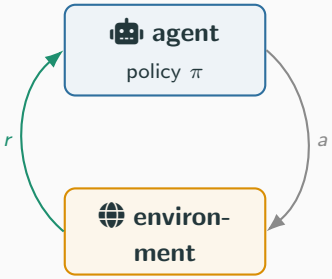
Reinforcement Learning

Bandits, policy gradients, and how they train language models

CSCI 567 — Machine Learning • Lecture 11 • Nov 20, 2026

Reinforcement learning: learn from rewards on your own actions, not from labels



In **state** s the agent picks an **action** a; the environment returns a scalar **reward** r. A chess engine picks a move, a robot arm a torque, a recommender an ad, a language model the next token.

	Lectures 01–10	**Reinforcement learning**
Feedback	the correct label $\mathbf{y}_i$	a reward for the action taken, nothing about the other actions
Data	fixed, i.i.d. from $\mathcal{D}$	generated by the agent's own policy
Output	a predictor $f(\mathbf{x})$	a policy $\pi(a \mid s)$

Nobody tells the agent which action was correct. It sees one number for the action it chose, and its choices decide what data it sees next. The goal is a **policy** π, a rule for choosing actions, that collects as much reward as possible over time.

Two difficulties that supervised learning did not have

Difficulty	What it means	Where we meet it
Exploration	A reward tells you about the action you took. To learn about the others you must try actions that currently look worse.	Section 1, bandits
No gradient	The environment returns a number, not a derivative. We cannot backpropagate through a game, a user, or a test suite.	Section 2, policy gradients

Scope. Today every decision gets its own reward: the agent sees a situation, acts once, and is scored. We leave out problems where a reward arrives many steps after the action that earned it, and the methods built for them (discounted returns, Q-learning, DQN). The one-decision setting is enough for the methods that train language models.

Outline

1. **Multi-armed bandits**: the exploration-exploitation tradeoff. Regret, the greedy rule, ϵ-greedy.

2. **Policy gradients**: learn the policy directly. The score-function identity, REINFORCE, baselines, entropy regularization.

3. **RL for language models**: the model as a policy, PPO, verifiable rewards and GRPO.

1. Multi-Armed Bandits

The exploration-exploitation tradeoff

The K-armed bandit and regret

Setting (Robbins, 1952). There are K arms (actions). Arm i pays a random reward in $[0, 1]$ with a fixed, unknown mean μ_i. In each round $t = 1, 2, \ldots$ the learner pulls one arm i_t and sees only that arm's reward. Examples: K page designs with one visitor per round, or K treatments with one patient per round.

Regret. Let $\mu^\star = \max_i \mu_i$, the gaps $\Delta_i = \mu^\star - \mu_i$, and $n_i(t)$ the number of pulls of arm i in the first t rounds.

$$R_t \;=\; \sum_{k=1}^{t} \big(\mu^\star - \mu_{i_k}\big), \qquad \mathbb{E}[R_t] \;=\; \sum_{i=1}^{K} \Delta_i \, \mathbb{E}[n_i(t)].$$

The second form says regret is a count: how often each bad arm is pulled, weighted by how bad it is.

Running example. $K = 5$ Bernoulli arms with means (0.30, 0.40, 0.50, 0.55, 0.70): $\mu^\star = 0.70$ and gaps (0.40, 0.30, 0.20, 0.15, 0). Pulling arms uniformly at random costs $\frac{1}{5}\sum_i \Delta_i = 0.21$ per round.

Linear regret means never learning; the greedy rule has it

Linear regret, $\mathbb{E}[R_t] = \Theta(t)$: a constant fraction of rounds is wasted forever. Playing any fixed suboptimal arm, or playing at random, has this.

Sublinear regret, $\mathbb{E}[R_t]/t \to 0$: the average reward converges to $\mu^\star$. The question is then how slowly R_t can grow.

The tradeoff. Exploiting uses what the estimates say now; exploring improves the estimates. An estimate only improves for an arm that is still being pulled.

Greedy. Pull each arm once, then always pull $\arg\max_i \hat{\mu}_i$, the arm with the best sample mean.

Take two arms with means 0.70 and 0.55. With probability $0.3 \times 0.55 = 0.165$ the good arm pays 0 and the bad arm pays 1 on their first pulls. From then on $\hat{\mu}_{\text{bad}} > 0 = \hat{\mu}_{\text{good}}$ forever: the good arm is never pulled again, so its estimate is never corrected.

$$\mathbb{E}[R_t] \;\geq\; 0.165 \times 0.15 \times (t-2).$$

Greedy exploits without exploring; uniform random play explores without exploiting. Both are linear.

ϵ-greedy: forced exploration at a constant cost per round

ϵ-greedy
In each round, with probability ϵ pull a uniformly random arm (explore); otherwise pull $\arg\max_i \hat{\mu}_i$ (exploit).

What it fixes. Every arm is pulled infinitely often, so every $\hat{\mu}_i \to \mu_i$ and the exploit rounds eventually pick the best arm. Greedy's failure cannot happen.

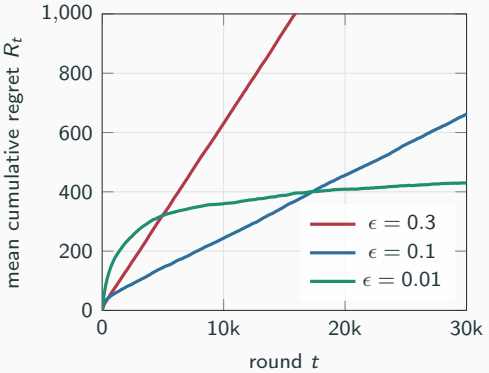
What it costs. The explore rounds never stop. In the long run each round costs

$$\epsilon \cdot \frac{1}{K} \sum_{i=1}^{K} \Delta_i \;=\; 0.1 \times 0.21 \;=\; 0.021 \qquad \text{(running example, } \epsilon = 0.1\text{)},$$

so the regret is still linear, with a slope ϵ times that of random play.

ϵ sets the tradeoff. A large ϵ learns the arm means quickly and then keeps wasting pulls. A small ϵ wastes little in the long run but takes long to find the best arm.

Computed: the exploration-exploitation tradeoff on the running example



Simulation from the Colab notebook: mean of 150 runs.

R_t at $t =$	$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.3$
1,000	173.2	**56.1**	73.5
10,000	360.2	**243.6**	630.6
30,000	**429.8**	661.8	1888.6
predicted slope	0.0021	0.021	0.063

Too little ($\epsilon = 0.01$): after 1,000 rounds it has made about 10 exploration pulls and is often still exploiting the wrong arm.

Too much ($\epsilon = 0.3$): it finds the best arm early, then pays 0.063 per round forever.

No ϵ wins at every horizon: $\epsilon = 0.1$ leads at $t = 10$k, $\epsilon = 0.01$ at $t = 30$k.

2. Policy Gradient Methods

Learn the policy directly, by gradient ascent on the reward

Parameterize the policy and maximize the expected reward by gradient ascent

From arms to situations. In most problems the best action depends on a **state** s: the user who arrived, the board position, the prompt. A bandit keeps one number $\hat{\mu}_i$ per arm. One number per state and action is impossible when the states are all images or all texts. We need a model that generalizes across states.

Policy gradient idea. Let the model output the policy itself, a distribution over actions, and do gradient ascent on the expected reward:

$$\pi_\theta(a \mid s) = \frac{e^{f_\theta(s,a)}}{\sum_{a'} e^{f_\theta(s,a')}}, \qquad J(\theta) = \mathbb{E}_{s,\ a\sim\pi_\theta(\cdot|s)}\big[r(s,a)\big], \qquad \theta \leftarrow \theta + \eta\,\nabla_\theta J(\theta).$$

The policy is a softmax (Lecture 05) over the scores $f_\theta(s,a)$ of a network. Acting means sampling $a \sim \pi_\theta(\cdot \mid s)$.

The obstacle. θ enters J through *which actions get sampled*. We cannot backpropagate through a sampled action, and the reward comes from an environment that returns a number and no gradient.

The score-function identity: the gradient of an expectation as an expectation

Fix a state and drop it from the notation: an action $a \sim \pi_\theta$ and a reward $r(a)$, so $J(\theta) = \mathbb{E}_{a\sim\pi_\theta}[r(a)] = \sum_a \pi_\theta(a)\, r(a)$.

$$\nabla_\theta J(\theta) = \sum_a r(a)\, \nabla_\theta \pi_\theta(a) \qquad r(a) \text{ does not depend on } \theta$$

$$= \sum_a r(a)\, \pi_\theta(a)\, \frac{\nabla_\theta \pi_\theta(a)}{\pi_\theta(a)} \qquad \text{multiply and divide by } \pi_\theta(a)$$

$$= \sum_a \pi_\theta(a)\, r(a)\, \nabla_\theta \log \pi_\theta(a) \qquad \text{since } \nabla \log \pi = \nabla \pi / \pi$$

$$= \mathbb{E}_{a\sim\pi_\theta}\big[\, r(a)\, \nabla_\theta \log \pi_\theta(a)\,\big] \qquad \text{a sum weighted by } \pi_\theta(a) \text{ is an expectation.}$$

$\nabla_\theta\, \mathbb{E}_{a\sim\pi_\theta}[r(a)] \;=\; \mathbb{E}_{a\sim\pi_\theta}\big[r(a)\, \nabla_\theta \log \pi_\theta(a)\big]$ (Williams, 1992). The vector $\nabla_\theta \log \pi_\theta(a)$ is called the **score function**.

Why the identity removes the obstacle, and what it says for a softmax policy

Three things changed.

- The right side is an expectation over the policy's own actions, so it can be estimated by sampling:

$$\nabla_\theta J \approx \frac{1}{N} \sum_{n=1}^{N} r(a_n)\, \nabla_\theta \log \pi_\theta(a_n).$$

- The reward is only *evaluated* at the sampled actions. It is never differentiated, so it can be a game, a user, or a test suite.
- $\nabla_\theta \log \pi_\theta$ is the gradient of our own model: ordinary backpropagation.

Softmax policy. With $\pi_\theta(a) \propto e^{\theta_a}$,

$$\nabla_\theta \log \pi_\theta(a) = \mathbf{e}_a - \boldsymbol{\pi}_\theta,$$

the one-hot vector of the sampled action minus the probabilities. This is the log-likelihood gradient of multiclass logistic regression (Lecture 05) with the sampled action as the label.

The update $\theta \leftarrow \theta + \eta\, r(a)\, (\mathbf{e}_a - \boldsymbol{\pi}_\theta)$ is a supervised step toward the action just taken, scaled by its reward.

REINFORCE: make the sampled action more likely in proportion to its reward

REINFORCE (Williams, 1992)
Repeat: observe a state s; sample an action $a \sim \pi_\theta(\cdot \mid s)$; receive the reward r; update

$$\theta \leftarrow \theta + \eta\, r\, \nabla_\theta \log \pi_\theta(a \mid s).$$

Reading the update. $\nabla_\theta \log \pi_\theta(a \mid s)$ is the direction that makes the action just taken more likely. It is one sample ($N = 1$) of the identity, so the update is stochastic gradient ascent on J. Actions with a high reward are pushed up strongly, actions with a low reward weakly, and actions with a negative reward are pushed down.

As weighted supervised learning. The update is a gradient step on the cross-entropy loss with the agent's own action as the label, weighted by its reward. Imitate yourself, in proportion to how well it went.

On-policy. The expectation is over actions from the *current* π_θ. After each update the old samples come from the wrong distribution, so they cannot be reused without a correction (Section 3).

A baseline reduces the variance without changing the expectation

Claim. For any b that does not depend on the action,

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

so $\nabla_\theta J = \mathbb{E}\big[(r(a) - b)\,\nabla_\theta \log \pi_\theta(a)\big]$ for every such b: the estimator stays unbiased.

Why it helps. Suppose all rewards are positive, say between 99 and 101. Without a baseline every sampled action is pushed up by about 100, and only the small differences carry information. With $b = 100$ the better-than-average actions go up and the worse ones go down.

Advantage. The baseline may depend on the state. The usual choice is an estimate $b(s)$ of the expected reward in state s, and the weight becomes the **advantage**

$$A = r - b(s)\,:$$

how much better this action did than expected.

Section 3 estimates the baseline from a group of samples.

Entropy regularization: an incentive for the policy to keep exploring

Problem. A policy gradient explores only through the randomness of π_θ. If a few lucky rewards push one action's probability near 1, the others are almost never sampled and get no update: the greedy rule's failure again.

Entropy $H(\pi) = -\sum_a \pi(a) \log \pi(a)$ measures how random the policy is: $\ln K$ for the uniform policy, 0 for a deterministic one.

Entropy bonus (Williams, Peng, 1991)
Maximize $\mathbb{E}_{a\sim\pi_\theta}[r(a)] + \beta\, H(\pi_\theta)$:

$$\theta \leftarrow \theta + \eta\big[\, r\, \nabla_\theta \log \pi_\theta(a) + \beta\, \nabla_\theta H(\pi_\theta)\big]$$

The new term needs no sample: it pulls the policy toward uniform at every step.

Computed (Colab notebook). Section 1's 5-armed bandit, softmax policy $\pi_\theta(a) \propto e^{\theta_a}$, $\eta = 0.5$, 5,000 rounds, 300 runs.

β	runs ending on the best arm	final π(best)	expected reward
0	89%	0.889	0.682
0.01	91%	0.912	0.686
0.05	**100%**	0.985	**0.697**
0.2	70%	0.494	0.587

Without the bonus, 11% of runs lock onto a worse arm. With $\beta = 0.2$ the policy stays near uniform and earns 0.587; the best arm pays 0.70. β sets the tradeoff, as ϵ did. A3C (Mnih et al., 2016) and PPO on Atari use 0.01.

3. RL for Language Models

The model as a policy, PPO, and verifiable rewards

A language model is a policy: the prompt is the state, the response is the action

RL term	For a language model
state s	the prompt x
action a	a complete response $y = (y_1, \ldots, y_T)$, sampled token by token
policy $\pi_\theta(a \mid s)$	$\pi_\theta(y \mid x) = \prod_{t=1}^{T} \pi_\theta(y_t \mid x, y_{<t})$, a product of softmaxes over the vocabulary
reward $r(s, a)$	one score $r(x, y)$ for the finished response
$\nabla_\theta \log \pi_\theta(a \mid s)$	$\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(y_t \mid x, y_{<t})$, one backward pass through the model

Why RL is needed here. Many responses to a prompt are acceptable, and for hard problems nobody has written the ideal one. There is no single label to regress on.

What is available. A response can often be *scored*: by a person, or by an automatic checker. A score is a reward. The set of actions is far too large to enumerate, but it can be sampled.

RL fine-tuning is reward-weighted likelihood on the model's own samples

Section 2's gradient with this dictionary, next to the pretraining gradient from Lecture 08:

$$\textbf{RL:} \qquad \nabla_\theta J = \mathbb{E}_{y \sim \pi_\theta(\cdot|x)}\Big[\, r(x,y) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(y_t \mid x, y_{<t}) \Big]$$

$$\textbf{Pretraining:} \qquad \mathbb{E}_{y \sim \text{data}}\Big[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(y_t \mid x, y_{<t}) \Big]$$

Same computation. Both are log-likelihood gradients of a response, from the same backward pass.

Different data and weights. Pretraining raises the likelihood of text someone wrote. RL raises the likelihood of the model's *own* responses in proportion to their reward.

Consequence. The model can become better than its training text: it is no longer limited to imitating responses that already exist.

Cost. Every gradient step needs new responses sampled from the current model, and the estimate is noisy.

PPO: several gradient steps on one batch of samples, safely

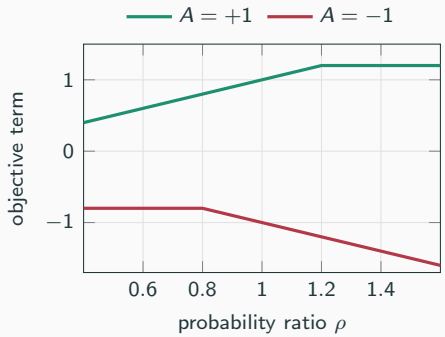
Problem. Sampling is the expensive part, and REINFORCE uses each batch for one step. After several steps on the same batch, π_θ has moved away from the policy $\pi_{\theta_{\text{old}}}$ that produced the samples, and the gradient estimate is wrong.

Fix. Reweight each sample by the probability ratio $\rho(\theta) = \dfrac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)}$, and stop rewarding changes once ρ leaves $[1 - \epsilon_{\text{clip}},\, 1 + \epsilon_{\text{clip}}]$.

PPO clipped objective (Schulman et al., 2017)

$$\mathbb{E}\Big[\min\big(\rho A,\ \text{clip}(\rho, 1-\epsilon_{\text{clip}}, 1+\epsilon_{\text{clip}})\, A\big)\Big]$$

A is the advantage of the sampled action; the paper uses $\epsilon_{\text{clip}} = 0.2$.



One term of the objective, computed from the formula with $\epsilon_{\text{clip}} = 0.2$. A good action (green) gains nothing beyond $\rho = 1.2$, a bad one (red) nothing below $\rho = 0.8$: the curve is flat there, so the gradient is zero. Each action's probability changes by about $\pm 20\%$ per batch.

RLVR and GRPO: a checkable reward, with the group mean as the baseline

Reinforcement learning with verifiable rewards (RLVR). For some tasks the reward needs no learned model: a math answer matches the reference or it does not; a program passes the hidden tests or it does not. The reward $r \in \{0, 1\}$ is computed by a rule.

GRPO (Shao et al., 2024): the advantage without a value network

For each prompt, sample a group of G responses, score each, and use

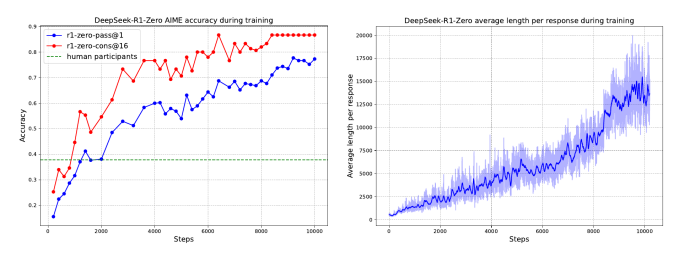
$$\hat{A}_i = \frac{r_i - \mathrm{mean}(r_1, \ldots, r_G)}{\mathrm{std}(r_1, \ldots, r_G)}$$

as the advantage in PPO's clipped objective.

Example. $G = 4$ responses with rewards $(1, 0, 0, 1)$: mean 0.5, std 0.5, so $\hat{A} = (+1, -1, -1, +1)$. The two correct responses become more likely and the two wrong ones less likely.

When there is no signal. If all G responses are right, or all are wrong, the prompt contributes no gradient. Learning comes from prompts the model solves *sometimes*: Section 1's exploration problem again.

RLVR at scale: DeepSeek-R1-Zero trained from a correctness reward alone



DeepSeek-AI (2025), Figure 1. Left: accuracy on AIME 2024 during RL training (blue: one sample; red: majority vote over 16; green: average human participant). Right: average response length on the training set.

Setup. GRPO on a pretrained base model, with no supervised fine-tuning stage and no human reasoning examples. Rule-based reward: correct final answer or passing tests, plus the required format. 32 questions $\times$ 16 responses per step, 10,400 steps.

Result. AIME 2024 accuracy rises from 15.6% to 77.9% (86.7% with a 16-sample vote). Responses grow to over ten thousand tokens, though the reward says nothing about length. The reward is a rule, not a learned model.

Summary: the methods of this lecture

Method	What it does	Number from today
ϵ-greedy	random arm with probability ϵ, else $\arg\max_i \hat{\mu}_i$	$\epsilon = 0.1$: regret 0.021 per round, forever
REINFORCE	$\theta \leftarrow \theta + \eta\, r\, \nabla \log \pi_\theta(a \mid s)$	finds the best arm in 89% of runs
Entropy bonus	adds $\beta\, \nabla H(\pi_\theta)$ to the update	100% of runs with $\beta = 0.05$
PPO	clipped objective: several steps per batch	$\epsilon_{\text{clip}} = 0.2$
RLVR with GRPO	rule-based reward, group mean as baseline	AIME 15.6% $\rightarrow$ 77.9%

ϵ-greedy estimates the value of each arm and acts on the estimates. The others learn the *policy* directly, by the same gradient: raise the log-probability of sampled actions in proportion to how well they did.

Three ideas to keep, and one caution

Exploration against exploitation. Feedback covers only the actions taken. Remedies: forced random pulls (ϵ-greedy), or a random policy kept random by an entropy bonus.

Score function. $\nabla \mathbb{E}[r] = \mathbb{E}[r \nabla \log \pi]$ turns a number from the environment into a gradient for the policy, with no model of the environment.

Baseline. Subtracting the expected reward leaves the gradient unbiased and lowers its variance: better-than-expected actions go up, the others go down.

The link to language models. A language model is a policy that maps a prompt to a distribution over responses. RL with verifiable rewards is a policy-gradient method (PPO, GRPO) applied to it.

Caution. A policy optimizes the reward it is given, and a rule-based reward is only as good as its checker. Lecture 01's question returns: is the measured number the thing we care about?

Coming up

Discussion today. Exam 1 (midterm) discussion; project report review.

HW4 is due Monday 11/30, 11:59pm. No class next week (Thanksgiving, 11/27).

Lecture 12 (12/4). Safety (alignment, hallucination, red-teaming, privacy) and interpretability (autoencoders, sparse autoencoders).

Colab notebook. ϵ-greedy on the 5-armed bandit, a numerical check of the score-function identity, and REINFORCE with an entropy bonus: the computed numbers on today's slides.

References

- Robbins. *Some Aspects of the Sequential Design of Experiments.* Bull. AMS 58, 1952.
- Williams. *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning.* Machine Learning 8, 1992.
- Schulman, Wolski, Dhariwal, Radford, Klimov. *Proximal Policy Optimization Algorithms.* arXiv:1707.06347, 2017.
- Williams, Peng. *Function Optimization Using Connectionist Reinforcement Learning Algorithms.* Connection Science 3(3), 1991.
- Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning.* ICML, 2016.
- Shao et al. *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models.* arXiv:2402.03300, 2024.
- DeepSeek-AI. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning.* arXiv:2501.12948, 2025.