.

14. Positive advantage: stop rewarding overly large probability increases

If $\hat{A}_t > 0$, we want the probability of this action to increase, hence $r_t > 1$ is beneficial. Equation (21) reduces to

$$L_t^{\text{CLIP}} = \min(r_t, 1 + \epsilon)\hat{A}_t = \begin{cases} r_t\hat{A}_t, & r_t \leq 1 + \epsilon, \\ (1 + \epsilon)\hat{A}_t, & r_t > 1 + \epsilon. \end{cases} \quad (22)$$

Set $\hat{A}_t = 2$ and $\epsilon = 0.2$:

Ratio r_t	Unclipped $r_t\hat{A}_t$	PPO contribution
0.8	1.6	1.6
1.0	2.0	2.0
1.1	2.2	2.2
1.2	2.4	2.4
1.5	3.0	2.4
2.0	4.0	2.4

Once $r_t > 1.2$, that sample no longer rewards further increases in its probability; its derivative in that direction is zero.

Motivation: Avoid excessively reinforcing a sample that appears advantageous under the old policy.

15. Negative advantage: stop rewarding overly large probability decreases

For $\hat{A}_t < 0$, reducing the action probability is beneficial. Multiplying by a negative advantage reverses the min/max relationship:

$$L_t^{\text{CLIP}} = \max(r_t, 1 - \epsilon) \hat{A}_t = \begin{cases} (1 - \epsilon) \hat{A}_t, & r_t < 1 - \epsilon, \\ r_t \hat{A}_t, & r_t \geq 1 - \epsilon. \end{cases} \quad (23)$$

With $\hat{A}_t = -2$ and $\epsilon = 0.2$:

Ratio r_t	Unclipped $r_t \hat{A}_t$	PPO contribution
0.5	-1.0	-1.6
0.7	-1.4	-1.6
0.8	-1.6	-1.6
1.0	-2.0	-2.0
1.2	-2.4	-2.4
1.5	-3.0	-3.0

PPO maximizes this objective, so reducing r_t is helpful for a negative-advantage sample. Once $r_t < 0.8$, however, further probability decreases produce no additional surrogate reward.

16. Why do we need **min** instead of clipping alone?

Suppose we optimized only $\text{clip}(r, 1 - \epsilon, 1 + \epsilon) \hat{A}$. If $\hat{A} > 0$, $r = 0.5$, and $\epsilon = 0.2$, then a good action has already been made much less likely—an **unfavorable** change. Clipping alone returns $0.8\hat{A}$ and gives zero derivative everywhere below $r = 0.8$, providing no correction signal.

With **min**, the selected term is instead $r\hat{A}$, because $r\hat{A} < 0.8\hat{A}$. The positive derivative persists and pushes the action probability back up. The symmetric argument applies to unfavorable probability increases for negative-advantage actions.

Direction beneficial but change excessive:	stop extra surrogate reward;
Direction unfavorable for the advantage:	retain corrective gradient.

Since $\min(x, y) \leq x$, the sample-wise bound $L^{\text{CLIP}} \leq L^{\text{CPI}}$ holds. This is a pessimistic **lower bound on the unclipped surrogate, not a guaranteed lower bound on the true return J** .

17. Gradient-level interpretation of clipping

Because the denominator in the probability ratio is fixed while optimizing the current rollout,

$$\nabla_{\theta} r_t(\theta) = r_t(\theta) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t). \quad (24)$$

Away from nonsmooth boundaries, the single-sample gradient is

$$\nabla_{\theta} L_t^{\text{CLIP}} = \begin{cases} 0, & \hat{A}_t > 0, r_t > 1 + \epsilon, \\ 0, & \hat{A}_t < 0, r_t < 1 - \epsilon, \\ \hat{A}_t r_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t), & \text{otherwise.} \end{cases} \quad (25)$$

At $\theta = \theta_{\text{old}}$, all ratios equal one. The initial local gradient therefore agrees with CPI / advantage policy gradient (up to the chosen normalization). **PPO does not change the initial improvement direction; it moderates the gain from further changes as repeated updates move the probability ratios away from one.**

18. PPO does not enforce a hard ratio or KL constraint

Choosing $\epsilon = 0.2$ does **not** ensure $0.8 \leq r_t \leq 1.2$ after an optimizer step. Clipping changes an objective, not the feasible parameter set. Parameters are shared across states and samples, and several mini-batch epochs can still drive the average KL too high.

Practical PPO systems therefore monitor

$$\bar{D}_{\text{KL}}(\pi_{\text{old}} \| \pi_{\theta})$$

and often use target-KL early stopping, learning-rate adaptation, and/or gradient clipping. Distinguish three separate operations: **ratio clipping** limits surrogate incentives, **gradient clipping** limits gradient norms, and **action clipping** limits executed command values.

Part VI. GAE: How Do We Estimate the Advantage?

19. Monte Carlo advantage: motivation and cost

The $\hat{A}_t$ used by PPO is not directly observed. By definition $A^{\pi} = Q^{\pi} - V^{\pi}$. The simplest estimate substitutes the sampled reward-to-go for the action value:

$$\boxed{\hat{A}_t^{\text{MC}} = G_t - V_{\phi}(s_t)}, \quad G_t = \sum_{l=0}^{T-t-1} \gamma^l r_{t+l}. \quad (26)$$

For a properly completed episode, the conditional expectation of the sampled return G_t equals $Q^{\pi}(s_t, a_t)$. No value-function bootstrap of distant future outcomes is needed. However, imagine a grasping task where success is rewarded only at time 100. Whether an action at time 10 receives a large Monte Carlo return depends on many subsequent random events; the gradient estimate may therefore have high variance.

Question: Could we estimate advantage with shorter segments of experience instead of depending on the entire future trajectory?

20. One-step TD advantage from the Bellman equation

The Bellman identity reads

$$Q^{\pi}(s_t, a_t) = \mathbb{E}[r_t + \gamma V^{\pi}(s_{t+1}) | s_t, a_t].$$

Consequently, the one-step temporal-difference (TD) residual

$$\boxed{\delta_t = r_t + \gamma V_{\phi}(s_{t+1}) - V_{\phi}(s_t)} \quad (27)$$

is a candidate advantage estimator:

$$\boxed{\hat{A}_t^{(1)} = \delta_t}. \quad (28)$$

It needs just one transition s_t, a_t, r_t, s_{t+1} . Replacing random distant rewards by $V_\phi(s_{t+1})$ often reduces variance, but introduces bootstrap bias if the critic is inaccurate. At a true terminal state the future value is zero; at an artificial time cutoff it is usually appropriate to bootstrap.

Estimator	Main benefit	Main limitation
Monte Carlo	Less bias from bootstrapping future values	High variance from uncertain long-term returns
One-step TD	Lower variance and immediate feedback	Greater dependence on critic accuracy

21. Deriving n-step advantage estimates

The one-step estimate is

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

The two-step estimate is

$$\begin{aligned} \hat{A}_t^{(2)} &= r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t) \\ &= [r_t + \gamma V(s_{t+1}) - V(s_t)] \\ &\quad + \gamma [r_{t+1} + \gamma V(s_{t+2}) - V(s_{t+1})] \\ &= \boxed{\delta_t + \gamma \delta_{t+1}}. \end{aligned} \quad (29)$$

Similarly, the three-step estimate is

$$\hat{A}_t^{(3)} = \delta_t + \gamma \delta_{t+1} + \gamma^2 \delta_{t+2}.$$

For a general horizon n ,

$$\boxed{\hat{A}_t^{(n)} = \sum_{l=0}^{n-1} \gamma^l \delta_{t+l}}. \quad (30)$$

Small n emphasizes value bootstrapping; large n uses more observed future rewards. **GAE's motivation:** Rather than choosing one n , smoothly aggregate many n-step estimators.

22. Exponential weighting and the GAE derivation

Let $0 \leq \lambda < 1$. Define

$$\boxed{\hat{A}_t^{\text{GAE}(\gamma, \lambda)} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \hat{A}_t^{(n)}}. \quad (31)$$

The weights sum to one: $(1 - \lambda) \sum_{n \geq 1} \lambda^{n-1} = 1$. Substitute the n-step expansion and exchange the summation order:

$$\begin{aligned}
\hat{A}_t^{\text{GAE}} &= (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{l=0}^{n-1} \gamma^l \delta_{t+l} \\
&= \sum_{l=0}^{\infty} \gamma^l \delta_{t+l} (1 - \lambda) \sum_{n=l+1}^{\infty} \lambda^{n-1} \\
&= \sum_{l=0}^{\infty} \gamma^l \delta_{t+l} \lambda^l.
\end{aligned}$$

Therefore,

$$\boxed{\hat{A}_t^{\text{GAE}(\gamma, \lambda)} = \sum_{l=0}^{\infty} (\gamma \lambda)^l \delta_{t+l}}. \quad (32)$$

Equivalently,

$$\hat{A}_t = \delta_t + \gamma \lambda \delta_{t+1} + (\gamma \lambda)^2 \delta_{t+2} + \dots$$

Two endpoints. With $\lambda = 0$, GAE becomes one-step TD: $\hat{A}_t = \delta_t$. As $\lambda \rightarrow 1$, GAE incorporates all later TD residuals; for a complete terminating episode, setting $\lambda = 1$ gives $G_t - V(s_t)$. Common initial choices are $\gamma = 0.99$ and $\lambda = 0.95$.

GAE does not mathematically guarantee lower variance in every problem. It introduces a tunable **bias–variance tradeoff**: smaller λ generally relies more on the critic, while larger λ generally relies more on sampled distant rewards.

23. The $O(T)$ backward recursion for GAE

Equation (32) implies

$$\boxed{\hat{A}_t = \delta_t + \gamma \lambda \hat{A}_{t+1}}. \quad (33)$$

Let $d_t = 1$ indicate a genuine MDP terminal transition. Then

$$\begin{aligned}
\delta_t &= r_t + \gamma(1 - d_t)V_\phi(s_{t+1}) - V_\phi(s_t), \\
\hat{A}_t &= \delta_t + \gamma \lambda (1 - d_t) \hat{A}_{t+1}.
\end{aligned} \quad (34)$$

A single backward pass from $T - 1$ computes all advantages in $O(T)$ time. If a fixed-length rollout stops without the MDP actually terminating, bootstrap from $V(s_T)$. But a recursion must not accidentally propagate the next episode's advantage across a reset. Implementations must distinguish **terminated** (genuine terminal), **truncated** (external time limit), and **rollout boundary** (a sampling segment boundary).

24. Constructing critic targets from GAE

GAE estimates

$$\hat{A}_t \approx Q^{\pi_{\text{old}}}(s_t, a_t) - V_{\text{old}}(s_t).$$

Define the TD(λ) return target by

$$\boxed{\hat{R}_t^\lambda = \hat{A}_t^{\text{GAE}} + V_{\phi_{\text{old}}}(s_t)}. \quad (35)$$

$V_{\phi_{\text{old}}}$ denotes the critic values saved when targets for the rollout were calculated. Hold these values fixed during the PPO epochs. The construction aligns the actor's advantage estimates with the critic's regression target.

24.1 Where do critic labels come from? Bellman targets and semi-gradients

The critic does not know the true $V^{\pi_{\text{old}}}(s)$. Instead, samples (s_t, a_t, r_t, s_{t+1}) collected under the behavior policy provide approximate regression targets. For a nonterminal transition, the one-step target and temporal-difference error are

$$\boxed{y_t^{\text{TD}} = r_t + \gamma V_{\phi_{\text{old}}}(s_{t+1})}, \quad \delta_t = y_t^{\text{TD}} - V_{\phi_{\text{old}}}(s_t).$$

If $d_t = 1$ denotes true MDP termination, use $y_t^{\text{TD}} = r_t + \gamma(1 - d_t)V_{\phi_{\text{old}}}(s_{t+1})$. A time limit or rollout boundary is not necessarily a true terminal transition: bootstrap from the appropriate *final observation*, not a reset observation.

Hold the target constant while fitting the critic:

$$\ell_t(\phi) = \frac{1}{2} (V_{\phi}(s_t) - \text{stopgrad}(y_t^{\text{TD}}))^2,$$

$$\boxed{\nabla_{\phi} \ell_t = (V_{\phi}(s_t) - y_t^{\text{TD}}) \nabla_{\phi} V_{\phi}(s_t)}.$$

This is a **semi-gradient TD update**: gradients flow through the current prediction, but not through the bootstrapped target. Backpropagating through the target would change the objective being optimized. Standard PPO does **not require** a separate slowly updated target network of the kind commonly used in DQN. The old critic snapshot and fixed per-rollout regression targets generally suffice.

24.2 Why TD(λ) returns rather than only one-step targets?

GAE, derived in Sections 22–23, supplies a return target consistent with the actor's advantage estimate:

$$\boxed{\hat{R}_t^{\lambda} = V_{\text{old}}(s_t) + \hat{A}_t^{\text{GAE}}}.$$

On a nonterminal portion of a trajectory, this is equivalent to

$$\hat{R}_t^{\lambda} = r_t + \gamma \left[(1 - \lambda) V_{\text{old}}(s_{t+1}) + \lambda \hat{R}_{t+1}^{\lambda} \right].$$

At a rollout boundary, initialize the recursion using the appropriate bootstrap value $V_{\text{old}}(s_T)$; at true termination, future return is zero. With $\lambda = 0$, the target is one-step TD. With $\lambda = 1$ on a complete terminal trajectory, it is the Monte Carlo return. Intermediate values mix multi-step reward information with bootstrapping.

The two critic snapshots must not be conflated. V_{old} generates targets, while V_{ϕ} is being trained. During all minibatch epochs of one PPO update, hold $\hat{A}_t$, $\hat{R}_t^{\lambda}$, and the old action log probabilities fixed, even after the critic parameters change.

Part VII. The Complete Actor–Critic Loss

25. Actor loss

First calculate and freeze $\hat{A}_t$ for the old-policy rollout; advantage normalization is also common. The actor maximizes

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

For gradient-descent software such as PyTorch, minimize

$$\boxed{\mathcal{L}_{\text{actor}} = -L^{\text{CLIP}}}. \quad (36)$$

If the advantage estimates were produced by a differentiable critic currently inside the computational graph, detach them from the actor objective to avoid unintentionally backpropagating through the advantage targets.

25.1 How can the actor learn without labeled optimal actions?

Behavior cloning needs expert labels a_t^* ; PPO does not. Its own sampled actions are weighted by their **relative quality**, $\hat{A}_t$. Begin with the unclipped surrogate,

$$L_t^{\text{CPI}}(\theta) = \rho_t(\theta) \hat{A}_t, \quad \rho_t(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\text{old}}(a_t | s_t)}.$$

With the old denominator and advantage held fixed,

$$\boxed{\nabla_\theta [-L_t^{\text{CPI}}] = -\hat{A}_t \rho_t(\theta) \nabla_\theta \log \pi_\theta(a_t | s_t)}.$$

At the beginning of an update, $\theta = \theta_{\text{old}}$ and $\rho_t = 1$. Clipping is inactive, so the initial direction agrees with the usual advantage policy gradient:

- For $\hat{A}_t > 0$, increase the density of this sampled action; **do not** regress the network output to it as a supervised label.
- For $\hat{A}_t < 0$, decrease that action's density.
- For $\hat{A}_t \approx 0$, the sample gives little local policy-update signal.

These are **per-sample, local gradient contributions**. Shared network parameters may change densities at many states, and other samples can counteract an individual contribution. PPO clipping removes additional surrogate gain for favorable but overly large ratio changes; the exact piecewise gradient was derived in Section 17.

25.2 A numerical example of an actor update

Suppose a sampled discrete action has $\pi_{\text{old}}(a_t | s_t) = 0.20$, $\epsilon = 0.2$, and $\hat{A}_t = +2$. If the new probability is 0.22, the ratio is 1.1 and the sample's unclipped contribution is 2.2. If it becomes 0.30, the ratio is 1.5 but PPO-Clip credits only $1.2 \times 2 = 2.4$ to this sample: further increases yield no additional clipped surrogate gain. An analogous plateau occurs when a **negative-advantage** action's probability is decreased too far. This discrete example is for intuition; a continuous Gaussian policy uses **densities** and joint action log probabilities, not nonzero single-point probabilities.

26. Critic loss

Train the critic to regress the TD(λ) return:

$$\boxed{\mathcal{L}_{\text{critic}}(\phi) = \mathbb{E}_t[(V_\phi(s_t) - \hat{R}_t^\lambda)^2]}. \quad (37)$$

Direct Monte Carlo return targets are another option. Some PPO implementations use value-function clipping or multiply the squared error by $1/2$; neither is essential to the basic PPO-Clip objective. An inaccurate critic can corrupt the sign or magnitude of advantages, so policy-update stability depends strongly on value estimation quality.

26.1 How a critic prediction error changes network parameters

Using a per-sample MSE with the conventional factor $\frac{1}{2}$,

$$\ell_t^V(\phi) = \frac{1}{2}(V_\phi(s_t) - \hat{R}_t^\lambda)^2, \quad \boxed{\nabla_\phi \ell_t^V = (V_\phi(s_t) - \hat{R}_t^\lambda) \nabla_\phi V_\phi(s_t)}.$$

For example, let $V_{\text{old}}(s_t) = 4$, $V_{\text{old}}(s_{t+1}) = 5$, $r_t = 2$, and $\gamma = 0.9$, with no termination. The one-step TD target and error are

$$y_t^{\text{TD}} = 2 + 0.9 \times 5 = 6.5, \quad \delta_t = 6.5 - 4 = 2.5.$$

If $V(s_t)$ were an independently adjustable scalar, $\partial \ell / \partial V = 4 - 6.5 = -2.5$, so gradient descent would raise its prediction. In a neural network, the error is propagated through $\nabla_{\phi} V_{\phi}(s_t)$ to shared parameters, so an individual state's predicted value need not change monotonically.

The prediction concerns long-term return, not immediate reward. Even when $r_t = 0$, bootstrap values or subsequent rewards can supply a learning signal. Critic training is noisy, bootstrapped regression against estimated targets, not supervised regression against known ground truth.

26.2 Does lower critic loss guarantee a good value function?

No. Targets depend on the old critic, finite rollout, and stochastic rewards. Actor updates also alter the next iteration's policy and state distribution. Low training MSE alone proves neither unbiased value estimates nor higher episode return. A common diagnostic is **explained variance**:

$$\text{EV} = 1 - \frac{\text{Var}(\hat{R}_t^{\lambda} - V_{\phi}(s_t))}{\text{Var}(\hat{R}_t^{\lambda})}.$$

It becomes unstable when the target variance is nearly zero, and the target itself may still be biased. Inspect success rates, actual returns, and target/value scales as well. Some implementations clip changes in value predictions; value clipping does not guarantee greater true value accuracy.

27. Motivation for the entropy bonus

A stochastic policy may become too concentrated early, ceasing to explore alternative grasps. Use the policy entropy

$$H(\pi_{\theta}(\cdot | s)) = -\mathbb{E}_{a \sim \pi_{\theta}}[\log \pi_{\theta}(a | s)]. \quad (38)$$

to encourage sufficient exploration. Let $L^{\text{entropy}} = \mathbb{E}_t[H(\pi_{\theta}(\cdot | s_t))]$. Higher entropy is not always preferable, particularly near robot deployment. The entropy coefficient may be zero or annealed during training.

27.1 How does entropy regularization affect Gaussian standard deviation?

The entropy of an unsquashed one-dimensional Gaussian $\mathcal{N}(\mu, \sigma^2)$ is

$$H(\pi) = \frac{1}{2} \log(2\pi e \sigma^2) = \frac{1}{2} \log(2\pi e) + \log \sigma, \quad \frac{\partial H}{\partial \log \sigma} = 1.$$

Consequently, minimizing the total loss with $-c_e H$ contributes a gradient encouraging larger $\log \sigma$ when $c_e > 0$. The advantage-weighted policy gradient may raise or lower the standard deviation, as explained in Section 30. **More entropy is not always desirable**: excessive action noise can destabilize contact and trigger physical safety limits. Entropies add across independent Gaussian dimensions. The entropy of a tanh-squashed action distribution is not identical to that of its underlying Gaussian.

28. Final PPO training objective

A common joint maximization objective combines all three terms:

$$L^{\text{PPO}}(\theta, \phi) = L^{\text{CLIP}}(\theta) - c_v \mathcal{L}_{\text{critic}}(\phi) + c_e L^{\text{entropy}}(\theta). \quad (39)$$

Equivalently, minimize

$$\mathcal{L}_{\text{total}} = -L^{\text{CLIP}} + c_v \mathcal{L}_{\text{critic}} - c_e L^{\text{entropy}}. \quad (40)$$

- c_v determines the relative weight of value regression.
- c_e determines the strength of entropy regularization.
- With separate actor and critic parameters, the two networks can have separate updates; with shared parameters, their gradients interact.

A PPO optimization phase typically uses several mini-batch epochs over a fixed rollout, keeping the stored old log probabilities, advantages, and return targets constant during that phase.

28.1 The gradient graph: which loss trains which network?

After rollout collection and GAE, the learning signal splits into **two different regression/optimization paths**:

$$\begin{aligned} \pi_{\theta_{\text{old}}} &\longrightarrow (s_t, a_t, r_t, s_{t+1}, \log \pi_{\text{old}}, V_{\text{old}}) \\ &\downarrow \text{TD errors and GAE} \\ \hat{A}_t &\xrightarrow{\text{stopgrad}} L^{\text{CLIP}}(\theta) \longrightarrow \text{actor parameters } \theta, \\ \hat{R}_t^\lambda = \hat{A}_t + V_{\text{old}}(s_t) &\xrightarrow{\text{stopgrad}} \mathcal{L}_{\text{critic}}(\phi) \longrightarrow \text{critic parameters } \phi. \end{aligned}$$

Why stop the gradient through advantage? The actor should improve the surrogate by changing $\pi_\theta(a_t | s_t)$, not by manipulating V_ϕ or the labels used to score sampled actions. Freezing advantage does not prevent the critic from learning through its own loss.

With separate networks, actor and critic can have independent optimizers and learning rates. With a shared visual encoder/backbone, gradients of the actor, critic, and entropy terms all reach the shared parameters. **stopgrad** still freezes the **advantage and return targets**, but does not block the valid critic-loss gradient through its shared backbone. Sharing saves parameters but can create gradient competition; it is an architectural choice, not a PPO requirement.

28.2 Asymmetric actor–critic for robotics

During simulation training, an actor may receive only deployable camera images, proprioception, or estimated states, while the critic also receives privileged simulator information such as exact object poses or contact states:

$$\pi_\theta(a_t | o_t), \quad V_\phi(x_t), \quad x_t \text{ may contain privileged training information.}$$

This **asymmetric actor–critic** setup can simplify value prediction, but privileged inputs must **never inadvertently leak into the deployed actor**. For history dependence and partial observability, define the critic’s information set and bootstrap assumptions carefully. Asymmetry is optional and separate from the clipped PPO objective.

Part VIII. Continuous-Action PPO and Robot Control

29. Gaussian policy log probabilities and importance ratios

Robot actions are often continuous vectors: joint position increments, desired velocities, or torques. A typical actor uses a diagonal Gaussian

$$\pi_\theta(a | s) = \mathcal{N}(\mu_\theta(s), \text{diag}(\sigma_1^2, \dots, \sigma_n^2)). \quad (41)$$

The joint log probability of an action vector is

$$\log \pi_{\theta}(a | s) = -\frac{1}{2} \sum_{i=1}^n \left[\frac{(a_i - \mu_i)^2}{\sigma_i^2} + 2 \log \sigma_i + \log(2\pi) \right]. \quad (42)$$

The numerically convenient ratio is

$$r_t = \exp(\log \pi_{\theta}(a_t | s_t) - \log \pi_{\text{old}}(a_t | s_t)). \quad (43)$$

Implementation detail. Sum per-dimension log probabilities to obtain the **joint** action log probability *before* exponentiation. Clipping ratios separately for each action dimension is not standard PPO for a joint policy. A tanh-squashed Gaussian requires the appropriate change-of-variables correction to its log density. If an environment directly hard-clips sampled actions, carefully distinguish the density of the sampled action from that of the executed command.

30. How does a Gaussian-policy gradient move the action mean?

Consider a one-dimensional Gaussian with fixed variance:

$$a \sim \mathcal{N}(\mu_{\theta}(s), \sigma^2).$$

Differentiating its log density with respect to the mean gives

$$\frac{\partial \log \pi_{\theta}(a | s)}{\partial \mu_{\theta}} = \frac{a - \mu_{\theta}(s)}{\sigma^2}. \quad (44)$$

The mean-related component of the unclipped policy gradient is therefore

$$\nabla_{\theta} J \propto \mathbb{E} \left[\hat{A} \frac{a - \mu_{\theta}(s)}{\sigma^2} \nabla_{\theta} \mu_{\theta}(s) \right]. \quad (45)$$

For $\hat{A} > 0$, this tends to move the mean toward the sampled action; for $\hat{A} < 0$, it tends to move the mean away. PPO clipping limits further *surrogate benefit* from excessive probability changes. If σ is learned, there are additional gradient terms associated with variance.

30.1 Learning log standard deviation as well as the mean

Let the Gaussian standard deviation be trainable and define $\eta_{\theta}(s) = \log \sigma_{\theta}(s)$. In one dimension,

$$\log \pi_{\theta}(a | s) = -\frac{1}{2} \left[\frac{(a - \mu_{\theta}(s))^2}{e^{2\eta_{\theta}(s)}} + 2\eta_{\theta}(s) + \log(2\pi) \right].$$

Differentiating with respect to the mean and log standard deviation gives

$$\frac{\partial \log \pi_{\theta}}{\partial \mu_{\theta}} = \frac{a - \mu_{\theta}}{\sigma_{\theta}^2}, \quad \frac{\partial \log \pi_{\theta}}{\partial \eta_{\theta}} = \frac{(a - \mu_{\theta})^2}{\sigma_{\theta}^2} - 1.$$

For an unclipped sample, gradient **ascent** on the policy objective multiplies both terms by $\hat{A}_t \rho_t$. With positive advantage, a sample more than one standard deviation from the mean ($|a - \mu| > \sigma$) tends to increase $\log \sigma$; a closer sample tends to decrease it. Negative advantage reverses those tendencies. A good action therefore does not automatically imply lower noise: the actor learns both a distribution's **location and width**.

Two common continuous-control parameterizations are:

1. **State-independent log standard deviation.** The network predicts $\mu_\theta(s)$, while trainable parameters η_i supply one log standard deviation per action dimension. This is simple and common in locomotion implementations.
2. **State-dependent log standard deviation.** The network predicts both $\mu_\theta(s)$ and $\eta_\theta(s)$. This can be more expressive but has different stability considerations.

Practical implementations may bound log standard deviations and monitor both entropy and executed control noise. Training typically samples stochastic actions; evaluation/deployment often uses the mean or another deterministic mapping, but PPO does not mathematically require this deployment convention. A tanh-squashed Gaussian requires consistently transformed log densities.

31. Example: residual PPO for robot manipulation

Suppose a baseline controller produces

$$a_t^{\text{base}} = \pi_{\text{base}}(s_t),$$

while PPO learns a stochastic residual

$$\Delta a_t \sim \pi_\theta(\cdot \mid s_t).$$

The executed command is

$$a_t = a_t^{\text{base}} + \Delta a_t. \quad (46)$$

The residual may adjust joint targets, velocities, end-effector pose.

The ratio must correspond to the **residual action distribution**:

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

When residuals are scaled, transformed nonlinearly, or filtered through a safety controller, clearly distinguish which variables have the log density used for learning and which commands are actually executed. PPO does not require differentiable contact dynamics, but does require adequate rollouts, rewards, and state/value estimation.

Part IX. A Complete PPO Iteration

32. Ten steps from rollout collection to policy updates

1. **Freeze the old policy.** Set $\pi_{\text{old}} = \pi_{\theta_k}$ and sample several episodes or fixed-length rollouts.
2. **Store rollout data.** Record (s_t, a_t, r_t, s_{t+1}) , termination flags, $\log \pi_{\text{old}}(a_t \mid s_t)$, and $V_{\text{old}}(s_t)$.
3. **Compute TD errors.** Use $\delta_t = r_t + \gamma V_{\text{old}}(s_{t+1}) - V_{\text{old}}(s_t)$ with proper true-terminal masking.
4. **Calculate GAE backward.** Use $\hat{A}_t = \delta_t + \gamma \lambda \hat{A}_{t+1}$ and prevent recursion across unrelated episodes.
5. **Compute critic targets.** Set $\hat{R}_t^\lambda = \hat{A}_t + V_{\text{old}}(s_t)$ and hold the targets fixed.
6. **Optionally normalize advantages.** Use $\tilde{A}_t = (\hat{A}_t - \mu_A) / (\sigma_A + \varepsilon_{\text{num}})$.
7. **Evaluate current log probabilities on stored actions.** For the **original actions** a_t sampled by the old policy, compute current $\log \pi_\theta(a_t \mid s_t)$.
8. **Compute ratios and clipped actor loss.** Use $r_t = \exp(\log \pi_\theta - \log \pi_{\text{old}})$ and Eq. (21).
9. **Optimize actor and critic.** Minimize the joint objective or separate losses for multiple epochs, monitoring KL, clip fraction, and value prediction.
10. **Collect new on-policy data.** Use the updated policy for the next rollout rather than treating arbitrarily old replay data as ordinary PPO training data.

32.1 When do the actor and critic parameters actually change?

Interpret the ten steps above as distinct phases:

1. **Rollout and GAE.** Environment states, sampled actions, and rewards change during collection; $\hat{A}_t$ and $\hat{R}_t^\lambda$ are then computed once. The behavior policy π_{old} and the critic snapshot V_{old} remain fixed while those targets are created.
2. **K minibatch epochs.** Update current actor parameters θ , critic parameters ϕ , and the resulting *new* log probabilities and value predictions. The sampled actions, *old* log probabilities, $\hat{A}_t$, and $\hat{R}_t^\lambda$ remain frozen.
3. **Next rollout.** The updated actor becomes the new behavior policy. Gather new data instead of treating arbitrarily old samples as fresh on-policy PPO experience.

Thus, within one PPO optimization phase, **critic predictions can change but the critic regression targets remain fixed**. Recomputing old log probabilities or advantage from the moving parameters inside the same update changes the surrogate away from the behavior-policy snapshot it was designed for.

33. Compact, adaptable PyTorch implementations

The following function implements the central PPO actor loss. Both `new_log_prob` and `old_log_prob` must contain **joint** action log probabilities, and the old log probabilities and advantages must remain fixed within the update phase.

```
import torch

def ppo_actor_loss(new_log_prob, old_log_prob,
                  advantages, clip_eps=0.2):
    # The old log probability and advantage are fixed rollout data.
    old_log_prob = old_log_prob.detach()
    advantages = advantages.detach()

    log_ratio = new_log_prob - old_log_prob
    ratio = torch.exp(log_ratio)

    objective_unclipped = ratio * advantages
    objective_clipped = torch.clamp(
        ratio, 1.0 - clip_eps, 1.0 + clip_eps
    ) * advantages

    return -torch.minimum(
        objective_unclipped, objective_clipped
    ).mean()
```

A minimal backward GAE calculation is shown next. The `bootstrap_mask[t]` determines whether a subsequent-state value is included in the TD target. The `continue_mask[t]` determines whether the advantage recursion is allowed to cross the current transition. Correct handling of true terminations, timeouts, and rollout boundaries must come from the environment and rollout buffer.

```
import torch

def compute_gae(rewards, values, next_values,
               bootstrap_mask, continue_mask,
               gamma=0.99, lam=0.95):
    """All tensors are indexed by rollout time."""
    T = rewards.shape[0]
    advantages = torch.zeros_like(rewards)
    running = torch.zeros_like(rewards[0])

    for t in reversed(range(T)):
        delta = (rewards[t]
                 + gamma * bootstrap_mask[t] * next_values[t]
                 - values[t])
        running = (delta
                  + gamma * lam * continue_mask[t] * running)
```

```

        advantages[t] = running

    returns = advantages + values
    return advantages, returns

```

For parallel simulation, tensors can have shape `[T, N_env]`; never propagate `continue_mask` across an episode reset. The `values` and `next_values` used in return targets should come from the fixed rollout-time critic.

For a diagonal Gaussian policy:

```

# dist: torch.distributions.Normal, action shape [batch, action_dim]
new_log_prob = dist.log_prob(actions).sum(dim=-1)
# entropy shape [batch]
entropy = dist.entropy().sum(dim=-1)

```

These examples implement the essential formulas; a complete training program still needs a rollout buffer, environment interaction, network definitions, optimizers, correct `terminated/truncated` handling, and metric logging.

33.1 Combining the three losses into one differentiable minibatch update

The following addition illustrates the three losses for an **already prepared, fixed rollout minibatch**. It does not replace the GAE/termination-mask handling above, and it is not a complete training package.

```

import torch

def ppo_joint_losses(actor, critic, obs, actions,
                    old_log_prob, advantages, returns,
                    clip_eps=0.2, value_coef=0.5,
                    entropy_coef=0.0):
    # actor(obs) returns [batch, action_dim] mean and log_std.
    mean, log_std = actor(obs)
    dist = torch.distributions.Independent(
        torch.distributions.Normal(mean, log_std.exp()), 1
    )
    new_log_prob = dist.log_prob(actions)
    actor_loss = ppo_actor_loss(
        new_log_prob, old_log_prob.detach(),
        advantages.detach(), clip_eps
    )
    value_pred = critic(obs).squeeze(-1)
    value_loss = 0.5 * (value_pred - returns.detach()).square().mean()
    entropy = dist.entropy().mean()
    total_loss = actor_loss + value_coef * value_loss - entropy_coef * entropy
    return total_loss, actor_loss, value_loss, entropy

```

With separate parameters, actor and critic losses can be backpropagated and stepped with separate optimizers. With a shared backbone, `total_loss.backward()` aggregates gradients on shared parameters. The `Independent(Normal, 1)` distribution yields the joint log probability for the entire action vector; other action distributions require matching density calculations. Importantly, `returns` is computed from **old values plus GAE** *before* the optimization epochs.

Part X. Hyperparameters and Diagnostics

34. Common starting hyperparameters and what they mean

Hyperparameter	Common starting point	Main role
Discount γ	0.99	Weight assigned to distant rewards
GAE λ	0.95	Advantage-estimator bias–variance tradeoff
Clip ϵ	0.1–0.2	Extra surrogate gains from overly large probability ratios
Actor learning rate	3×10^{-4}	Actor update speed
PPO epochs	3–10	Number of optimization passes over one rollout
Target KL	0.01–0.03	Threshold for early stopping of actor updates
Entropy coefficient	0 or small positive	Amount of exploration encouragement
Value-loss weight	About 0.5	Relative weight of critic regression

These are experimental starting points, not universal optima. Large-scale parallel locomotion frameworks such as RSL-RL may use quite different learning rates, batch sizes, clipping ranges, and numbers of epochs.

The distinction between γ and λ : The discount factor γ defines how the learning objective values distant rewards; λ determines how much an advantage estimator incorporates distant sampled TD information. Both appear through $(\gamma\lambda)^l$ in GAE, but their conceptual roles are different.

Useful diagnostics include episode return, task success rate, value loss, explained variance, policy entropy, approximate KL, clip fraction, and gradient norm. If a surrogate improves while actual robot success decreases, investigate distribution shift, critic error, excessive KL, and reward design.

34.1 Reading diagnostics as actual actor–critic learning problems

- **High value loss or low explained variance.** Possible causes include noisy critic targets, limited approximation capacity, bootstrap bias, and changing reward scales. Inspect target/value ranges, true termination masks, critic learning rate, and normalization.
- **Actor loss improves while success does not.** The surrogate objective may fail to track actual performance. Check *new* rollout returns, approximate KL, clip fraction, reward design, and advantage noise.
- **Persistently high approximate KL or clip fraction.** The actor may be updating too quickly on a fixed rollout. Check actor learning rate, PPO epochs, minibatch size, and target-KL stopping.
- **Entropy collapses early.** Inspect shrinking trainable log standard deviations, their allowed range, the action distribution, and the entropy coefficient.
- **Robot contact becomes jittery.** Excessive exploratory noise or poorly chosen action scaling/control frequency may be responsible. Check action scale, log standard deviation, control period, and joint/contact limits.

Two learning rates and two time scales matter. The actor learning rate controls policy drift; the critic learning rate controls tracking of changing value targets. A critic that learns too slowly may corrupt GAE; overly aggressive updates can overfit noisy targets or destabilize optimization. Do not tune either rate from value loss alone: monitor actual task performance and policy KL. More epochs reuse data more aggressively but also increase departure from the data-collecting policy.

Part XI. Connections to REINFORCE, Actor–Critic, and TRPO

35. Algorithmic progression: one design choice per problem

Algorithm	Core expression / idea	Problem addressed
REINFORCE	$\nabla \log \pi_{\theta} G$	Learn a policy without environment derivatives
Advantage actor-critic	$\nabla \log \pi_{\theta} \hat{A}$	Reduce variance using a state-value baseline
TRPO	Maximize $\mathbb{E}[r\hat{A}]$ subject to KL	Keep the surrogate useful in a local trust region
PPO-Clip	$\min(r\hat{A}, \text{clip}(r)\hat{A})$	Discourage excessive probability changes with first-order optimization
PPO with GAE	$\hat{A}_t = \sum_l (\gamma\lambda)^l \delta_{t+l}$	Control the bias-variance tradeoff of advantage estimates

PPO fundamentally remains an **on-policy policy-gradient / actor-critic algorithm**. At the beginning of each rollout update, the clipped and unclipped surrogate objectives have the same first-order gradient. Their difference arises as several optimization steps on the same dataset cause probability ratios to move away from one.

PPO is especially practical for simulated robotic control because large numbers of environments can be sampled in parallel, optimization is relatively simple, differentiable dynamics are unnecessary, and Gaussian policies naturally model continuous actions. On a physical robot, however, collecting interactions is expensive, so off-policy methods such as SAC may offer important sample-efficiency advantages. Diffusion and flow-matching action policies often lack the conveniently computable explicit log probabilities required by standard PPO, motivating specialized modifications.

Part XII. Final Formula Sheet and Research Takeaways

Formula 1: Policy gradient

$$\nabla_{\theta} J \propto \mathbb{E} [\nabla_{\theta} \log \pi_{\theta}(a | s) A^{\pi}(s, a)].$$

Motivation: Optimize without backpropagating through possibly nondifferentiable physical dynamics.

Formula 2: Importance sampling

$$r_t = \pi_{\theta}(a_t | s_t) / \pi_{\text{old}}(a_t | s_t).$$

Motivation: Construct a local new-policy surrogate from actions sampled by the old policy.

Formula 3: TRPO's trust-region idea

$$\bar{D}_{\text{KL}}(\pi_{\text{old}} \| \pi_{\theta}) \leq \delta.$$

Motivation: Control the risk that the surrogate becomes misleading under distribution shift.

Formula 4: PPO clipped surrogate

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

Motivation: Conservatively cap surrogate incentives for excessive favorable changes while retaining corrective gradients for unfavorable changes.

Formula 5: Generalized advantage estimation

$$\hat{A}_t = \sum_{l \geq 0} (\gamma \lambda)^l \delta_{t+l}, \quad \delta_t = r_t^{\text{reward}} + \gamma V(s_{t+1}) - V(s_t).$$

Motivation: Smoothly trade off reliance on long-term sampled rewards against reliance on short-horizon value bootstrapping. Here r_t^{reward} distinguishes the immediate reward from PPO's probability ratio $r_t(\theta)$.

Final loss to minimize:

$$\mathcal{L}_{\text{PPO}} = -L^{\text{CLIP}} + c_v \mathcal{L}_{\text{critic}} - c_e L^{\text{entropy}}.$$

Main research insight. PPO is not a new method for differentiating environments. It uses policy gradients to identify local directions of improvement, importance sampling to construct a surrogate from stored actions, clipping to discourage over-optimization, and GAE to improve advantage estimation. **GAE principally addresses estimator quality; clipping principally addresses policy-update stability.** The two components solve different yet complementary problems.

References and further reading

1. Schulman, J., et al. (2017). *Proximal Policy Optimization Algorithms*. arXiv:1707.06347. <https://arxiv.org/abs/1707.06347>
2. Schulman, J., et al. (2015). *Trust Region Policy Optimization*. ICML. <https://arxiv.org/abs/1502.05477>
3. Schulman, J., et al. (2015). *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. ICLR 2016. <https://arxiv.org/abs/1506.02438>
4. OpenAI Spinning Up. *PPO Documentation*. <https://spinningup.openai.com/en/latest/algorithms/ppo.html>

Copyright and license: CC BY 4.0.

Code note: Code fragments illustrate algorithmic mechanisms; they have not been validated as end-to-end training scripts.