Lecture Script

Learning Through Exploration

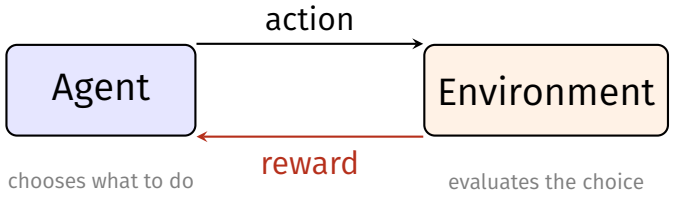
Outline

1. When gradients aren't enough

So far:

- Differentiable programming: push gradients through a simulator
- SBI: simulator is a black box, but we can still do inference by simulating forward

Now a different setting: **optimize a decision** where the objective **cannot be differentiated through**.



Why can't we just use backpropagation on the reward as usual?

Two things break the gradient:

- **Sampling.** The agent draws one action from its distribution. The reward only tells you how *that* action did — not what would have happened with any other. Compare to classification: cross-entropy uses the full probability vector and a known label. No sampling needed.

- **The environment.** The reward comes from the action interacting with the world (physics, game rules, an opponent) — not from the action itself. In classification, the "environment" is just the loss function: you wrote it, so you can differentiate through it. In RL, the environment is external and opaque.

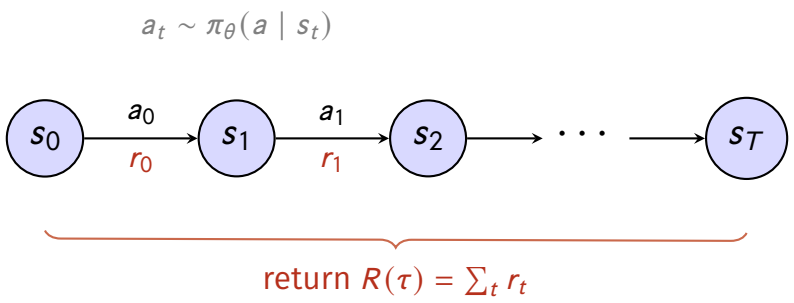
Beyond the non-differentiable reward, several other things make RL harder than supervised learning:

- **Delayed reward (credit assignment).** You make 200 moves, then find out you lost. Which move was the mistake?
- **Stochastic outcomes.** You made a good move but lost anyway because the opponent got lucky. Was the move actually bad, or were you just unlucky?
- **Exploration vs. exploitation.** Do you keep doing what's worked, or try something new that might be better?
- **No fixed dataset.** The data the agent learns from depends on its own policy, which is changing as it learns.

2. The RL problem

Markov Decision Process

The general RL setup: an agent interacts with an environment over discrete time steps.



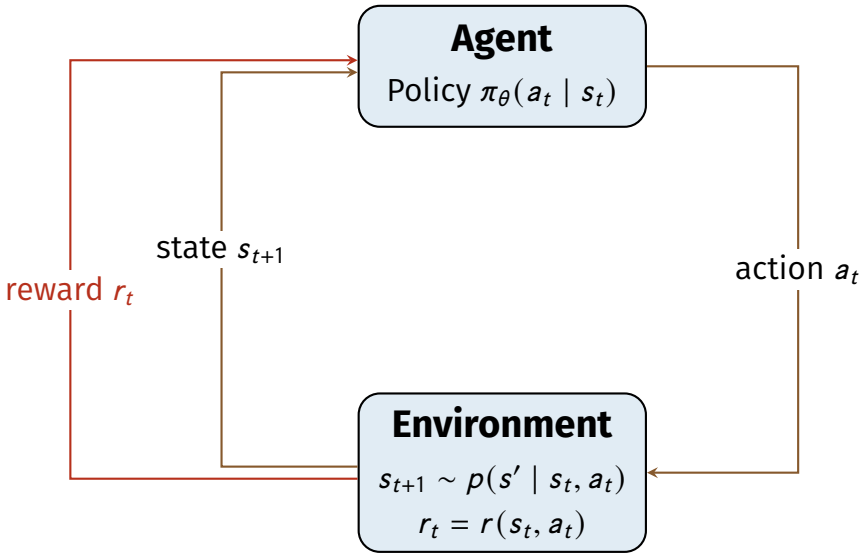
At each step t:

1. Agent observes state s_t

2. Agent chooses action $a_t \sim \pi_\theta(a \mid s)$ according to its **policy** π
3. Environment transitions to $s_{t+1} \sim p(s' \mid s_t, a_t)$ and returns reward $r_t = r(s_t, a_t)$

A **trajectory** (or episode) is a sequence $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T)$.

The full interaction loop, now with notation:



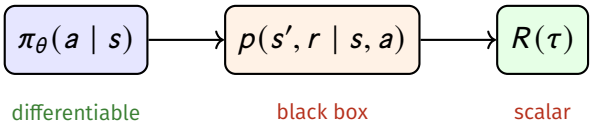
The objective

The goal is to find a policy π_θ that maximizes expected cumulative reward:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} r_t\right] = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

where $R(\tau) = \sum_t r_t$ is the **return** (total reward) of a trajectory.

- We want $\nabla_\theta J(\theta)$ so we can do gradient ascent
- Problem: $R(\tau)$ depends on the environment dynamics $p(s' \mid s, a)$, which are a black box
- We can differentiate through π_θ (our neural network), but not through the environment



▷ How do we get gradients when we can't differentiate through the reward?

3. Policy gradient and REINFORCE

Can't differentiate through the reward or the environment. But we *can* differentiate through our policy π_θ.

The trick: rewrite $\nabla_\theta J$ so the non-differentiable parts appear only inside an expectation $\rightarrow$ estimate by sampling.

Derivation

The probability of a trajectory under policy π_θ is:

$$p(\tau \mid \theta) = p(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t \mid s_t) \cdot p(s_{t+1} \mid s_t, a_t)$$

The objective is:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\theta)}[R(\tau)] = \int p(\tau \mid \theta)\, R(\tau)\, d\tau$$

Differentiate:

$$\nabla_\theta J = \int \nabla_\theta p(\tau \mid \theta)\, R(\tau)\, d\tau$$

Now the **log-derivative trick** (also called the REINFORCE trick) to turn the gradient into an expectation. For any distribution $p(\tau \mid \theta)$:

$$\nabla_\theta p(\tau \mid \theta) = p(\tau \mid \theta) \cdot \nabla_\theta \log p(\tau \mid \theta)$$

Just the chain rule: $\nabla \log f = \nabla f / f$, so $\nabla f = f \cdot \nabla \log f$. Substituting:

$$\nabla_\theta J = \int p(\tau \mid \theta) \cdot \nabla_\theta \log p(\tau \mid \theta) \cdot R(\tau)\, d\tau = \mathbb{E}_{\tau \sim \pi_\theta}[\nabla_\theta \log p(\tau \mid \theta) \cdot R(\tau)]$$

Now expand $\log p(\tau \mid \theta)$:

$$\log p(\tau \mid \theta) = \log p(s_0) + \sum_{t=0}^{T-1} \Big[\log \pi_\theta(a_t \mid s_t) + \log p(s_{t+1} \mid s_t, a_t) \Big]$$

When we take ∇_θ, the initial state $p(s_0)$ and transition dynamics $p(s' \mid s, a)$ don't depend on θ, so they vanish:

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

This is the **policy gradient theorem**. The environment dynamics have disappeared from the gradient. We only need:

1. The ability to **sample** trajectories by running the policy in the environment
2. The ability to **differentiate** through $\log \pi_\theta(a \mid s)$ — which is just our neural network
3. The scalar return $R(\tau)$ — which we observe, don't need to differentiate through

Intuition

What does this gradient do? Look at the update for a single trajectory:

$$\Delta\theta \;\propto\; \underbrace{\left(\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right)}_{\text{direction: make these actions more likely}} \cdot \underbrace{R(\tau)}_{\text{scale: how good was this trajectory?}}$$

$\nabla_\theta \log \pi_\theta(a \mid s)$ is the direction that makes action a more likely in state s. $R(\tau)$ scales this.

The REINFORCE algorithm

Putting this into practice:

REINFORCE

1. Collect N trajectories $\{\tau^{(i)}\}$ by running π_θ in the environment
2. Compute returns $R(\tau^{(i)}) = \sum_t r_t^{(i)}$ for each
3. Estimate the gradient:

$$\hat{g} = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_t \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}) \right) R(\tau^{(i)})$$

4. Update: $\theta \leftarrow \theta + \alpha\, \hat{g}$

Variance reduction with baselines

REINFORCE has **high variance**:

- Returns might be 100 and 105. Both positive → both get reinforced, even the worse one
- Gradient dominated by absolute scale of R, not relative quality

Fix: subtract a baseline b:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\left(\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right) \big(R(\tau) - b \big) \right]$$

Still unbiased for any b that doesn't depend on the actions:

$$b \cdot \mathbb{E}_{\tau \sim \pi_\theta}[\nabla_\theta \log p(\tau \mid \theta)] = b \cdot \nabla_\theta \int p(\tau \mid \theta)\, d\tau = b \cdot \nabla_\theta 1 = 0$$

Simplest baseline: $b = \bar{R}$ (average return across the batch).

- Better than average → positive reinforcement

- Worse than average → negative reinforcement
- Gradient is now about *relative* quality, not absolute scale

On-policy vs. off-policy

- REINFORCE is **on-policy**: collect trajectories, use them once, discard
- **Off-policy** methods reuse old data — more sample-efficient but trickier (data came from a different policy)
- Algorithms like PPO handle this with clipped updates that prevent the policy from changing too fast

▹ How does this connect to LLMs?

4. RL for LLMs (more later)

Same policy gradient framework, different vocabulary:

RL concept	**LLM training**
Policy $\pi_\theta(a \mid s)$	LLM π_θ(token | context)
Action	Generate a response
Reward	Human preference score or verifier output
Environment	Reward model or automated checker

Two flavors:

- **RLHF**: reward = human preference ("which response is better?")
- **RLVR**: reward = automated verifier ("was the answer to this math problem correct?")

Group Relative Policy Optimization (used in DeepSeek-R1). Essentially REINFORCE with a per-prompt baseline:

1. For each prompt, sample G responses (group) from the current policy

2. Score each with a verifier (“is this math correct?”)
3. Baseline b = average score within the group
4. Apply the REINFORCE update