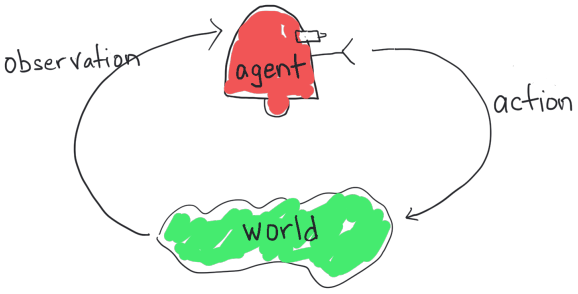
Reinforcement learning

Lecture 10: values, policy gradients, and PPO

Leslie Kaelbling

6.4210 Fall 2026

Learning a policy



The agent chooses an action

The world returns a new state and a scalar reward

We want actions that lead to a large total reward

We consider both discrete and continuous state and action sets

$\mathcal{S}$: set of states $\mathcal{A}$: set of actions π: policy
t: time step $s_t \in \mathcal{S}$: state $a_t \in \mathcal{A}$: action r_t: reward

What makes RL hard?

- A reward does not tell us which action we should have taken
- An action can affect rewards far into the future
- Our actions also determine the training data we collect

On a robot, a failed attempt may also require a physical reset

Different RL settings with different data and objectives

- **Online, in the world:** experience arrives as we act; we care about reward during learning as well as afterward
- **Online, in simulation:** we collect simulated experience; we care about the resulting policy's performance
- **Offline:** we learn from a fixed dataset of transitions (s_t, a_t, r_t, s_{t+1}), with no new exploration

We will mostly think about learning in simulation

Return and the horizon

The *return* adds up future rewards

$$\text{Finite horizon: } \sum_{t=0}^{H-1} r_t \qquad \text{Discounted: } \sum_{t=0}^{\infty} \gamma^t r_t$$

- The optimal finite-horizon policy depends on the time remaining
- With $0 \leqslant \gamma < 1$, distant rewards count less
- The infinite-horizon discounted problem has an optimal *stationary* policy: it does not depend on time (our focus)

H: number of steps in the horizon γ: discount factor

The optimal action-value function

$Q^*(s, a)$ is the expected return after taking a in s, then acting optimally

$$Q^*(s, a) = \max_{\pi} \mathbb{E}_{\tau \sim p_\pi(\mathcal{T}|s,a)} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

If we know Q^*, choosing an action is simple:

$$\pi^*(s) = \arg\max_{a} Q^*(s, a)$$

τ: a trajectory $\mathcal{T}$: random trajectory p_π: trajectory distribution under π
$\mathbb{E}$: expectation π^*: optimal policy arg max: action attaining the largest value

The Bellman equation

$$Q^*(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim P(S'|s,a)} \left[\max_{a'} Q^*(s', a') \right]$$

- $R(s, a)$ is the expected immediate reward
- $P(S' \mid s, a)$ describes the next state
- The value now is immediate reward plus discounted future value[1]

s': next state S': random next state a': candidate next action
P: transition distribution

[1] There are many Bellman equations: this is the optimality equation for Q^*

Q learning (for discrete states and actions)

One transition gives a sample-based, bootstrapped estimate of $Q(s, a)$

$$y = \underbrace{r}_{\text{observed reward}} + \gamma \underbrace{\max_{a'} Q[s', a']}_{\text{estimated future value}}$$

The value estimate moves toward that target

$$Q[s, a] \leftarrow Q[s, a] + \alpha \underbrace{\big(y - Q[s, a]\big)}_{\text{TD error}}$$

At a true terminal transition, $y = r$

$Q[s, a]$: current tabular value estimate y: learning target
α: learning rate TD: temporal difference (s, a, r, s'): observed transition

The Markov property

A *Markov state* summarizes everything in the history that helps predict the next state and reward, given the next action

$$P(S_{t+1}, R_t \mid S_0, A_0, \ldots, S_t, A_t) = P(S_{t+1}, R_t \mid S_t, A_t)$$

- Conditioning on the full past would not improve the prediction
- Position alone may be insufficient: velocity can affect what happens next

S_t, A_t, R_t: random state, action, and reward at time t
Lowercase letters denote their observed values

Convergence of tabular Q learning

For a finite Markov decision process with bounded rewards and $0 \leqslant \gamma < 1$

- Every state-action pair is visited infinitely often
- Each pair's learning rates satisfy

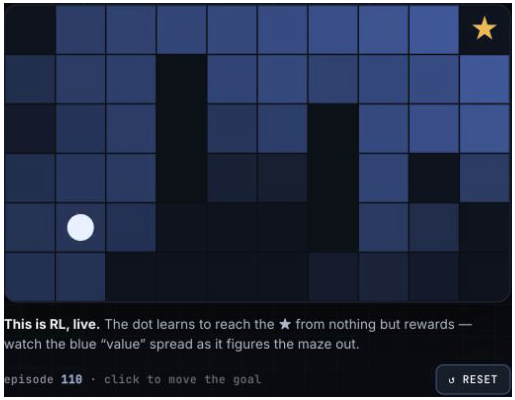
$$0 < \alpha_n(s, a) \leqslant 1, \qquad \sum_{n=1}^{\infty} \alpha_n(s, a) = \infty, \qquad \sum_{n=1}^{\infty} \alpha_n(s, a)^2 < \infty$$

Then Q converges to Q^* with probability one

These guarantees do not automatically carry over to neural networks

n: update count for the pair (s, a)
Exploration is essential: choosing a currently greedy action can miss better alternatives

Q learning in a gridworld



Click the image to play the recording

Demo: reinforcement-learning.com

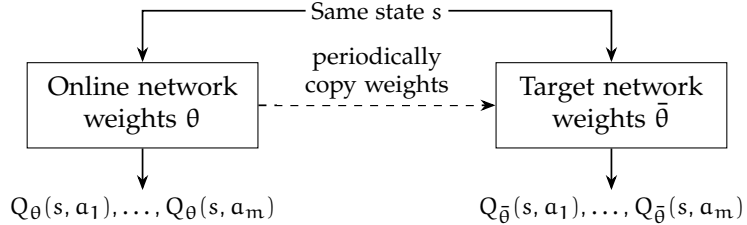
Reusing experience

A *replay buffer* stores past transitions:

$$\mathcal{D} = \{(s, a, r, s', \text{terminal})\}$$

- We sample batches and use the same experience for many updates
- Rewards and transitions stay fixed; value targets are recomputed
- Q learning is *off-policy*: experience can come from an older or different policy

A deep Q network

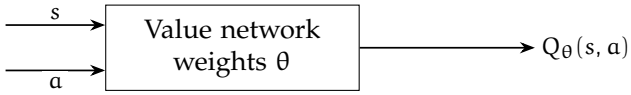


$$y = r + \gamma \max_{a'} Q_{\bar{\theta}}(s', a'), \qquad L(\theta) = \tfrac{1}{2}\big(y - Q_\theta(s, a)\big)^2$$

We hold y fixed when differentiating the loss

θ: online network weights $\quad \bar{\theta}$: target network weights
L: squared-error loss $\quad a_1, \ldots, a_m$: the m discrete actions

Continuous actions



$$a = \arg\max_{a'} Q_\theta(s, a')$$

- The network predicts a value for one state-action pair
- Continuous actions require an optimization each time we act
- Better to add a policy network to produce the action directly

Evaluating a policy

A stochastic policy gives an action distribution:

$$a_t \sim \pi(\cdot \mid s_t)$$

Its state value is

$$V^{\pi}(s) = \mathbb{E}_{\tau \sim p_{\pi}(\mathcal{T}|s)} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

Policy evaluation predicts the return from following a given policy

$\pi(a \mid s)$: probability or density of action a in state s
$V^{\pi}(s)$: expected return starting at s and following π

Policy evaluation: Monte Carlo and TD

Both methods evaluate a *fixed policy* by fitting $V_\phi(s_t)$ to y_t

$$\text{Monte Carlo:} \quad y_t = g_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k$$

$$\text{TD(0):} \quad y_t = r_t + \gamma V_\phi(s_{t+1})$$

- MC waits for the observed return, which can be noisy
- TD bootstraps using its own prediction of future return
- TD depends on the Markov property; MC does not

ϕ: value-network weights T: episode termination time
g_t: observed return from time t k: index of a future time step

Optimizing the policy directly

We want to *maximize* the expected return

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\mathcal{T})}[R(\tau)], \qquad R(\tau) = \sum_{t=0}^{H-1} \gamma^t r_t$$

- The policy network π_θ specifies a distribution over actions
- p_θ is the trajectory distribution induced by π_θ
- REINFORCE estimates a gradient using sampled trajectories

θ: policy weights here J: expected-return objective
$R(\tau)$: discounted return of trajectory τ

REINFORCE

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

- The derivation is not obvious! The return weights the log-probability gradient
- Increase likelihood of actions in trajectories with high return
- We only need experience of the world, with no model

∇_θ: gradient with respect to policy weights log: natural logarithm

The REINFORCE learning loop

1. The current policy generates a batch of trajectories
2. Their returns weight the policy's log-probability gradients
3. A gradient-ascent step updates the policy weights

$$\theta \leftarrow \theta + \alpha \widehat{g}$$

Fresh batches give an unbiased gradient estimate at the current weights, but the estimate can have high variance

$\widehat{g}$: gradient estimate from the sampled batch

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 fixed, action-independent baseline preserves the expected gradient
- If the usual return is 100, returns of 102 and 98 give weights of $+2$ and -2
- A useful baseline predicts the return from the current state

$b_t(s)$: action-independent baseline for state s at time t

Learning the baseline

$$b_\phi(s_t, t) \approx g_t$$

- The current baseline supplies the weights for the policy update
- Squared-error regression then fits the baseline to this batch's returns
- The fitted baseline serves the next batch

Fitting first can make the baseline depend on the very outcomes it is supposed to compare against

$b_\phi(s, t)$: learned baseline with weights ϕ

REINFORCE with a learned baseline

1 Initialize θ, ϕ and choose α, N, H, γ

2 Repeat:

3 Collect N trajectories with π_θ, starting from p_0

4 Compute backward, for each trajectory i: $g_H^{(i)} = 0, \quad g_t^{(i)} = r_t^{(i)} + \gamma g_{t+1}^{(i)}$

5 Form $w_t^{(i)} = g_t^{(i)} - b_\phi(s_t^{(i)}, t)$ before fitting the baseline

6 Hold the weights $w_t^{(i)}$ fixed and compute

$$\widehat{g} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H-1} \gamma^t w_t^{(i)} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)})$$

7 Update $\theta \leftarrow \theta + \alpha \widehat{g}$

8 Fit $b_\phi(s, t)$ to the targets g_t for the next batch

N: batch size i: trajectory index p_0: initial-state distribution

Proximal policy optimization

PPO is generally more effective than basic REINFORCE

1. A learned value baseline estimates how much better an action was than expected
2. A probability ratio lets us reuse a recent batch of experience for several updates
3. Clipping discourages moving too far from the collection policy
4. Estimate actions' advantage over baseline from short rollouts

PPO: proximal policy optimization GAE: generalized advantage estimation

Learning dexterous manipulation



Click the image to play the video

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

PPO and the advantage

Proximal policy optimization (PPO) reuses a recent batch for several policy updates

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

$Q^{\pi}(s, a)$ evaluates taking a in s, then following π
For complete episodes, a simple estimate is

$$\widehat{\mathsf{adv}}_t = g_t - V_{\phi}(s_t)$$

A positive advantage favors that action; we keep the estimates fixed while updating the policy

$Q^{\pi}(s, a)$: action value under π $A^{\pi}(s, a)$: true advantage
$\widehat{\mathsf{adv}}_t$: sampled advantage estimate V_{ϕ}: learned value baseline

Comparing the new and old policy

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

- $\pi_{\theta_{\text{old}}}$ collected the batch; initially $\rho_t = 1$
- The gradient of $\rho_t \widehat{\text{adv}}_t$ initially matches the log-probability update weighted by advantage
- This ratio adjusts action probabilities at recorded states

θ_{old}: weights that collected the batch ρ_t: new-to-old probability ratio

The clipped PPO objective

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

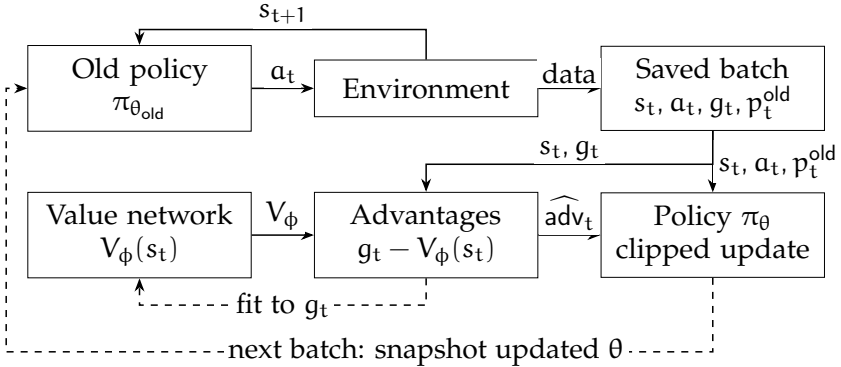
$$\ell_t(\theta) = \min\Big(\rho_t(\theta)\widehat{\mathrm{adv}}_t,\ \mathrm{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{\mathrm{adv}}_t\Big)$$

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

Clipping removes the incentive for a large change in the favored direction

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

The PPO batch cycle

1. Snapshot the policy and collect fresh episodes
2. Compute and save returns, old action probabilities, and advantages
3. Take several policy updates to maximize the clipped objective
4. Fit the value network to the saved return targets
5. Discard the batch and collect fresh experience

PPO is still called on-policy: it reuses each recent batch for a limited time

Generalized advantage estimation

GAE supplies advantage estimates for PPO without waiting for an episode to end, combining rewards with the critic's prediction of what remains

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

$$\widehat{\mathrm{adv}}_t = \sum_{\ell=0}^{K-1} (\gamma\lambda)^\ell \delta_{t+\ell}$$

A true terminal state uses zero next-state value
A collection cutoff uses the critic's prediction

δ_t: one-step TD error K: steps to the segment or episode boundary
ℓ: offset from time t $\lambda \in [0, 1]$: tradeoff between long returns and bootstrapping

The role of the value prediction

- $\lambda = 0$ uses a single TD error
- $\lambda = 1$ over a complete episode gives $g_t - V_\phi(s_t)$
- Intermediate values trade noisy long returns for reliance on predictions

The value network is a *critic*: it helps the policy learn which outcomes were better than expected

PPO pseudocode: complete-episode returns

1 Initialize θ, ϕ and choose $N, H, \gamma, \epsilon, \alpha, M$

2 Repeat:

3 Set $\theta_{\text{old}} \leftarrow \theta$ and collect N complete H-step episodes with $\pi_{\theta_{\text{old}}}$ from p_0

4 Compute $g_H = 0$, then $g_t = r_t + \gamma g_{t+1}$ backward

5 Save $p_t^{\text{old}} = \pi_{\theta_{\text{old}}}(a_t \mid s_t)$ and $\widehat{\text{adv}}_t = g_t - V_\phi(s_t)$

6 Repeat M times, holding p_t^{old} and $\widehat{\text{adv}}_t$ fixed:

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

7 Fit V_ϕ to g_t by minimizing $\sum_{i,t}(V_\phi(s_t^{(i)}) - g_t^{(i)})^2$

8 Discard the batch

M: policy updates per batch i: episode index
Episode indices are suppressed in the return and advantage computations