Deep Learning (1470)

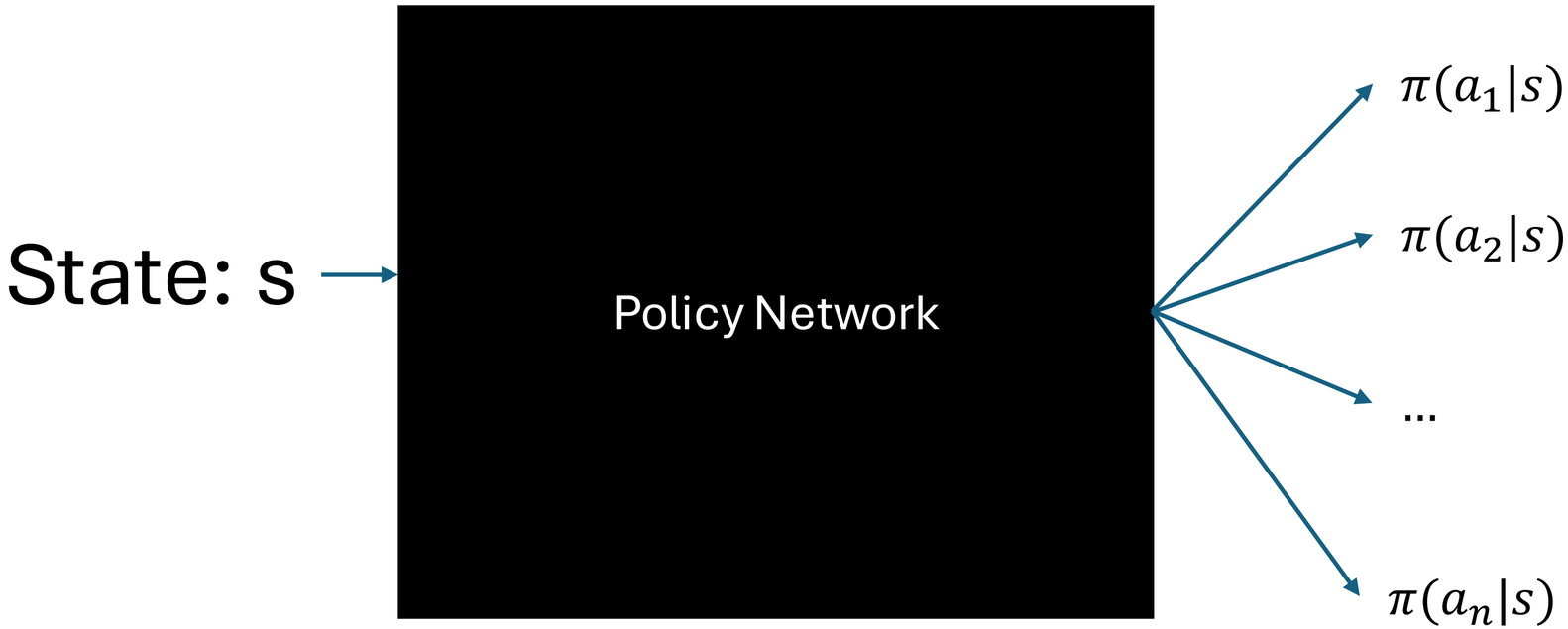
Randall Balestriero

Class 20: Policy Gradient and Actor-Critic

Recap!

Policies

Why learn Q-values first and turn them into a policy? Why not just learn a policy?



How do we train a policy network?

Need to find an appropriate loss function.

What's our objective?

Find a policy π such that the value of the start state is maximized:

$$\pi = \text{argmax}_{\pi} \left(V(s_0)\right)$$

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Returns of a specific trajectory

Probability of a trajectory occurring

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Log-Derivative Trick

We can rewrite the derivative of a function using the derivative of the natural log function:

$$\nabla \ln f(x) = \frac{\nabla \mathrm{f}(x)}{f(x)}$$

$$\nabla f(x) = f(x) \nabla \ln \mathrm{f}(\mathrm{x})$$

When applied to $\Pr(\tau|\theta)$:

$$\nabla_\theta \Pr(\tau|\theta) = \Pr(\tau|\theta)\, \nabla_\theta \ln \Pr(\tau|\theta)$$

Log Probability Trick

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

$$\nabla_\theta \Pr(\tau|\theta) = \Pr(\tau|\theta)\, \nabla_\theta \ln \Pr(\tau|\theta)$$

This gradient term is what we want to calculate

Log Probability Trick

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

$$\nabla_\theta \Pr(\tau|\theta) = \Pr(\tau|\theta)\, \nabla_\theta \ln \Pr(\tau|\theta)$$

This gradient term is what we want to calculate

Log of product -> sum of logs

$$\nabla_\theta \ln \Pr(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T} \ln P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Log of product -> sum of logs

$$\nabla_\theta \ln \Pr(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T} \ln P(s_{t+1}|s_t, a_t) + \ln \pi_\theta(a_t|s_t)$$

Derivative of sum -> sum of derivative

$$\nabla_\theta \ln \Pr(\tau|\theta) = \sum_{t=0}^{T} \nabla_\theta \ln P(s_{t+1}|s_t, a_t) + \nabla_\theta \ln \pi_\theta(a_t|s_t)$$

Gradient of a trajectory

$$\nabla_\theta \ln \Pr(\tau|\theta) = \sum_{t=0}^{T} \nabla_\theta \ln P(s_{t+1}|s_t, a_t) + \nabla_\theta \ln \pi_\theta(a_t|s_t)$$

State transition function
does not depend on θ!

$$\nabla_\theta \ln \Pr(\tau|\theta) = \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)$$

Policy Gradient Derivation

Putting it all back together:

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$ Our Objective

$$\nabla_\theta J(\theta) = \sum_{\tau} \nabla_\theta \Pr(\tau|\theta)\, G(\tau)$$ Take the gradient

$$\nabla_\theta J(\theta) = \sum_{\tau} \Pr(\tau|\theta) G(\tau) \nabla_\theta \ln \Pr(\tau|\theta)$$ Log-Derivative Trick

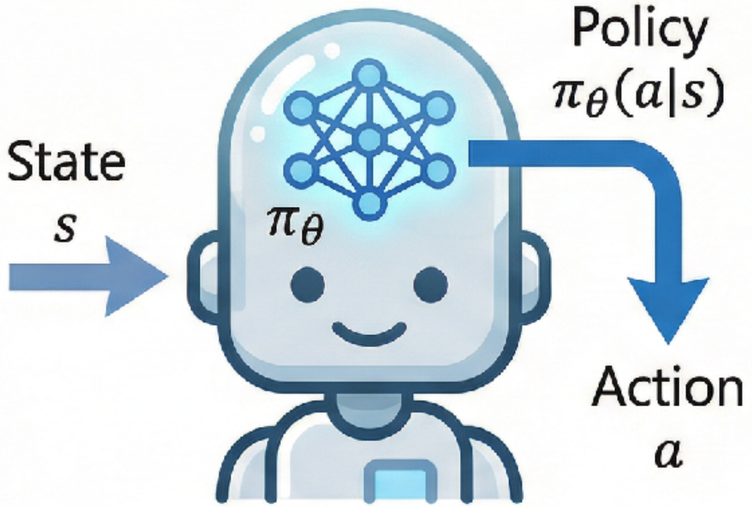
$$\nabla_\theta J(\theta) = \sum_{\tau} [\Pr(\tau|\theta) G(\tau) \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)\ ]$$ Gradient of a Trajectory

$$\nabla_\theta J(\theta) = \mathbb{E}[G_0 \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$ Convert back to Expectation

Policy Gradients

Direct Policy Optimization

Core Idea



Directly parameterize and optimize the policy

Key Equation

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a|s) \cdot R\right]$$

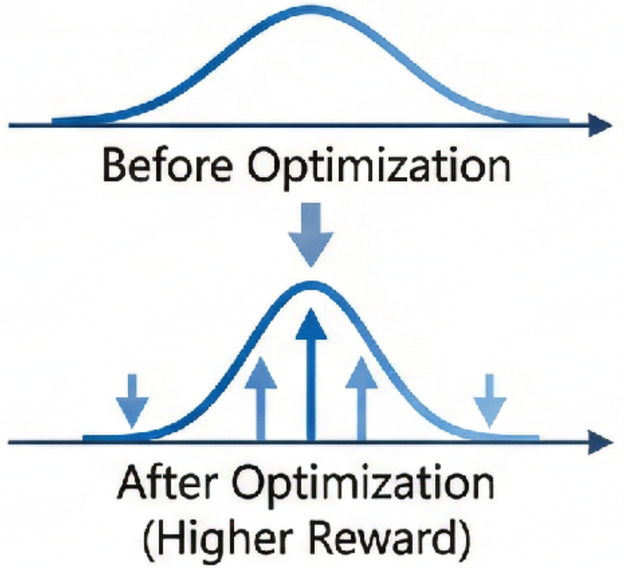
Policy gradient theorem

Gradient of log-prob × Return

Intuition

Good action?
→ Increase probability

Bad action?
→ Decrease probability

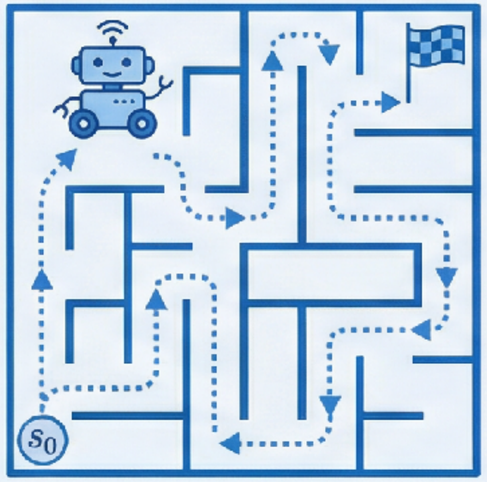


Foundation for: REINFORCE, A2C, A3C, PPO, DDPG, SAC...

REINFORCE

Monte Carlo Policy Gradient (Williams, 1992)

Algorithm Flow



Robot runs a complete episode in maze (trajectory τ)

Collect: $s_0, a_0, r_0, s_1, a_1, r_1, \ldots s_t$

Wait until episode ENDS to compute return G_t

Key Equations

Return: $G_t = \sum \gamma^k r_{t+k}$

(sum of discounted rewards)

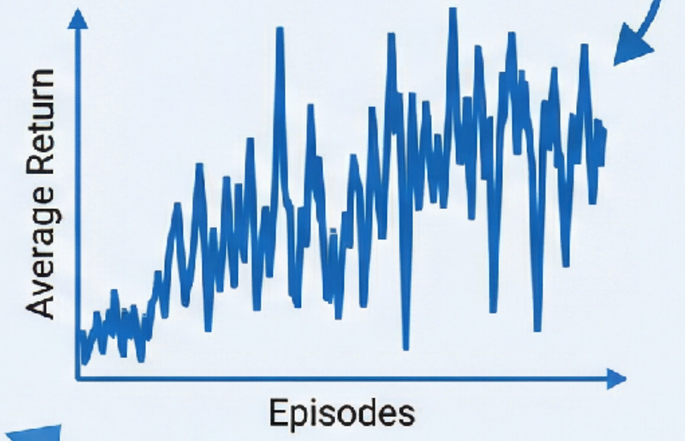
Update:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$$

Weight gradient by actual return

Properties

- ✓ Simple and elegant
- ✓ Unbiased gradient estimate
- ✗ High variance (wait for full episode)
- ✗ Sample inefficient (on-policy)



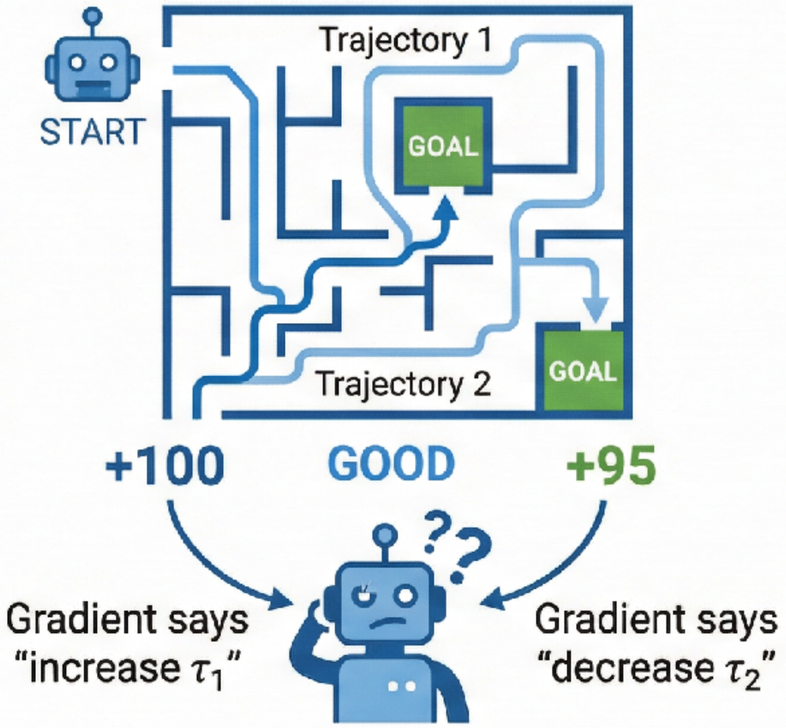
Key insight box

Entire episode return G_t → high variance but unbiased

Reducing Variance

From REINFORCE to Actor-Critic

The Problem



High variance: good actions punished by relative comparison

The Solution: Baseline $b(s)$

Key insight: Subtract a baseline that doesn't depend on action

$$\nabla_\theta J = \mathbb{E}[\nabla_\theta \log \pi(a|s) \cdot (G_t - b(s))]$$

Baseline $b(s)$ reduces variance without adding bias

Best baseline? The value function $V(s)$!

This leads us to... **Actor-Critic methods**

Advantage Function

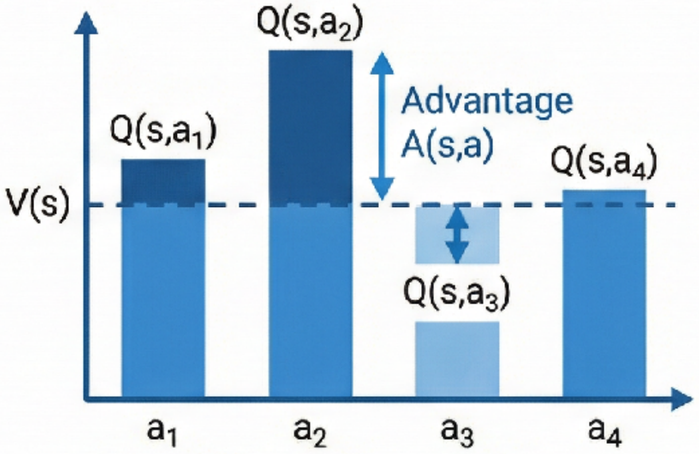
$$A(s,a) = Q(s,a) - V(s)$$

= "How much BETTER is this action?"

Or:

$$A(s,a) \approx G_t - V(s)$$

= "Return minus expected"

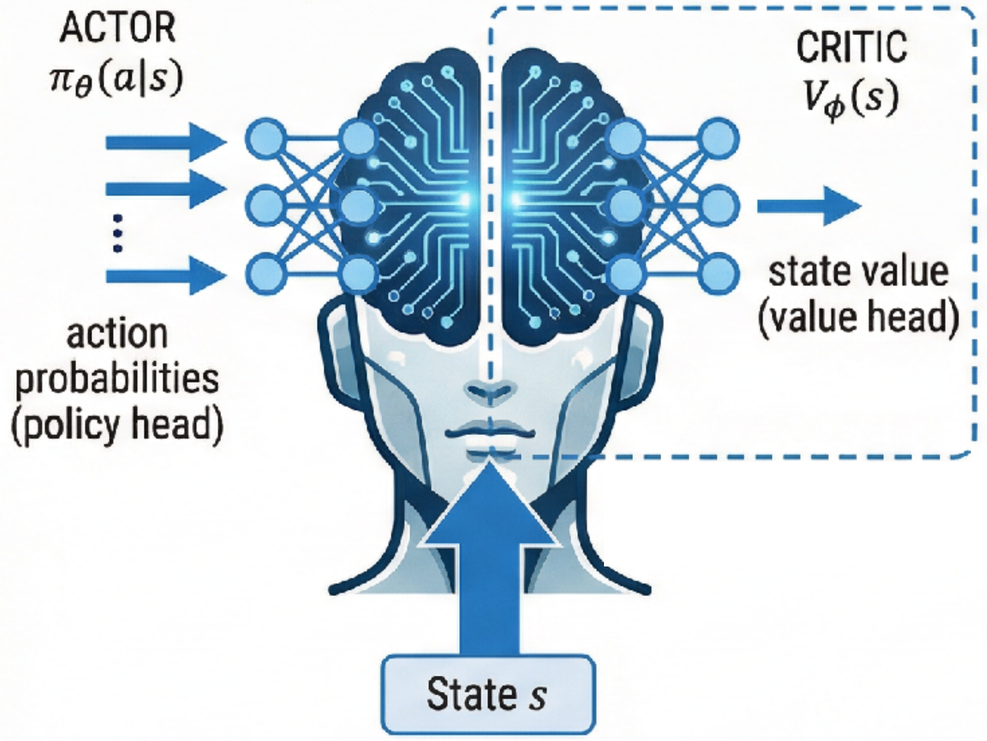


Positive advantage → better than average → increase prob

A2C
Advantage Actor-Critic (Synchronous)

Architecture

Robot with split brain
(actor side + critic side)



Key Equations

Advantage:

$$A_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

(TD error as advantage)

Actor loss:

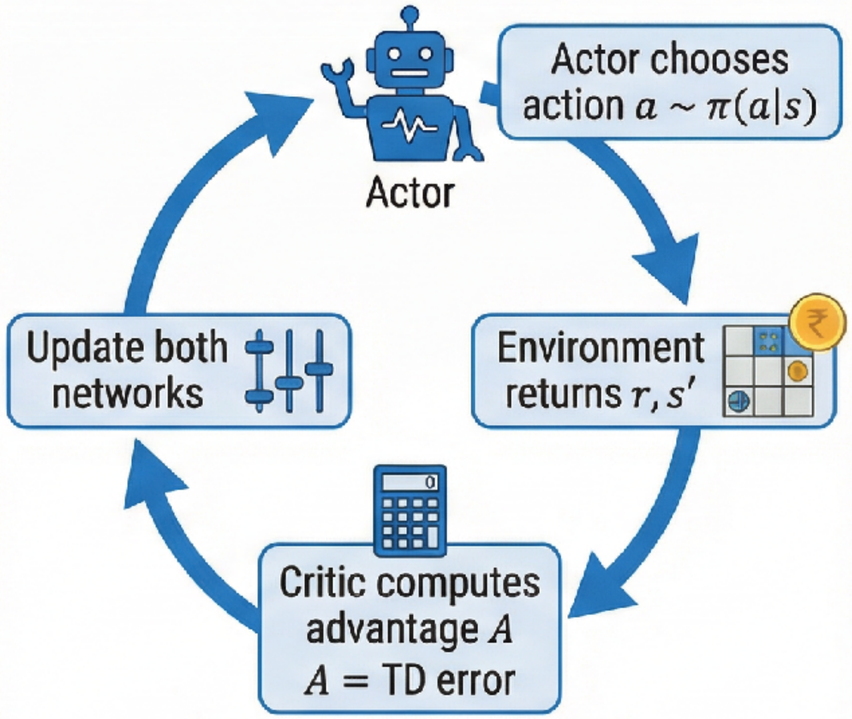
$$L_{actor} = -\log \pi(a|s) \cdot A_t$$

Critic loss:

$$L_{critic} = \left(r_t + \gamma V(s_{t+1}) - V(s_t)\right)^2$$

Critic evaluates, Actor improves

How it works



Can update every step (not wait for episode!)

Properties

✓ Lower variance than REINFORCE (using critic)
✓ Online updates (don't wait for episode end)
✓ Single machine, synchronous

Going Off-Policy

From On-Policy to Off-Policy Actor-Critic

On-Policy Limitation

On-policy: Must discard old data

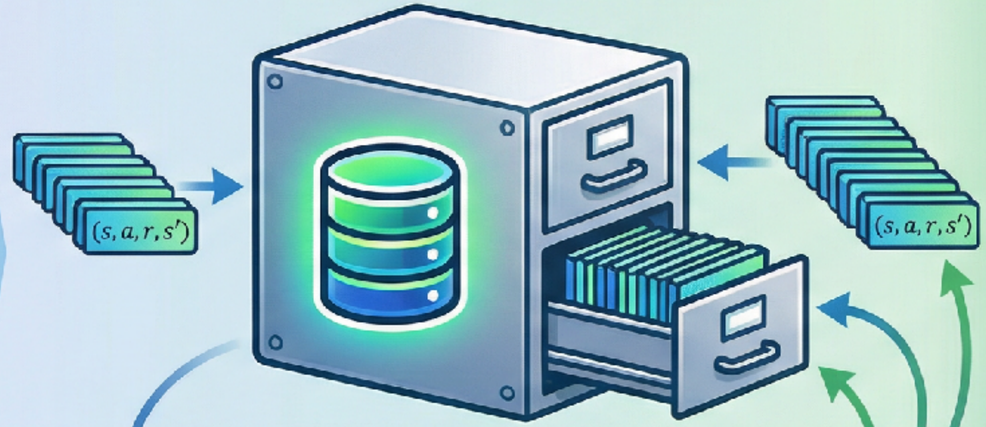
- Data from old π is stale

- Inefficient use of samples

The Key Insight

What if we could **REUSE past experience?**

- Store (s, a, r, s') transitions



- Sample mini-batches for training

Off-Policy Benefits

Sample efficiency: "Learn more from each experience"

Decorrelated data: "Break temporal correlations"

Stability: "Smoother gradients from diverse batches"

But wait... policy gradients need $\nabla \log \pi(a|s)$ from **CURRENT** policy! $\nabla J \approx \mathbb{E}[\nabla \log \pi(a|s) * R]$ → **DDPG**

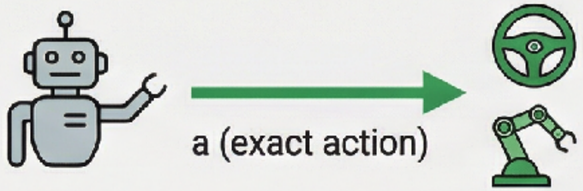
Solution: Learn $Q(s, a)$ instead → **DDPG**, **SAC** $Q(s, a)$ → **SAC**

DDPG

Deep Deterministic Policy Gradient (Lillicrap et al., 2015)

Key Idea

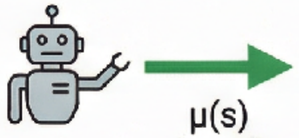
- Deterministic policy: $a = \mu_\theta(s)$



For continuous action spaces 

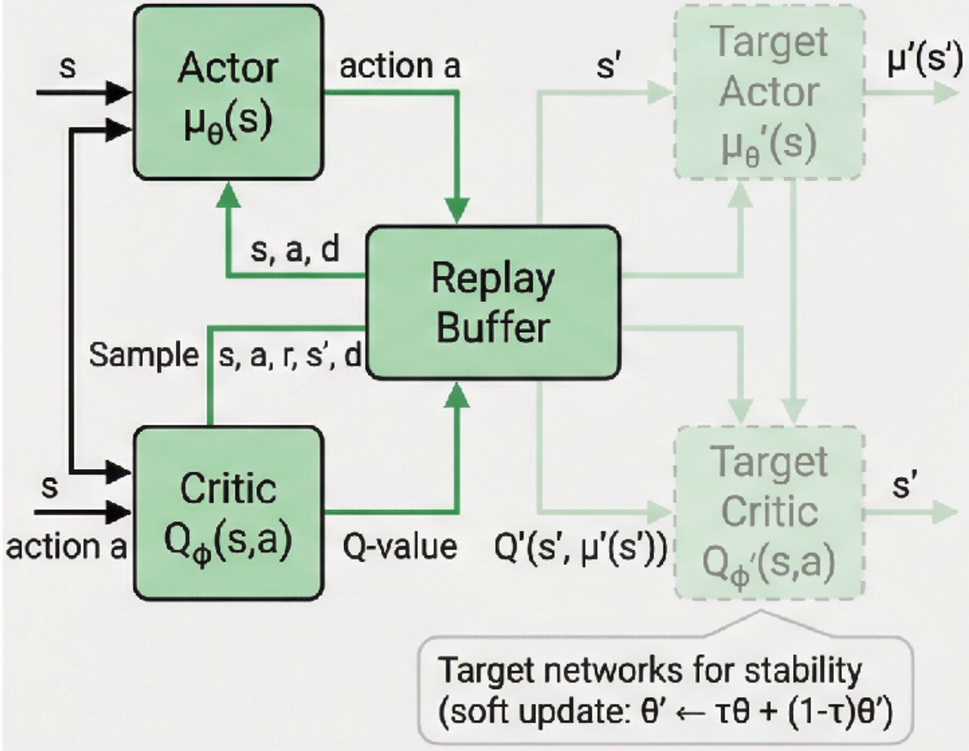
Stochastic (e.g., PG)

Deterministic (DDPG)



Contrast with stochastic: $\pi(a|s)$ vs $\mu(s)$

Architecture



Key Equations

- Critic update: minimize

$$(Q(s,a) - (r + \gamma Q'(s', \mu'(s')))^2$$

- Actor update: maximize

$$Q(s, \mu(s)) \text{ via chain rule}$$

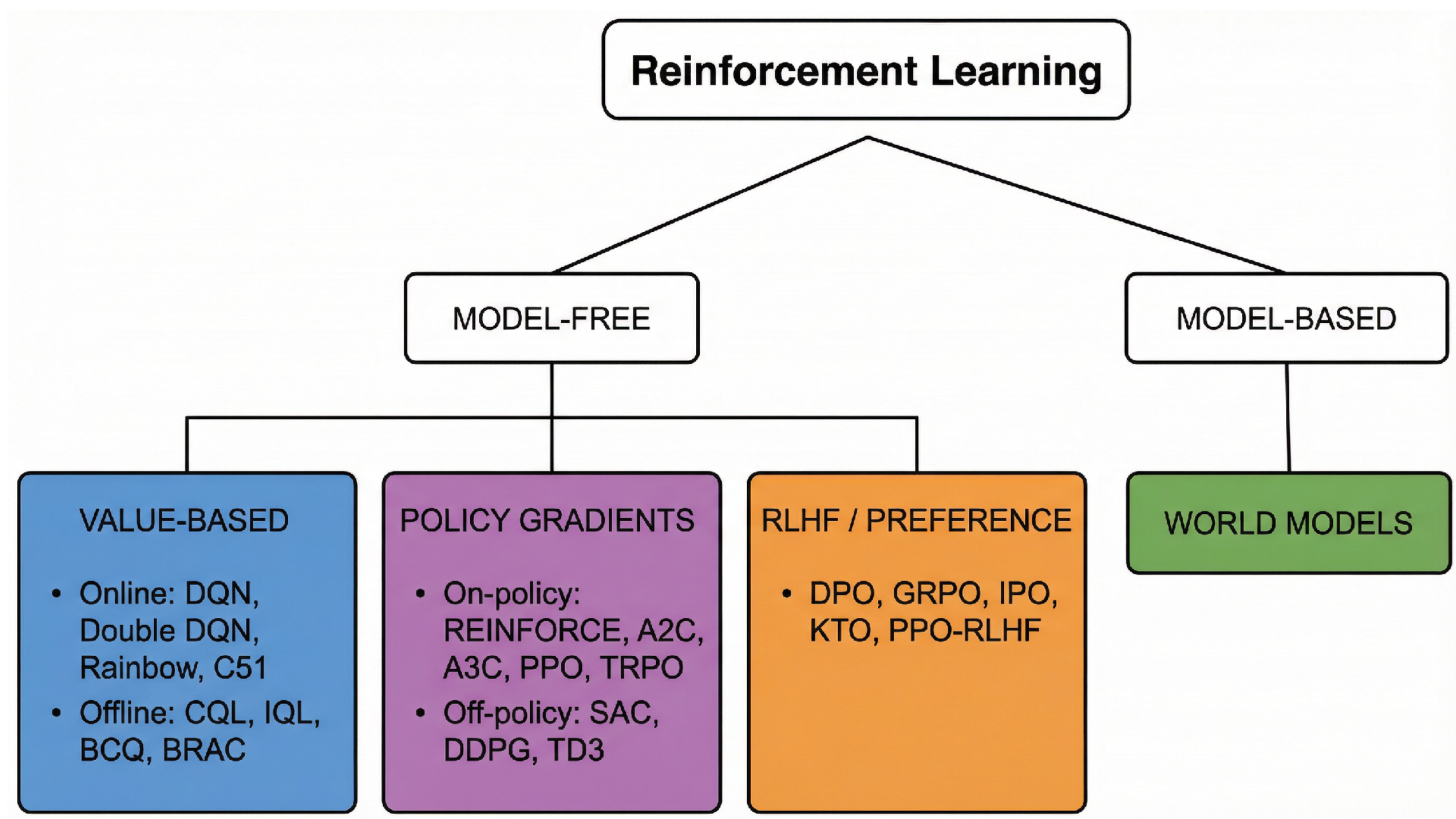
- Deterministic policy gradient

$$\nabla_\theta J \approx \nabla_a Q(s,a)|_{a=\mu(s)} \cdot \nabla_\theta \mu(s)$$

Gradient through Q to improve actor

Key innovations

- ✓ Deterministic policy (continuous actions)
- ✓ Replay buffer (off-policy, sample efficient)
- ✓ Target networks (stability)
- + Exploration noise: $a = \mu(s) + \text{noise}$


Reinforcement Learning
MODEL-FREE
MODEL-BASED
VALUE-BASED
• Online: DQN, Double DQN, Rainbow, C51
• Offline: CQL, IQL, BCQ, BRAC
POLICY GRADIENTS
• On-policy: REINFORCE, A2C, A3C, PPO, TRPO
• Off-policy: SAC, DDPG, TD3
RLHF / PREFERENCE
• DPO, GRPO, IPO, KTO, PPO-RLHF
WORLD MODELS

See you on Monday!