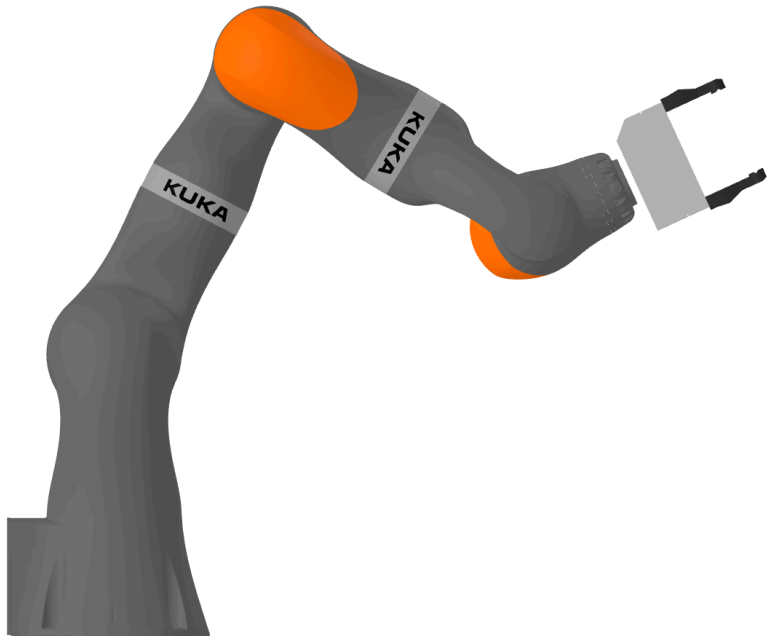


Reinforcement Learning II

MIT 6.4210/2

Robotic Manipulation

Fall 2026, Lecture 11

Massachusetts Institute of Technology

Reinforcement learning

Lecture 11: baselines, PPO, and robot learning

Leslie Kaelbling

6.4210 Fall 2026

Return to go

An action needs credit only for rewards that follow it

$$g_t = \sum_{k=t}^{H-1} \gamma^{k-t} r_k$$

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

The outer γ^t discounts from time zero
For an undiscounted finite-horizon objective, $\gamma = 1$

A baseline gives a comparison

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{t=0}^{H-1} \gamma^t \big(g_t - b_t(s_t)\big) \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]$$

- A good baseline **reduces variance in the gradient updates**, making learning less noisy
- A fixed, action-independent baseline preserves the expected gradient
- Predicting the expected return from the current state gives a useful baseline

$b_t(s)$: action-independent baseline for state s at time t
Variance reduction depends on the baseline; an arbitrary baseline need not help

4

Why an action-independent baseline cancels

At a fixed state, the average log-probability gradient is zero:

$$\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[\nabla_\theta \log \pi_\theta(a \mid s)] = \sum_a \pi_\theta(a \mid s) \frac{\nabla_\theta \pi_\theta(a \mid s)}{\pi_\theta(a \mid s)}$$
$$= \nabla_\theta \sum_a \pi_\theta(a \mid s) = \nabla_\theta 1 = 0$$

Multiplying by a fixed $b_t(s)$ still gives zero

- Fitting the baseline to this batch *before* the policy update can make it depend indirectly on the sampled actions
- Using the previous baseline avoids that dependence

Learning the baseline

The expected return is a useful baseline:

$$b_\phi(s,t) \approx V_t^{\pi_\theta}(s) = \mathbb{E}[G_t \mid S_t = s]$$

- This mean minimizes the expected squared weight $(G_t - b_t(s))^2$
- It often reduces gradient variance, but need not minimize it
- Regression uses (s_t, t) as input and observed g_t as the target

The current baseline supplies the policy-update weights
We then fit it to this batch's returns for use on the next batch

$b_\phi(s,t)$: learned baseline with weights ϕ G_t: random return to go

REINFORCE with a learned baseline

Problem inputs: *env* (including p_0), horizon H, discount γ

Choose batch size N and step size α (or a schedule); initialize θ, ϕ

1. Collect N independent H-step rollouts of π_θ in *env*, each from $s_0 \sim p_0$; trajectory index $i = 1, \ldots, N$
2. Compute backward: $g_H^{(i)} = 0, \quad g_t^{(i)} = r_t^{(i)} + \gamma g_{t+1}^{(i)}$
3. Return minus baseline gives each step's weight: $w_t^{(i)} = g_t^{(i)} - b_\phi(s_t^{(i)}, t)$
4. Hold $w_t^{(i)}$ fixed and take one policy-gradient step:
   $$\theta \leftarrow \theta + \frac{\alpha}{N} \sum_{i,t} \gamma^t w_t^{(i)} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)})$$
5. Regress returns on state and time: $\phi \leftarrow \text{regress}(\{((s_t^{(i)}, t), g_t^{(i)})\}_{i,t})$
6. Discard the batch and repeat from step 1

Proximal policy optimization (PPO)

Problem inputs: *env* (including p_0), horizon H, discount γ

Choose N, α, clipping tolerance ϵ, updates per batch M; initialize θ, ϕ

1. Save $\theta_{\text{old}} \leftarrow \theta$; roll out $\pi_{\theta_{\text{old}}}$ for N complete H-step episodes in *env*, each from a fresh $s_0 \sim p_0$
2. Compute backward: $g_H^{(i)} = 0, \quad g_t^{(i)} = r_t^{(i)} + \gamma g_{t+1}^{(i)}$
3. Save $\widehat{\text{adv}}_t^{(i)} = g_t^{(i)} - V_\phi(s_t^{(i)}) = w_t^{(i)}$ and $p_t^{\text{old},(i)} = \pi_{\theta_{\text{old}}}(a_t^{(i)} \mid s_t^{(i)})$
4. Hold these estimates fixed; take M updates using the clipped loss:

$$\theta \leftarrow \theta + \alpha \nabla_\theta L_{\text{clip}}(\theta)$$

5. Regress returns on state and time: $\phi \leftarrow \text{regress}(\{((s_t^{(i)}, t), g_t^{(i)})\}_{i,t})$
6. Discard the batch and repeat from step 1

$V_\phi(s_t) = b_\phi(s_t, t)$: same baseline Blue: changes from REINFORCE

Learning dexterous manipulation



Click the image to play the video

OpenAI (2019): Solving Rubik's Cube with a Robot Hand: Perturbations

Comparing the new and old policy

PPO calls the return-minus-baseline weight an *advantage estimate*:

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s), \qquad \widehat{\mathrm{adv}}_t = g_t - V_{\phi}(s_t)$$

The probability ratio compares the new and collection policies:

$$\rho_t(\theta) = \frac{\pi_{\theta}(a_t \mid s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)}$$

- Initially $\rho_t = 1$; positive advantage favors increasing this action's probability
- The ratio adjusts action probabilities at recorded states

θ_{old}: weights that collected the batch ρ_t: new-to-old probability ratio
$Q^{\pi}(s, a)$: return from taking a, then following π $V^{\pi}(s)$: return under π

Why the ratio gives the initial policy gradient

The samples, old probabilities, and advantage estimates are fixed

$$\begin{aligned}\nabla_\theta[\rho_t(\theta)\widehat{adv}_t] &= \widehat{adv}_t \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \\ &= \widehat{adv}_t \rho_t(\theta) \nabla_\theta \log \pi_\theta(a_t \mid s_t)\end{aligned}$$

At $\theta = \theta_{old}$, the ratio is 1, giving the usual advantage-weighted log-probability gradient

This objective uses states already in the batch
A changed policy also visits different states, so the approximation becomes less reliable as the policy moves away

The clipped PPO objective

While reusing a batch, we want to avoid moving too far from the policy that generated that experience

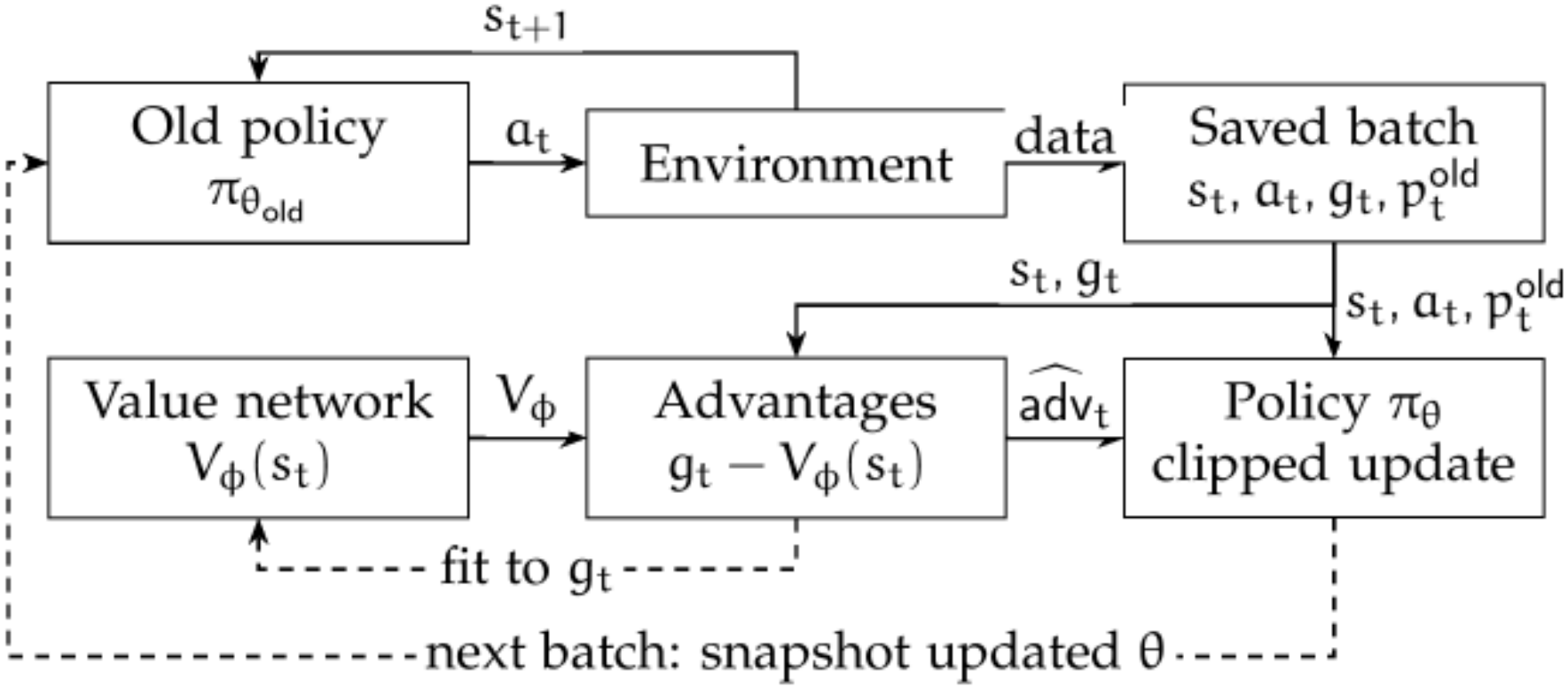
$$\ell_t(\theta) = \min\Big(\rho_t(\theta)\widehat{\mathsf{adv}}_t,\ \mathsf{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{\mathsf{adv}}_t\Big)$$

$$L_{\mathsf{clip}}(\theta) = \widehat{\mathbb{E}}_{\tau\sim p_{\theta_{\mathsf{old}}}}\left[\sum_t \gamma^t \ell_t(\theta)\right]$$

Clipping stops rewarding large changes in the favored direction
It does not impose a hard bound on the updated ratio

ℓ_t: sample's objective term L_{clip}: batch objective $\widehat{\mathbb{E}}$: batch average
ϵ: clipping tolerance $\mathsf{clip}(x, l, u) = \min(\max(x, l), u)$: clamp x between bounds l, u

PPO: data and learning signals



Solid arrows: data flow Dashed arrows: weight updates

$p_t^{old} = \pi_{\theta_{old}}(a_t \mid s_t)$: saved probability or density of the sampled action

Generalized advantage estimation

GAE combines rewards with old-critic predictions from short rollouts

$$\delta_t = r_t + \gamma V_{\phi_{\text{old}}}(s_{t+1}) - V_{\phi_{\text{old}}}(s_t), \qquad \widehat{\text{adv}}_t = \sum_{\ell=0}^{K-1} (\gamma\lambda)^\ell \delta_{t+\ell}$$

- $\lambda = 0$: one TD error; $\lambda = 1$ over a complete episode: $g_t - V_{\phi_{\text{old}}}(s_t)$
- Longer returns are noisy; bootstrapping can add prediction bias
- Terminal: next value zero; cutoff: bootstrap; stop sums at resets
- Use the old critic and keep advantages fixed during updates

K: steps to the segment or episode boundary $\quad \lambda \in [0, 1]$

Common fixed value target: $\widehat{\text{adv}}_t + V_{\phi_{\text{old}}}(s_t)$; HW6 details are in the notes' appendix

Combining value and policy methods

- The *actor* represents a policy and chooses actions
- The *critic* predicts returns and guides policy improvement
- PPO already combines them: its value network helps estimate advantages

Off-policy actor-critic methods can keep learning from a replay buffer collected by earlier policies

A learned actor for continuous actions

The actor learns the continuous-action search that maximizing $Q(s, a)$ requires

Critic: minimize prediction error in ϕ, holding the targets y fixed

$$L_Q(\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}[(Q_\phi(s,a) - y)^2], \qquad y = r + \gamma \bar{Q}(s', \bar{\pi}(s'))$$

Actor: maximize predicted return in θ, holding the critic fixed

$$L_\pi(\theta) = \mathbb{E}_{s\sim\mathcal{D}}[Q_\phi(s, \pi_\theta(s))]$$

The actor gradient passes through the critic's action input
We differentiate through learned networks, without differentiating through the world

$\mathcal{D}$: replay buffer $\bar{Q}, \bar{\pi}$: fixed target versions of the critic and actor
The bootstrap term is zero for a true terminal transition; $\pi_\theta(s)$ is a deterministic actor

Why use a stochastic policy?

For a finite MDP with the usual discounted-return objective, there is an optimal deterministic policy

- Stochastic policies can still help during learning
- Randomness provides exploration and can smooth the expected-return objective
- Searching this larger policy space can help optimization, but does not guarantee avoiding poor local optima

An entropy incentive gives the policy a reason to keep several options open

Maximum entropy RL

Entropy measures the uncertainty in the policy's action:

$$\mathcal{H}(\pi(\cdot \mid s)) = -\mathbb{E}_{a \sim \pi(\cdot|s)}[\log \pi(a \mid s)]$$

- For a finite action set, a deterministic choice has zero entropy
- Spreading probability across actions increases entropy
- For continuous actions, the same expression is differential entropy

$\pi(a \mid s)$: policy probability or density $\mathcal{H}$: entropy log: natural logarithm

Reward and entropy

$$J_\beta(\pi) = \mathbb{E}_{\tau \sim p_\pi(\mathcal{T})} \left[\sum_{t=0}^{\infty} \gamma^t \big(r_t + \beta \mathcal{H}(\pi(\cdot \mid s_t)) \big) \right]$$

- Temperature $\beta > 0$ controls the tradeoff; several comparably good actions can retain probability
- Entropy changes the objective, beyond adding exploration noise
- *Soft actor-critic* (SAC) optimizes this objective off-policy: a stochastic actor, replay, and typically two critics

J_β: reward-plus-entropy objective γ: discount r_t: reward
p_π: trajectory distribution induced by π

Value, policy, and combined methods

Method	Experience	Strengths and limitations
Value-based Q-learning	Off-policy replay	Reuses data; Bellman updates need a Markov state; maximizing over continuous actions can be difficult
Policy search REINFORCE, PPO	On-policy; PPO: limited batch reuse	Return-based gradients work for memoryless policies with non-Markov observations; fresh rollouts can be expensive
Actor-critic TD3, SAC	Off-policy replay	Learns continuous actions and reuses data; actor can exploit critic errors, which bootstrapping can amplify

Cheap simulation: PPO; costly interaction with Markov states: replay
PPO is also actor-critic; non-Markov critics can bias GAE

Combining behavior cloning and RL

- BC provides useful behavior from demonstrations
- RL supplies feedback from executing the policy and recovering from errors
- Sparse rewards are easier to use when the policy can already get near success

Demonstrations can guide exploration, initialize learning, or supply an imitation objective

BC-guided exploration and replay

Demonstrations can help an off-policy learner collect and reuse useful experience

- Early exploration mostly follows BC; later, mix in learned actions and random exploration to discover improvements
- Demonstrations can seed replay when we have transitions and their rewards
- Imitation only needs state–action pairs; these are different training signals
- Demonstrations are not fresh on-policy rollouts for REINFORCE or PPO

BC initialization and continued imitation

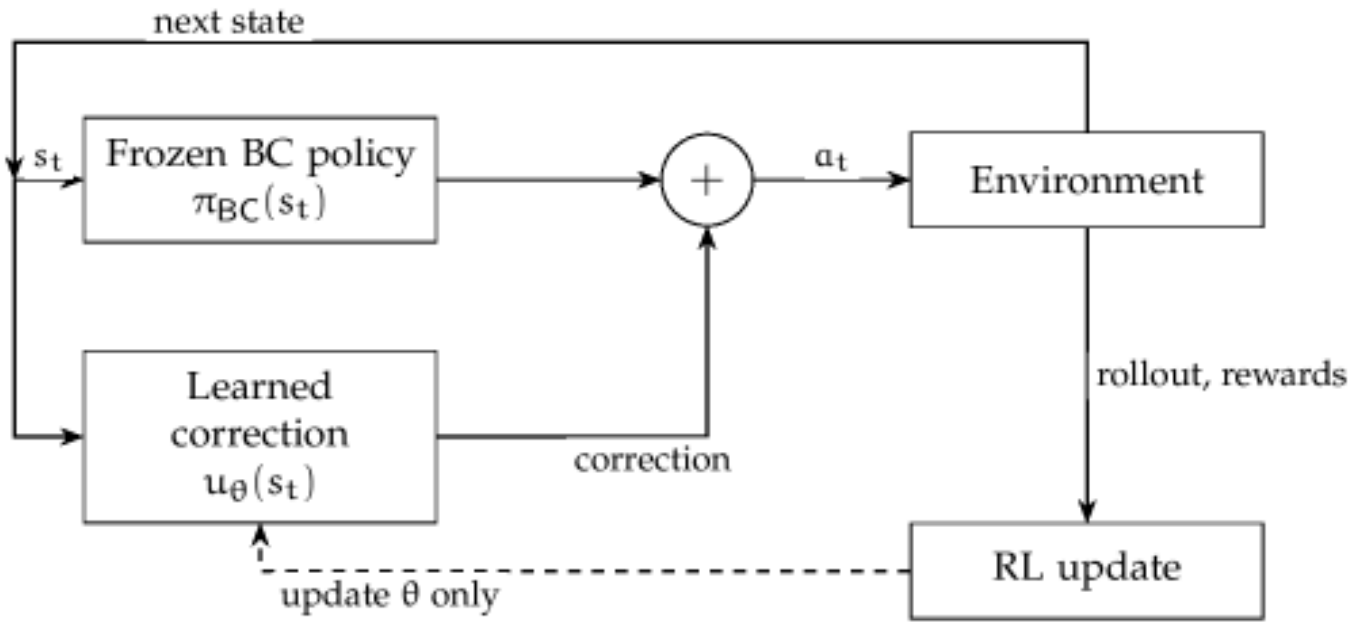
The cloned policy initializes the RL policy, and imitation can continue:

$$\max_{\theta} \; J_{\mathrm{RL}}(\theta) + \lambda_{\mathrm{BC}} \mathbb{E}_{(s,a^*)\sim\mathcal{D}_{\mathrm{demo}}} \left[\log \pi_\theta(a^* \mid s)\right]$$

- The RL term rewards outcomes; the BC term preserves demonstrated behavior
- Reducing λ_{BC} gives RL more freedom to improve
- Too much imitation can preserve mistakes; too little can lose the skill

J_{RL}: RL objective λ_{BC}: imitation weight
$\mathcal{D}_{\mathrm{demo}}$: demonstration pairs a^*: demonstrated action

Residual RL: a frozen policy and a learned correction



$$a_t = \pi_{BC}(s_t) + u_\theta(s_t)$$

The BC policy stays fixed; the learned correction starts near zero

RL adjusts for friction, placement errors, or required force

Small action corrections can change a trajectory substantially, yet may be insufficient for a new strategy

Solid arrows: data Dashed arrow: update θ only u_θ: learned residual

Sparse rewards: exploration, curricula, and shaping

For peg insertion, reward might be 1 for success and 0 otherwise
If the peg never gets near the hole, most attempts look equally bad

- Demonstrations or a controller can bring the robot near useful states; exploration then tries variations on a skill
- A *curriculum* starts with easier tasks, such as a peg close to the hole
- *Reward shaping* supplies intermediate feedback for progress

Shaping can change the best behavior: hovering near a target might become more rewarding than completing the task

The robot's learning setup

- Exploring gripper-position commands can be more useful than exploring raw joint torques
- A controller underneath the policy can supply useful structure
- Speed and force limits, interventions, and reset procedures affect what experience we can collect

A method with fewer attempts may still be slow if every reset needs a person

Partial observability, memory, and information gathering

An observation o_t is what the robot senses; s_t is the full world state

- An empty and a full box can look the same but need different lifting forces
- A history window or recurrent network can supply memory: $\pi(a \mid h_t)$ or $Q(h_t, a)$; training needs sequences
- Actions can reveal hidden information: a gentle lift can reveal mass

Memory does not automatically recover a Markov state
The benefit of gathering information may appear much later

h_t: history of observations and actions

Simulation and the reality gap

- Simulation makes experience and resets cheap
- A policy can depend on errors in simulated friction, contact, motor delays, or appearance
- This mismatch is the *reality gap*

Good performance in one simulated world does not establish performance on the real robot

Domain randomization

Training varies the simulated world:

- Object masses and friction
- Camera positions, lighting, and sensor noise
- Action delays

Training optimizes average return over these variations
Too little variation can leave a brittle policy; too much can make learning harder
If no policy succeeds in every world, average performance involves compromises

Teacher-student training

- A simulated teacher can see exact poses, contact forces, or friction: *privileged information* unavailable on the deployed robot
- A student uses supervised imitation to match the teacher's actions from the robot's actual observations, possibly with memory
- Training a student to imitate a teacher is called *distillation*
- Policy-gradient learning can subsequently improve the student

The deployed student acts with the information the real robot has