Yale University

Reinforcement Learning: Markov Decision Processes

S&DS 265 · Introductory Machine Learning · Lecture 20

Zhuoran Yang

Fall 2026

AI usage: figures were generated, and these slides polished, with the help of AI tools.

Outline

Learning Objectives

In this lecture you will learn:

1. Why an acting learner produces its own data, and what breaks as a result;
2. How a Markov decision process represents states, actions, rewards, and dynamics;
3. How value functions summarize long-run consequences through Bellman equations;
4. How learning becomes one optimization problem, $\max_\theta J(\theta)$, over a policy class;
5. How the score trick differentiates that objective in the one-step case, without knowing the environment.

Outline

Motivating Success: AlphaGo Beats Lee Sedol, March 2016

Seoul, March 2016

illustration: Financial Times, credited educational use

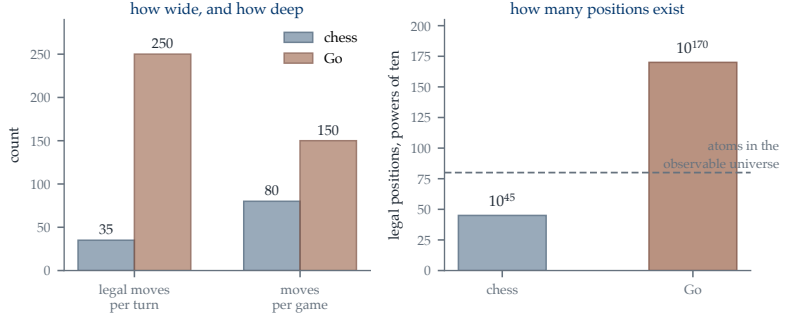
Game 4, move 78

Wesalius, CC BY-SA 4.0

A program beat one of the strongest Go players alive, four games to one. Go was the standing example of a game computers could not play, and expert forecasts in 2015 were still measured in decades. Nobody had told the program how to play.

Photos via Wikimedia Commons, accessed 20 August 2026: Lee Sedol, LG Electronics, CC BY 2.0; AlphaGo hardware, "Immigrant laborer," CC0; Game 4 diagram, Wesalius, CC BY-SA 4.0. Match result: Silver et al. (2016), Nature 529:484–489.

Why Go Was the Hard Case

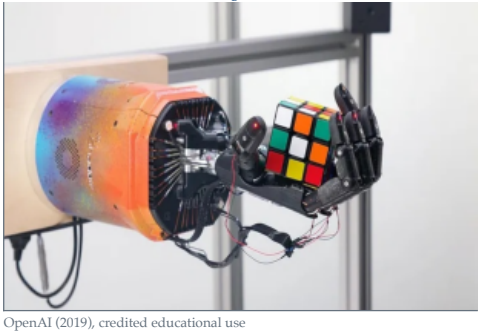


Chess fell to search: look ahead, score the positions, pick the best line. Go's tree is seven times wider and twice as deep, and it has 10^{170} legal positions — more than 10^{90} times the atoms in the observable universe. Nobody enumerates that, and nobody could write down a position score either. Both had to be learned.

Breadth and depth as quoted in Silver et al. (2016), Nature 529:484–489. Legal Go positions: Tromp (2016), exact enumeration; chess positions and the atom count are standard order-of-magnitude estimates.

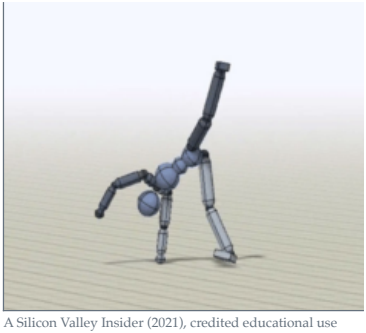
Success of RL in Robotics

a hand solving a Rubik's cube



OpenAI (2019), credited educational use

a simulated humanoid

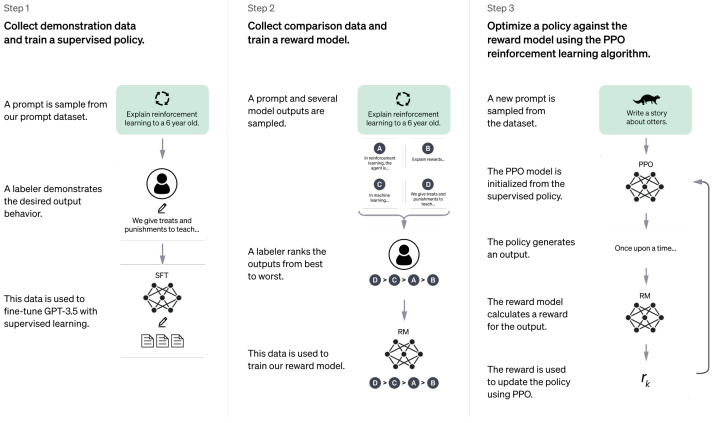


A Silicon Valley Insider (2021), credited educational use

A hand reorients a Rubik's cube in its palm; a simulated body learns to move without ever being shown how. Nobody wrote down the correct joint torques. There is no label to imitate — only an outcome that can be scored.

Left: OpenAI (2019), "Solving Rubik's Cube with a Robot Hand," arXiv:1910.07113, Figure 1 excerpt. Right: A Silicon Valley Insider (2021), accessed 20 August 2026. Both are credited educational excerpts.

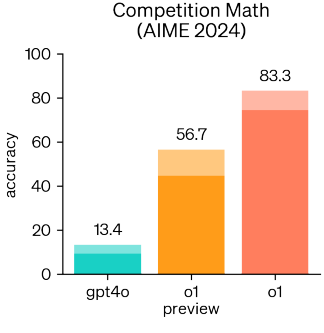
Training ChatGPT: Reward from Human Preference



People rank several model answers, a reward model is fitted to those rankings, and the language model is optimized against that reward by RL.

OpenAI, "Introducing ChatGPT," 30 November 2022; also Figure 2 of Ouyang et al. (2022). Credited educational excerpt.

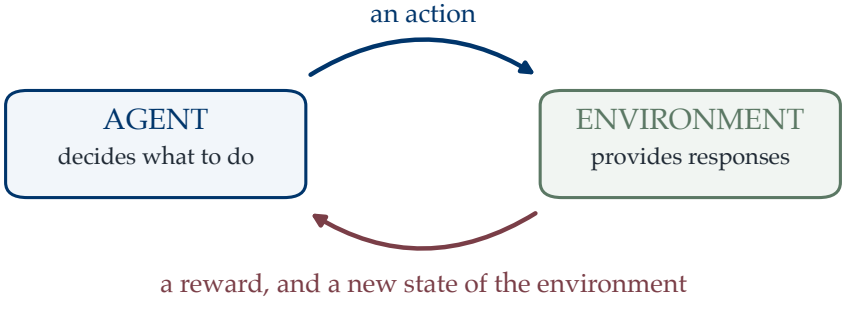
Training Reasoning Models: Reward from a Checker



Accuracy on the 2024 AIME: 13.4% for GPT-4o against 83.3% for o1. The gain came not from more labels but from training against a checker that grades the final answer. Lecture 23 asks where each reward comes from.

OpenAI, "Learning to Reason with LLMs," 12 September 2024. Solid bars are single-sample accuracy; the shaded extension is a majority vote over 64 samples.

What Is Reinforcement Learning?



A learner that improves from its own experience: it acts, sees what happens, and adjusts — with nobody supplying the right answer.

Why That Is Worth the Trouble

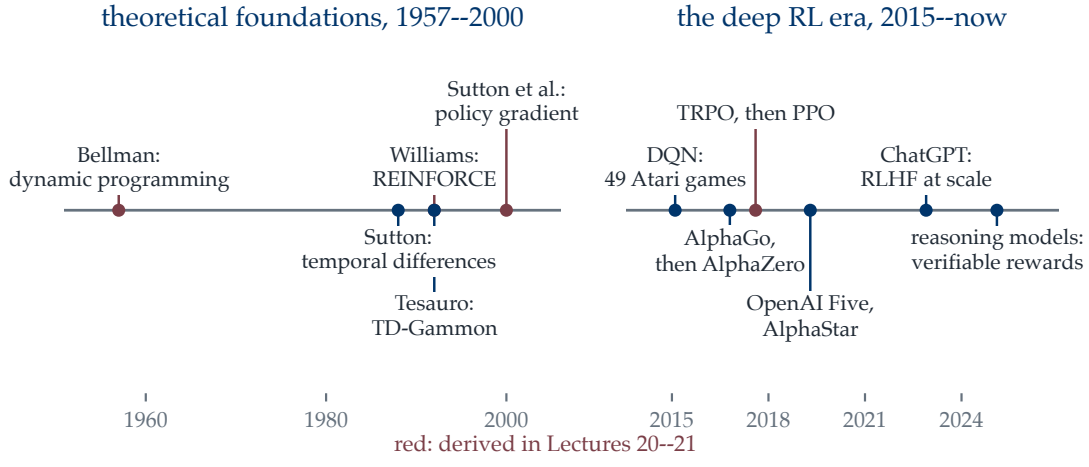
- **No sophisticated labels**. Nobody says which move, torque, or token is correct — only whether the outcome was good.
- **Can exceed its teachers**. Imitation is capped by the demonstrator; searching for the optimal strategy is not.
- **Learn from real experiences**. Learners from real experience data generated by itself. Deployed to the same or a similar environment.
- **One recipe, many domains**. Boards, robots, and language share the same problem structure and the same algorithms.

Why That Is Worth the Trouble

- **No sophisticated labels**. Nobody says which move, torque, or token is correct — only whether the outcome was good.
- **Can exceed its teachers**. Imitation is capped by the demonstrator; searching for the optimal strategy is not.
- **Learn from real experiences**. Learners from real experience data generated by itself. Deployed to the same or a similar environment.
- **One recipe, many domains**. Boards, robots, and language share the same problem structure and the same algorithms.

The price is that the learner generates its own training data by acting — which is the difficulty this lecture formalizes.

A Short History of Reinforcement Learning



Dates are publication or event dates, listed in the figure registry.

Outline

Problem Setup

An environment is a **Markov decision process** $(\mathcal{S}, \mathcal{A}, P, r, \mu, \gamma)$, and behaviour is a policy $\pi \in \Pi$:

$$s_0 \sim \mu, \qquad a_t \sim \pi(\cdot \mid s_t), \qquad s_{t+1} \sim P(\cdot \mid s_t, a_t).$$

Problem Setup

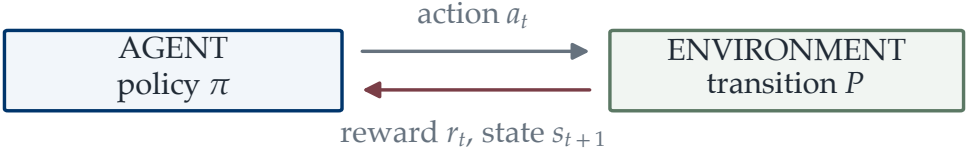
An environment is a **Markov decision process** $(\mathcal{S}, \mathcal{A}, P, r, \mu, \gamma)$, and behaviour is a policy $\pi \in \Pi$:

$$s_0 \sim \mu, \qquad a_t \sim \pi(\cdot \mid s_t), \qquad s_{t+1} \sim P(\cdot \mid s_t, a_t).$$

- **Target**: the expected discounted return $J(\pi) = \mathbb{E}_\pi[\sum_{t \geq 0} \gamma^t \cdot r(s_t, a_t)]$.
- **Model class**: a policy family Π, later a neural network π_θ.
- **Estimate**: $\pi^\star \in \arg\max_{\pi \in \Pi} J(\pi)$, from interaction data.

The expectation and the maximization are over the same π — which is exactly what makes this different.

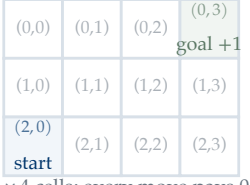
The Agent–Environment Loop



The arrow from action back to state is what supervised learning lacks.

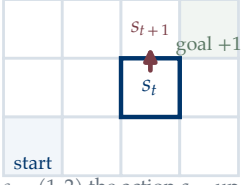
Toy Example: Gridworld

the gridworld



3×4 cells; every move pays 0 until the goal, which pays +1

one transition (s_t, a_t, r_t, s_{t+1})



from $s_t = (1,2)$ the action $a_t =$ up pays $r_t = 0$: the payoff comes two moves later

the two logged episodes

two different routes, both reaching the goal; the policy chose which rows got logged

The agent starts at $(2,0)$ and moves one cell at a time. Every move pays 0; only reaching the goal pays +1. Small enough to solve exactly, and it still has the same two difficulties as Go: credit arrives late, and the agent's own choices decide which rows get logged.

Grid and paths drawn from SDS265/data/rl/gridworld-logged-trajectories.csv; original course simulation.

The Logged Episodes as a Table

The same two episodes, written out the way the learner actually receives them.

episode	t	state	action	reward	next state	terminal
0	0	$(2,0)$	right	0	$(2,1)$	no
0	1	$(2,1)$	up	0	$(1,1)$	no
0	2	$(1,1)$	right	0	$(1,2)$	no
0	3	$(1,2)$	up	0	$(0,2)$	no
0	4	$(0,2)$	right	1	$(0,3)$	yes

Only the last move pays, as in Go.

Source: First episode of the gridworld log used in this lecture's notebook; original course simulation.

Reading One Transition

At time $t = 3$ of episode 0,

$$s_t = (1,2), \qquad a_t = \text{up}, \qquad r_t = 0, \qquad s_{t+1} = (0,2).$$

Reading One Transition

At time $t = 3$ of episode 0,

$$s_t = (1, 2), \qquad a_t = \text{up}, \qquad r_t = 0, \qquad s_{t+1} = (0, 2).$$

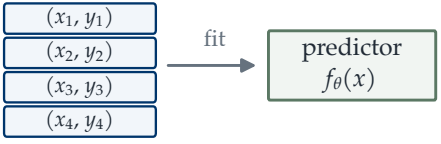
Two features have no analogue in supervised learning.

- The reward is 0, yet the move was good: it placed the agent one step from the goal. Credit arrives late.
- The row exists only because earlier actions led here. It is not an independent input–label pair.

Fixed Data against Induced Data

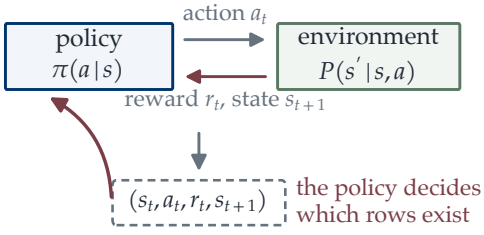
supervised learning

a fixed sample, drawn once



the rows are there before fitting starts,
and the predictor cannot change them

reinforcement learning



Contrasting an exogenous sample with a policy-induced trajectory.

Two Difficulties at Once

- **Delayed credit**: the reward that matters may arrive many steps after the action responsible for it.
- **Induced distribution**: changing the policy changes the data, so a quantity fitted on old data may not describe the new policy.

Two Difficulties at Once

- **Delayed credit**: the reward that matters may arrive many steps after the action responsible for it.
- **Induced distribution**: changing the policy changes the data, so a quantity fitted on old data may not describe the new policy.

Today: value functions and Bellman equations handle the first, and set up the objective $J(\pi)$ we will optimize. The second difficulty is the subject of Lecture 21.

Markov Decision Process

A discounted MDP is $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \mu, \gamma)$.

$\mathcal{S}$	states	$\mathcal{A}$	actions
$P(s' \mid s, a)$	transition law	$r(s, a)$	expected reward
μ	initial-state law	$\gamma \in [0, 1)$	discount factor

The environment supplies P and r; the learner supplies $\pi(a \mid s)$. Only the policy is chosen — everything else is the world.

Markov State Assumption

The whole framework rests on

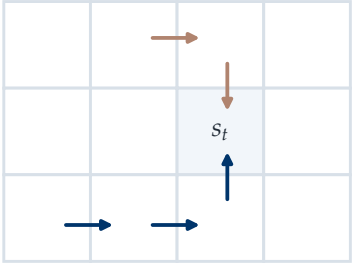
$$\mathbb{P}(s_{t+1}, r_t \mid s_0, a_0, \dots, s_t, a_t) = \mathbb{P}(s_{t+1}, r_t \mid s_t, a_t) :$$

the current state is a sufficient summary of the past. In the gridworld, s_t is the cell the agent occupies. Two agents standing in the same cell face the same future, however differently they got there.

Visualizing the Markov Assumption

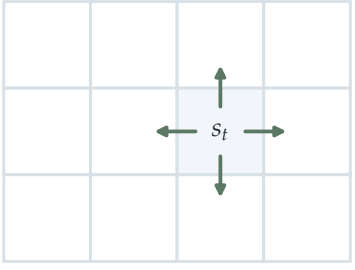
how the agent got here

two different histories



one arrived the long way, one the short way

where it can go next



$\mathbb{P}(s_{t+1} \mid s_t, a_t)$ is the same either way

Left: two histories, one long and one short, ending in the same cell. Right: the moves available from that cell. The route taken to arrive does not appear in the transition law — which is exactly what it means to say the position is a Markov state.

When the Assumption Fails

A position is Markov here only because nothing else about the past matters. It fails whenever the state omits something that does:

- A single screen frame omits velocity;
- A dialogue summary omits a promise made earlier;
- An account record omits a pending charge.

Choosing the state is modelling, not bookkeeping. Every equation below is valid only for a state that keeps the relevant history.

Stationarity

A second standing assumption: P and r do not depend on t. The rule that maps (s, a) to a next state is the same at step 3 and at step 3000.

$$P(s' \mid s, a) \text{ and } r(s, a), \text{ not } P_t(s' \mid s, a) \text{ and } r_t(s, a)$$

This is what makes a stationary policy $\pi(a \mid s)$ — one that looks only at the state, never at the clock — enough to behave optimally. Without it we would have to learn a different policy for every time step. The gridworld qualifies: moving up from $(1, 2)$ does the same thing whenever it is tried.

A Policy Is a Conditional Distribution

A **stationary policy** maps each state to a distribution over actions,

$$\pi(\cdot \mid s) \in \Delta(\mathcal{A}) \qquad \text{for every } s \in \mathcal{S}.$$

It has the same type as a classifier: input a state, output class probabilities.

The action is the label for policy. The difference that this label is executed in the environment and changes the state.

Deterministic policies $a = \pi(s)$ are the special case that puts all mass on one action. Randomness will matter: it is what makes J differentiable once we optimize it.

Trajectory Distribution

A policy and the dynamics together define a distribution over whole paths $\tau = (s_0, a_0, s_1, a_1, \ldots)$:

$$p_\pi(\tau) = \mu(s_0) \prod_{t \geq 0} \pi(a_t \mid s_t) \cdot P(s_{t+1} \mid s_t, a_t).$$

Trajectory Distribution

A policy and the dynamics together define a distribution over whole paths $\tau = (s_0, a_0, s_1, a_1, \ldots)$:

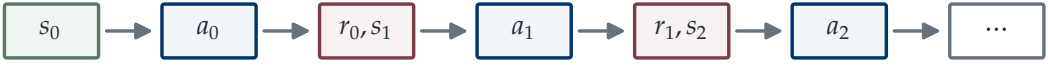
$$p_\pi(\tau) = \mu(s_0) \prod_{t \geq 0} \pi(a_t \mid s_t) \cdot P(s_{t+1} \mid s_t, a_t).$$

Taking the ratio for two policies cancels every environment factor:

$$\frac{p_{\pi'}(\tau)}{p_\pi(\tau)} = \prod_{t \geq 0} \frac{\pi'(a_t \mid s_t)}{\pi(a_t \mid s_t)}.$$

The learner controls exactly one factor per step — and a small change repeated many times can move the path distribution a great deal.

States, Actions, and Discounted Rewards



$$G_0 = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$$

Return and Policy Objective

The return from time t discounts later rewards geometrically,

$$G_t = \sum_{k \geq 0} \gamma^k \cdot r_{t+k}, \qquad J(\pi) = \mathbb{E}_{s_0 \sim \mu,\, a_t \sim \pi,\, s_{t+1} \sim P}[G_0].$$

Discounting does two things: it keeps the sum finite for an endless interaction, and it says that a reward now is worth more than the same reward later. The effective horizon is about $1/(1-\gamma)$ steps.

At $\gamma = 0.99$ that is roughly 100 steps; at $\gamma = 0.9$, ten. The discount is a modelling choice with a visible consequence. Appendix A.0 derives the $1/(1-\gamma)$.

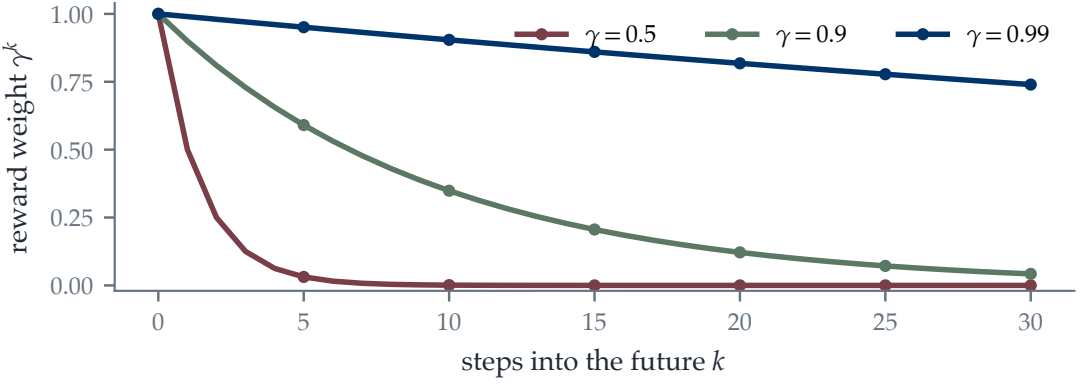
The Infinite Horizon Is an Approximation

Real environments stop. An Atari episode ends, a robot's battery runs out, a conversation closes. The horizon is finite — it is just long, and usually not known in advance.

$$\text{long finite horizon} \;\approx\; \text{infinite horizon, discounted}$$

Writing $\sum_{t \geq 0} \gamma^t \cdot r_t$ is a modelling convenience: it removes T from the problem, so one stationary policy and one value function suffice instead of a different pair for every remaining step. The price is that γ now sets how far ahead we are willing to look. Choose γ so that $1/(1-\gamma)$ is comparable to the episode length you actually care about.

Discounting and Effective Horizon



Exact reward weights for three discount factors.

Outline

State and Action Values

Fix a policy π. Its two value functions summarize the future:

$$V^\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s], \qquad Q^\pi(s,a) = \mathbb{E}_\pi[G_t \mid s_t = s,\, a_t = a].$$

V^π follows the policy from the start; Q^π forces one action first and follows the policy afterwards. Both are properties of one fixed policy. Nothing here is about the best policy yet — these numbers describe the behaviour we currently have.

Averaging V^π over the initial distribution recovers the objective: $J(\pi) = \mathbb{E}_{s_0 \sim \mu} V^\pi(s_0)$.

The Advantage

Their difference is the **advantage**,

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

which answers the only question a learner can act on: is this action better than what I usually do here? Under the policy's own action distribution it averages to zero,

$$\mathbb{E}_{a\sim\pi(\cdot|s)}[A^\pi(s,a)] = 0 \qquad \text{for every } s.$$

So advantages are deviations from a habit, not absolute quality. Some action in every state has $A^\pi \geq 0$; improving means moving probability towards it.

Source: One line of algebra in Appendix A.1.

The Return Repeats Itself

Split the first reward off the series:

$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \cdots = r_t + \gamma \cdot \underbrace{\{r_{t+1} + \gamma \cdot r_{t+2} + \cdots\}}_{G_{t+1}}.$$

An infinite sum has become one reward plus a discounted copy of the same object. Every method in these two lectures exploits that recursion.

This is what makes delayed credit tractable: a value one step ahead stands in for the entire future.

Bellman Equation for Policy Evaluation

Take expectations of $G_t = r_t + \gamma \cdot G_{t+1}$ given $s_t = s$, averaging over the action drawn by π and the next state drawn by P:

$$V^{\pi}(s) = \sum_{a} \pi(a \mid s) \Big\{ r(s,a) + \gamma \cdot \sum_{s'} P(s' \mid s,a) \cdot V^{\pi}(s') \Big\}$$

Bellman Equation for Policy Evaluation

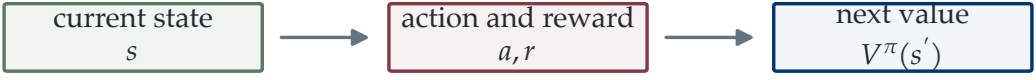
Take expectations of $G_t = r_t + \gamma \cdot G_{t+1}$ given $s_t = s$, averaging over the action drawn by π and the next state drawn by P:

$$V^\pi(s) = \sum_a \pi(a \mid s) \Big\{ r(s,a) + \gamma \cdot \sum_{s'} P(s' \mid s,a) \cdot V^\pi(s') \Big\}$$

This is the **Bellman equation for π**: a consistency condition the value of the policy we already have must satisfy.

Source: Iterated expectation, written out in Appendix A.2.

Bellman Backup



$$V^{\pi}(s) = \mathrm{E}_{\pi}[r + \gamma V^{\pi}(s') \mid s]$$

What the Equation Does and Does Not Say

The unknown function appears on both sides. V^π is characterized as a **fixed point**, not by an explicit formula.

- It evaluates one policy. It does not choose one.
- There is no maximum in it: the action is averaged with the weights $\pi(a \mid s)$ the policy actually uses.
- Every quantity is an expectation, so it can be estimated from sampled transitions when P is unknown.

In this course the Bellman equation is only ever used for evaluation. Choosing a better policy will be done by optimizing J directly.

The Same Equation in Action Values

Forcing the first action and then following π gives the pair

$$Q^\pi(s,a) = r(s,a) + \gamma \cdot \sum_{s'} P(s' \mid s,a) \cdot V^\pi(s'), \qquad V^\pi(s') = \sum_{a'} \pi(a' \mid s') \cdot Q^\pi(s',a').$$

Substituting the second into the first closes the recursion in Q^π alone. This is the form a learner can act on: it ranks the actions available in a state, not just the states.

The two together also give the one-step form of the advantage, $A^\pi(s,a) = \mathbb{E}[\, r_t + \gamma \cdot V^\pi(s_{t+1}) - V^\pi(s_t) \mid s_t, a_t \,]$, which Lecture 21 estimates from one transition.

Policy Evaluation with Finitely Many States

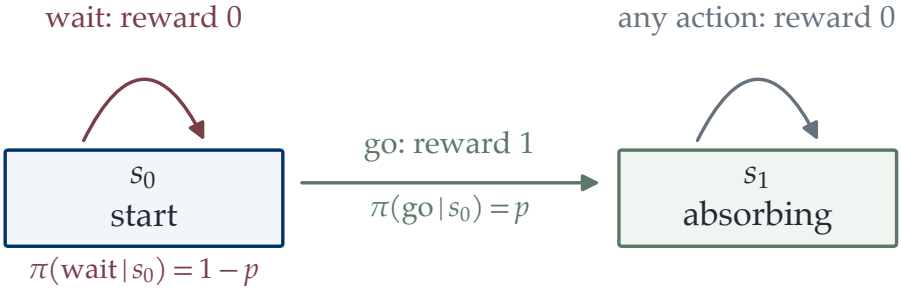
Write $r^\pi(s) = \sum_a \pi(a \mid s) \cdot r(s,a)$ and $P^\pi(s,s') = \sum_a \pi(a \mid s) \cdot P(s' \mid s,a)$. The Bellman equation is then linear:

$$V^\pi = r^\pi + \gamma P^\pi V^\pi \qquad \Longrightarrow \qquad V^\pi = (I - \gamma P^\pi)^{-1} r^\pi.$$

The inverse exists because $\gamma < 1$ makes γP^π a contraction: evaluating a fixed policy is a solved linear-algebra problem.

The formula explains, but nobody inverts a matrix with 10^{10} rows. Lecture 21 replaces it with least squares on sampled transitions.

A Two-State Example



Original course example with $\gamma = 0.9$; the baseline policy always waits ($p = 0$) and the candidate sets $p > 0$. Used again in Lecture 21.

Evaluating Two Policies in the Two-State MDP

The waiting policy never leaves s_0 and never scores:

$$V^{\pi}(s_0) = 0 + 0.9\, V^{\pi}(s_0) \implies V^{\pi}(s_0) = 0.$$

Evaluating Two Policies in the Two-State MDP

The waiting policy never leaves s_0 and never scores:

$$V^{\pi}(s_0) = 0 + 0.9\, V^{\pi}(s_0) \implies V^{\pi}(s_0) = 0.$$

The going policy takes the paying action with probability p at every visit, so

$$V^{\pi'}(s_0) = p \cdot 1 + (1-p)\,\gamma\, V^{\pi'}(s_0) \implies V^{\pi'}(s_0) = \frac{p}{1-\gamma(1-p)}.$$

At $p = 0.5$ and $\gamma = 0.9$ this is $0.5/0.55 = 10/11 \approx 0.909$.

Two policies, same environment, values computed from the same one-line equation. The remaining question is how to search over π.

Solving the Bellman Equation by Iteration

Read the right-hand side as an operator T^π acting on a value function:

$$(T^\pi V)(s) = \sum_a \pi(a \mid s)\Big\{r(s,a) + \gamma \cdot \sum_{s'} P(s' \mid s,a) \cdot V(s')\Big\},$$

so the Bellman equation says $V^\pi = T^\pi V^\pi$: the value function is the fixed point of T^π.

Solving the Bellman Equation by Iteration

Read the right-hand side as an operator T^π acting on a value function:

$$(T^\pi V)(s) = \sum_a \pi(a \mid s)\Big\{r(s,a) + \gamma \cdot \sum_{s'} P(s' \mid s,a) \cdot V(s')\Big\},$$

so the Bellman equation says $V^\pi = T^\pi V^\pi$: the value function is the fixed point of T^π.

Because $\gamma < 1$, the operator is a **contraction**:

$$\|T^\pi V - T^\pi W\|_\infty \le \gamma \cdot \|V - W\|_\infty.$$

So iterating $V_{k+1} = T^\pi V_k$ from any starting point converges to V^π, and the error falls by a factor γ per sweep.

Why Iteration Is Not Enough

A sweep of $V_{k+1} = T^{\pi} V_k$ touches every state, and needs the model P and r to compute the inner sum. Neither survives contact with a real problem:

- Go has more than 10^{170} positions; no array is that long;
- In a robot or a dialogue the state is continuous, so there is nothing to enumerate at all;
- P is unknown — we only get to sample from it.

Why Iteration Is Not Enough

A sweep of $V_{k+1} = T^{\pi} V_k$ touches every state, and needs the model P and r to compute the inner sum. Neither survives contact with a real problem:

- Go has more than 10^{170} positions; no array is that long;
- In a robot or a dialogue the state is continuous, so there is nothing to enumerate at all;
- P is unknown — we only get to sample from it.

The fix is the one this course has used all semester: replace a value at every state by a function fitted from data, and replace an exact expectation by a regression on samples.

Fitted Policy Evaluation

Collect n transitions, taking the action from the current policy:

$$s_i \sim \nu, \qquad a_i \sim \pi(\cdot \mid s_i), \qquad r_i,\ s_i' \sim P(\cdot \mid s_i, a_i).$$

Fitted Policy Evaluation

Collect n transitions, taking the action from the current policy:

$$s_i \sim \nu, \qquad a_i \sim \pi(\cdot \mid s_i), \qquad r_i,\ s_i' \sim P(\cdot \mid s_i, a_i).$$

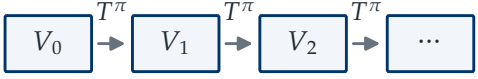
Given the current estimate $\widehat{V}_k$, form a regression target for each transition and fit the next estimate:

$$y_i = r_i + \gamma \cdot \widehat{V}_k(s_i'), \qquad \widehat{V}_{k+1} = \arg\min_{V \in \mathcal{U}} \frac{1}{n} \sum_{i=1}^{n} (V(s_i) - y_i)^2.$$

Only sampled transitions appear — no P, no sum over states. Each round is an ordinary least-squares fit, with $\widehat{V}_k$ supplying the labels. The action must come from π, because T^π averages over π's actions. Using someone else's actions evaluates someone else's policy.

Exact Backups against Fitted Backups

exact backups

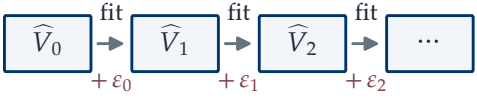


needs P and r, and one value per state

$$\|V_{k+1} - V^\pi\|_\infty \leq \gamma \|V_k - V^\pi\|_\infty$$

the distance to V^π shrinks every sweep

fitted backups, from samples



$s_i \sim \nu$ $\quad a_i \sim \pi(\cdot \mid s_i)$ $\quad r_i,\, s_i' \sim P$

target $y_i = r_i + \gamma \widehat{V}_k(s_i')$, then least squares on (s_i, y_i)

each sweep adds a regression error

Each sweep on the right injects a regression error ε_k, from finite data and from a function class that cannot fit the target exactly. The iterates reach a neighbourhood of V^π, not V^π.

What Carries into Deep Reinforcement Learning

Nothing above required $\mathcal{U}$ to be simple. Taking $\mathcal{U}$ to be a neural network gives the value-estimation step used in Lecture 21:

$$\text{sample with } \pi \;\rightarrow\; \text{build targets } r + \gamma \cdot \widehat{V}_k(s') \;\rightarrow\; \text{fit by SGD} \;\rightarrow\; \text{repeat}$$

What Carries into Deep Reinforcement Learning

Nothing above required $\mathcal{U}$ to be simple. Taking $\mathcal{U}$ to be a neural network gives the value-estimation step used in Lecture 21:

$$\text{sample with } \pi \;\rightarrow\; \text{build targets } r + \gamma \cdot \widehat{V}_k(s') \;\rightarrow\; \text{fit by SGD} \;\rightarrow\; \text{repeat}$$

The targets move as $\widehat{V}_k$ moves, so this is regression against a shifting label. Lecture 21 shows the devices that keep it stable.

Outline

From Evaluating to Choosing

Bellman equations tell us what the current policy is worth. They do not say which policy to run.

If we write the policy as a parametric family, can we simply do gradient ascent — the tool the whole course has been using?

The Optimization Problem

Restrict attention to a parametric family $\{\pi_\theta : \theta \in \mathbb{R}^d\}$ and write

$$\max_{\theta \in \mathbb{R}^d} J(\theta) = \mathbb{E}_{\tau \sim p_{\pi_\theta}}\Big[\sum_{t \geq 0} \gamma^t \cdot r(s_t, a_t)\Big]$$

Compare with every earlier lecture:

- Model class $\{\pi_\theta\}$, exactly as before;
- Objective $J(\theta)$ — but the expectation is over a distribution p_{π_θ} that moves with θ;
- Algorithm $\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$, exactly as before, if we can compute the gradient.

Policy Families

In every case a network produces **logits** $f_\theta(s) \in \mathbb{R}^{|\mathcal{A}|}$ and the policy is a softmax:

$$\pi_\theta(a \mid s) = \frac{\exp\{f_\theta(s)_a\}}{\sum_{a'} \exp\{f_\theta(s)_{a'}\}}.$$

- **Tabular softmax**: one parameter per state–action pair.
- **Neural policy**: f_θ is an MLP or a convolutional network on the state.
- **Language model**: state = the tokens so far, action = the next token, f_θ = a transformer. A chat model is a policy.

This is the same softmax layer as multiclass classification in Lecture 8. Only the training signal is different.

Why the Usual Gradient Argument Breaks

In supervised learning the objective is an average over a fixed sample, so the gradient passes straight through to the model:

$$\nabla_\theta \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \ell(f_\theta(x_i), y_i).$$

Why the Usual Gradient Argument Breaks

In supervised learning the objective is an average over a fixed sample, so the gradient passes straight through to the model:

$$\nabla_\theta \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \ell(f_\theta(x_i), y_i).$$

Here the sample is drawn from π_θ itself:

$$J(\theta) = \int G(\tau)\, p_{\pi_\theta}(\tau)\, d\tau.$$

Changing θ changes which trajectories appear, not just how they are scored.

The VAE had the same problem: an expectation over $q_\phi(z \mid x)$, which carries the parameters. The fix there was the reparameterization trick, and it applies here whenever the actions are continuous — see Appendix B.1.

Three Obstacles in One Objective

- The environment P is unknown, so we cannot write J in closed form and differentiate it.
- Even if P were known, the reward and the dynamics are usually not differentiable — a game engine, a simulator, a grader.
- The distribution being averaged over depends on θ, which is exactly the term we have no formula for.

What we can do is run the policy and observe the reward. Everything below is built from that one capability.

The Obvious Fallback, and What It Costs

Estimate each partial derivative by a finite difference:

$$\frac{\partial J}{\partial \theta_j} \approx \frac{J(\theta + \varepsilon e_j) - J(\theta)}{\varepsilon}.$$

The Obvious Fallback, and What It Costs

Estimate each partial derivative by a finite difference:

$$\frac{\partial J}{\partial \theta_j} \approx \frac{J(\theta + \varepsilon e_j) - J(\theta)}{\varepsilon}.$$

- d parameters need $d + 1$ separate evaluations of J;
- Each J is itself an expectation, so each needs many rollouts;
- Modern policies have d in the billions.

Correct, but costly: the rollout budget scales with the number of parameters.
Derivative-free methods remain a live research area and do work at moderate d.
Lecture 21 keeps the idea without paying per parameter.

Plan: Solve the Easiest Case First

Set the horizon to one. The agent sees a state, takes a single action, collects a reward, and stops:

$$J(\theta) = \mathbb{E}_{a \sim \pi_\theta}[r(a)].$$

Everything hard about reinforcement learning is switched off: no transitions, no delayed credit, no discounting. One difficulty remains — the distribution still depends on θ.

If we can differentiate this, we have the tool. This problem is called a bandit.

Outline

The Bandit Problem

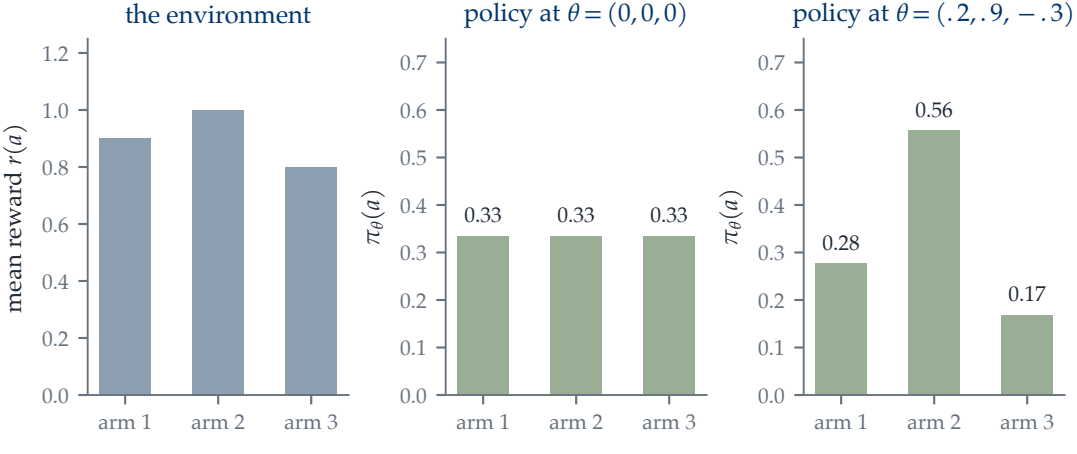
There are K actions. Choosing a returns a reward with unknown mean $r(a)$, and the episode ends. A policy is now just a distribution $\pi_\theta \in \Delta(\mathcal{A})$, and

$$J(\theta) = \mathbb{E}_{a \sim \pi_\theta}[r(a)] = \sum_a \pi_\theta(a) \cdot r(a).$$

Intuitively, this models the case where we have K options in front of us — say K candidate answers — and picking one earns it a grade. There is no next state to worry about, so the only question left is which option to favour.

The name comes from a row of slot machines — "one-armed bandits" — each with an unknown payout.

A Three-Armed Bandit and a Softmax Policy



Original course example: fixed arm means and the softmax policy at $\theta = (0, 0, 0)$ and $\theta = (0.2, 0.9, -0.3)$.

The Gradient of the Bandit Objective

The objective is a finite sum, so differentiating is immediate:

$$\nabla_\theta J(\theta) = \sum_a \nabla_\theta \pi_\theta(a)\; r(a).$$

The Gradient of the Bandit Objective

The objective is a finite sum, so differentiating is immediate:

$$\nabla_\theta J(\theta) = \sum_a \nabla_\theta \pi_\theta(a)\; r(a).$$

The sum runs over every action, and asks for $r(a)$ at each one. In a round we play one arm and see one reward; the other arms' rewards are never revealed.

So we rewrite $\nabla_\theta \pi_\theta$ using the score trick, which turns the sum into an expectation over the actions we actually take.

The Score Trick

derive

For any positive $\pi_\theta(a)$, the chain rule for log gives

$$\nabla_\theta \log \pi_\theta(a) = \frac{\nabla_\theta \pi_\theta(a)}{\pi_\theta(a)} \qquad \Longleftrightarrow \qquad \nabla_\theta \pi_\theta(a) = \pi_\theta(a) \cdot \nabla_\theta \log \pi_\theta(a).$$

The Score Trick

derive

For any positive $\pi_\theta(a)$, the chain rule for log gives

$$\nabla_\theta \log \pi_\theta(a) = \frac{\nabla_\theta \pi_\theta(a)}{\pi_\theta(a)} \qquad \Longleftrightarrow \qquad \nabla_\theta \pi_\theta(a) = \pi_\theta(a) \cdot \nabla_\theta \log \pi_\theta(a).$$

Substituting into the sum turns it back into an expectation:

$$\nabla_\theta J(\theta) = \sum_a \pi_\theta(a) \cdot \nabla_\theta \log \pi_\theta(a) \cdot r(a) = \boxed{\mathbb{E}_{a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a) \cdot r(a)]}$$

$\nabla_\theta \log \pi_\theta(a)$ is called the **score**. The identity is one line, and it is the whole reason policy gradients exist.

The REINFORCE Algorithm

The gradient is now an expectation under π_θ, needing r only at the action drawn — and an expectation under a distribution we can sample from is a quantity we can estimate by averaging.

The REINFORCE Algorithm

The gradient is now an expectation under π_θ, needing r only at the action drawn — and an expectation under a distribution we can sample from is a quantity we can estimate by averaging.

Draw $a_1, \dots, a_n \sim \pi_\theta$ and form

$$\hat{g} = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \log \pi_\theta(a_i) \cdot r(a_i), \qquad \mathbb{E}[\hat{g}] = \nabla_\theta J(\theta).$$

An unbiased gradient estimate from played actions only. This estimator is called REINFORCE.

Reading the Estimator

Recall from Lecture 8 that $-\nabla_\theta \log \pi_\theta(a^*)$ is exactly the cross-entropy gradient for the label a^*.

supervised classification	bandit policy gradient
a labelled action a^* is given	an action a is sampled from the model
push $\log \pi_\theta(a^*)$ up	push $\log \pi_\theta(a)$ up by $r(a)$
every example weighted equally	each example weighted by its reward

Policy gradient is weighted maximum likelihood on the model's own samples. Do more of what paid; do less of what did not.

The Softmax Score, in Closed Form

derive

With $\pi_\theta(a) = e^{\theta_a} / \sum_{a'} e^{\theta_{a'}}$,

$$\log \pi_\theta(a) = \theta_a - \log \sum_{a'} e^{\theta_{a'}},$$

so differentiating in θ_j gives

$$\frac{\partial}{\partial \theta_j} \log \pi_\theta(a) = \mathbf{1}\{j = a\} - \pi_\theta(j), \qquad \text{that is} \qquad \nabla_\theta \log \pi_\theta(a) = e_a - \pi_\theta.$$

The familiar "one-hot minus predicted probabilities". The update raises the chosen arm's logit and lowers all others in proportion to their current probability.

One Numerical Update

Three arms, $\theta = (0, 0, 0)$, so $\pi_\theta = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. Suppose we sample $a = 2$ and observe $r = 1$. Then

$$\hat{g} = r\,(e_2 - \pi_\theta) = 1 \cdot (-\tfrac{1}{3},\ \tfrac{2}{3},\ -\tfrac{1}{3}).$$

With step size $\eta = 0.3$,

$$\theta \leftarrow (-0.10,\ 0.20,\ -0.10), \qquad \pi_\theta = (0.30,\ 0.41,\ 0.30).$$

Arm 2 was played and paid, so its probability rose from 0.33 to 0.41. No knowledge of the environment was used.

Every Weight Is Positive

With $r = (0.9, 1.0, 0.8)$ every weight $r(a)$ is positive, so every update raises whichever arm happened to be drawn. Rewards are random too, so the weight varies even for a fixed arm.

Every Weight Is Positive

With $r = (0.9, 1.0, 0.8)$ every weight $r(a)$ is positive, so every update raises whichever arm happened to be drawn. Rewards are random too, so the weight varies even for a fixed arm.

The estimator is still unbiased: good arms are drawn more often, so the ranking is right on average. But a single $\hat{g}$ says more about which arm was sampled than about how good it was.

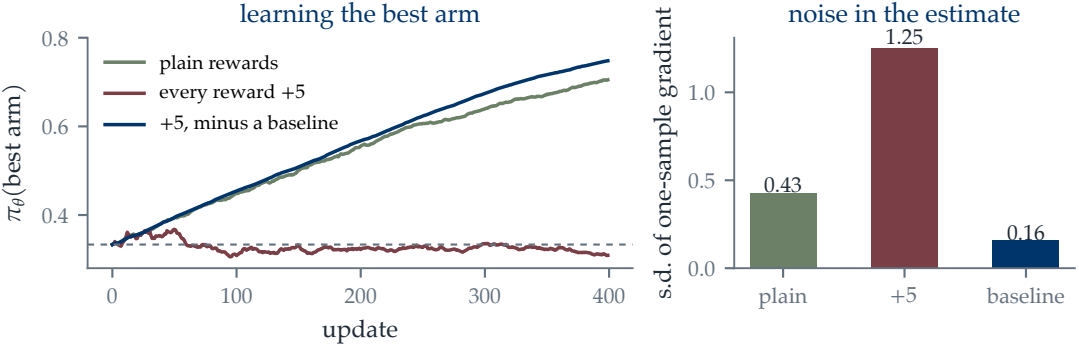
Every Weight Is Positive

With $r = (0.9, 1.0, 0.8)$ every weight $r(a)$ is positive, so every update raises whichever arm happened to be drawn. Rewards are random too, so the weight varies even for a fixed arm.

The estimator is still unbiased: good arms are drawn more often, so the ranking is right on average. But a single $\hat{g}$ says more about which arm was sampled than about how good it was.

Adding a constant to every reward changes nothing about the problem — the same arm is still best — yet it inflates every weight. The next slide runs exactly that experiment.

Simulating the Effect of a Constant Offset



The same bandit three ways. Adding 5 to every arm leaves the best arm unchanged, yet the run stalls at 1/3: the shared offset swamps the differences between arms. Subtracting the running mean reward restores it.

Original course simulation, seed 26520; $r = (0.9, 1.0, 0.8)$ with $\mathcal{N}(0, 0.3^2)$ noise, $\eta = 0.1$, 400 updates, 200 repetitions.

Subtracting a Baseline Costs Nothing

derive

For any constant b that does not depend on the action,

$$\mathbb{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a)\, b] = b \sum_a \pi_\theta(a) \frac{\nabla_\theta \pi_\theta(a)}{\pi_\theta(a)} = b \nabla_\theta \sum_a \pi_\theta(a) = b \nabla_\theta 1 = 0.$$

Subtracting a Baseline Costs Nothing

derive

For any constant b that does not depend on the action,

$$\mathbb{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a)\, b] = b \sum_a \pi_\theta(a) \frac{\nabla_\theta \pi_\theta(a)}{\pi_\theta(a)} = b\, \nabla_\theta \sum_a \pi_\theta(a) = b\, \nabla_\theta 1 = 0.$$

So subtracting it leaves the gradient unchanged:

$$\nabla_\theta J(\theta) = \mathbb{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a)\, \{r(a) - b\}].$$

Same expectation, different variance. Subtracting a baseline is the standard way to reduce the variance of this estimator, and it costs nothing in correctness. A natural choice is $b = J(\theta)$, the average reward under the current policy: then $r(a) - b$ is positive only for arms better than average, so the shared push disappears. The variance-minimizing b is slightly different — Appendix C.2. In statistics this device is called a **control variate**.

What Is Still Missing

With a one-step horizon we now have a complete algorithm: sample, weight by reward minus baseline, and ascend.

Two things break when the episode continues.

- **Credit assignment**: a reward at step 50 must be attributed to actions taken long before it.
- **Induced states**: the policy now decides which states are visited, and that dependence was absent from every line above.

Both are handled by one exact identity, which is where the next lecture begins.

Outline

Summary: The Model and the Objective

$$V^\pi(s) = \sum_a \pi(a \mid s)\Big\{r(s,a) + \gamma \cdot \sum_{s'} P(s' \mid s,a) \cdot V^\pi(s')\Big\}, \qquad J(\pi) = \mathbb{E}_{s_0 \sim \mu} V^\pi(s_0).$$

1. A Markov decision process separates what the world does, P and r, from what the learner chooses, π.
2. The Bellman equation turns an infinite horizon into a one-step recursion, and evaluates one fixed policy.
3. Learning is the single optimization problem $\max_\theta J(\theta)$ — with an objective whose sampling distribution moves with θ.

Summary: Differentiating an Expectation You Sample From

$$\nabla_\theta J(\theta) = \mathbb{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a)\{r(a) - b\}], \qquad \nabla_\theta \log \pi_\theta(a) = e_a - \pi_\theta.$$

1. The score trick rewrites a sum over all actions as an expectation over the actions actually played, so the environment need not be known or differentiable.
2. The estimator is weighted maximum likelihood on the policy's own samples: raise the log probability of what paid.
3. Any action-independent baseline leaves the gradient unchanged and can cut its variance several-fold.

Next Lecture

The bandit gradient needed one step. A real policy acts many times before anything is scored.

Lecture 21 — Policy Optimization asks what the exact difference in performance between two policies is, shows that the policy gradient is its limit, and derives actor–critic and PPO from that one identity.

Reading for this lecture: Sutton and Barto, *Reinforcement Learning*, Chapters 3–4; D2L Chapter 17.

Appendix

Four groups of notes skipped in the lecture hour.

A. Discounting, values, and Bellman. The effective horizon $1/(1-\gamma)$ and the cost of the infinite-horizon approximation; the advantage averages to zero; the Bellman equation; why $(I - \gamma P^{\pi})$ inverts.

B. Gradients through a sampling step. The reparameterization trick, and when to prefer it to the score function.

C. Baselines. Why one does not bias the gradient, and which minimizes its variance.

D. Beyond the bandit. The contextual bandit, where the baseline becomes a value function.

A.0 Where the Effective Horizon Comes From

derive

Sum the geometric weights that discounting places on future rewards:

$$\sum_{k\geq 0} \gamma^k = \frac{1}{1-\gamma}, \qquad 0 \leq \gamma < 1.$$

A finite horizon of H steps weights each of H rewards by 1, for a total weight of H. Matching the two totals gives

$$\boxed{H_{\text{eff}} = \frac{1}{1-\gamma}}$$

— the discounted problem spreads the same total weight that an undiscounted problem of $1/(1-\gamma)$ steps would. A second reading agrees: the normalized weights $(1-\gamma)\gamma^k$ form a geometric distribution on k with mean $\gamma/(1-\gamma)$.

A.0.1 Truncation Error of the Infinite Horizon

derive

Suppose rewards are bounded, $|r_t| \le R$, and the true episode ends at T. The tail that the infinite-horizon objective adds but the real problem does not is

$$\Big|\sum_{t \ge T} \gamma^t \cdot r_t\Big| \le R \cdot \sum_{t \ge T} \gamma^t = \frac{R \cdot \gamma^T}{1-\gamma}.$$

The bound decays geometrically in T. So when the episode is long relative to $1/(1-\gamma)$, treating a finite-horizon environment as infinite costs almost nothing — which is why Atari, robotics, and dialogue are all written as discounted infinite-horizon problems even though every episode stops.

Read the other way: if T is short relative to $1/(1-\gamma)$, the approximation is poor and the finite-horizon problem, with a time-dependent policy, is the honest formulation.

A.1 The Advantage Averages to Zero

derive

Conditioning on the first action gives

$$V^\pi(s) = \sum_a \pi(a \mid s) \cdot Q^\pi(s, a),$$

so

$$\begin{aligned} \mathbb{E}_{a \sim \pi(\cdot|s)}[A^\pi(s, a)] &= \sum_a \pi(a \mid s)\{Q^\pi(s, a) - V^\pi(s)\} \\ &= V^\pi(s) - V^\pi(s) = 0. \end{aligned}$$

Consequently $\max_a A^\pi(s, a) \geq 0$ at every state: an average of zero cannot have all terms negative. Some action is at least as good as current habit, which is what policy improvement will exploit.

A.2 The Bellman Equation from Iterated Expectation

derive

Start from the recursion $G_t = r_t + \gamma \cdot G_{t+1}$ and condition on $s_t = s$:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[r_t + \gamma \cdot G_{t+1} \mid s_t = s] \\ &= \sum_a \pi(a \mid s) \Big\{ r(s,a) + \gamma \cdot \sum_{s'} P(s' \mid s,a)\, \mathbb{E}_\pi[G_{t+1} \mid s_{t+1} = s'] \Big\} \\ &= \sum_a \pi(a \mid s) \Big\{ r(s,a) + \gamma \cdot \sum_{s'} P(s' \mid s,a) \cdot V^\pi(s') \Big\}. \end{aligned}$$

The middle step uses the Markov property twice: to drop the history from the transition law, and to recognize $\mathbb{E}_\pi[G_{t+1} \mid s_{t+1} = s']$ as $V^\pi(s')$ regardless of how s' was reached.

A.3 Why $(I - \gamma P^{\pi})$ Is Invertible

derive

Every row of P^{π} is a probability vector, so $\|P^{\pi}v\|_{\infty} \le \|v\|_{\infty}$ for any v. Hence

$$\|\gamma P^{\pi}v\|_{\infty} \le \gamma \cdot \|v\|_{\infty}, \qquad \gamma < 1.$$

If $(I - \gamma P^{\pi})v = 0$ then $v = \gamma P^{\pi}v$ and $\|v\|_{\infty} \le \gamma \cdot \|v\|_{\infty}$, which forces $v = 0$. A square matrix with trivial null space is invertible, so

$$V^{\pi} = (I - \gamma P^{\pi})^{-1} r^{\pi} = \sum_{k \ge 0} (\gamma P^{\pi})^{k} r^{\pi},$$

the second form being the Neumann series — literally "sum the discounted expected rewards over all future steps".

B.1 Reparameterization for Continuous Actions

derive

Let the policy be Gaussian with a network mean and scale,

$$a = \mu_\theta(s) + \sigma_\theta(s) \odot \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I),$$

which is exactly the VAE encoder's sampling step of Lecture 17. The noise ε carries no parameters, so the expectation can be written over a fixed distribution:

$$\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[Q(s,a)] = \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}[Q(s,\ \mu_\theta(s) + \sigma_\theta(s) \odot \varepsilon)].$$

B.1 Reparameterization for Continuous Actions

derive

Let the policy be Gaussian with a network mean and scale,

$$a = \mu_\theta(s) + \sigma_\theta(s) \odot \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I),$$

which is exactly the VAE encoder's sampling step of Lecture 17. The noise ε carries no parameters, so the expectation can be written over a fixed distribution:

$$\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[Q(s,a)] = \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}[Q(s,\ \mu_\theta(s) + \sigma_\theta(s) \odot \varepsilon)].$$

Now θ appears only inside the integrand, so the gradient passes through the expectation as in supervised learning:

$$\boxed{\nabla_\theta \mathbb{E}_{a\sim\pi_\theta}[Q(s,a)] = \mathbb{E}_\varepsilon[\nabla_a Q(s,a)|_{a=\mu_\theta+\sigma_\theta\odot\varepsilon} \cdot \nabla_\theta(\mu_\theta + \sigma_\theta \odot \varepsilon)]}$$

B.2 When to Prefer Each Estimator

derive

Reparameterization	Score function
needs a differentiable Q in a and a continuous, reparameterizable action law; usually much lower variance, because it uses $\nabla_a Q$ rather than only a scalar score	needs only $\log \pi_\theta(a \mid s)$ and a sampled return; works for discrete actions and for a Q available only as a number, at the cost of higher variance

This is the same trade-off as in Lecture 17, where the reparameterized gradient was chosen over the score-function form for precisely the variance reason. The main text uses the score function because the actions in these lectures are discrete and the return arrives as a sampled number. Continuous-control algorithms that do assume a differentiable critic — SAC and DDPG among them — use the reparameterized form of B.1.

C.1 A Baseline Does Not Bias the Gradient

derive

For any b depending on the state alone,

$$\begin{aligned}\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[\nabla_\theta \log \pi_\theta(a \mid s) \cdot b(s)] &= b(s) \sum_a \pi_\theta(a \mid s) \frac{\nabla_\theta \pi_\theta(a \mid s)}{\pi_\theta(a \mid s)} \\ &= b(s) \nabla_\theta \sum_a \pi_\theta(a \mid s) \;=\; b(s) \nabla_\theta 1 = 0.\end{aligned}$$

The step that matters is exchanging ∇_θ with the finite sum over actions, after which the sum is the constant 1. Nothing about the reward was used, which is why any function of the state may be subtracted.

C.2 Which Baseline Minimizes the Variance

derive

Write $u(a) = \nabla_\theta \log \pi_\theta(a)$ for one coordinate. Since the mean does not depend on b, minimizing the variance means minimizing

$$\mathbb{E}[u(a)^2\{r(a) - b\}^2].$$

Differentiating in b and setting the result to zero gives

$$b^\star = \frac{\mathbb{E}[u(a)^2 r(a)]}{\mathbb{E}[u(a)^2]},$$

a score-weighted average reward. In practice the unweighted mean reward $J(\theta)$ — or, with states, $V^{\pi_\theta}(s)$ — is used instead: it is nearly as good and can be estimated by a second network.

D.1 The Contextual Bandit

derive

Now a state is drawn first, $s \sim \mu$, and the episode is still one step:

$$J(\theta) = \mathbb{E}_{s\sim\mu}\, \mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[r(s,a)].$$

D.1 The Contextual Bandit

derive

Now a state is drawn first, $s \sim \mu$, and the episode is still one step:

$$J(\theta) = \mathbb{E}_{s\sim\mu}\, \mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[r(s,a)].$$

The score argument applies inside each state, and the baseline may now depend on the state:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\mu,\, a\sim\pi_\theta(\cdot|s)}[\nabla_\theta \log \pi_\theta(a \mid s)\{r(s,a) - b(s)\}].$$

The natural choice is $b(s) = V^{\pi_\theta}(s)$, the average reward in that state — which makes the weight $r(s,a) - V^{\pi_\theta}(s)$ an advantage. Remember this.