Policy Gradient

Andrew Perrault
perrault.17@osu.edu

CSE 5539

Limitations of value-based RL

Historical #1 limitation: infinite, continuous action spaces (joint angles, forces in robotics)
- Value-based RL must discretize

Policy-based approach can, e.g., output a Gaussian with mean μ, variance σ
- In principle, elegant solution to infinite actions, but not great in practice 😔

Limitations of value-based RL

Modern #1 limitation: handling imitative priors π_{prior}

- In value-based, first need to “convert” prior into implicit policy, i.e., fit a Q or V to that policy

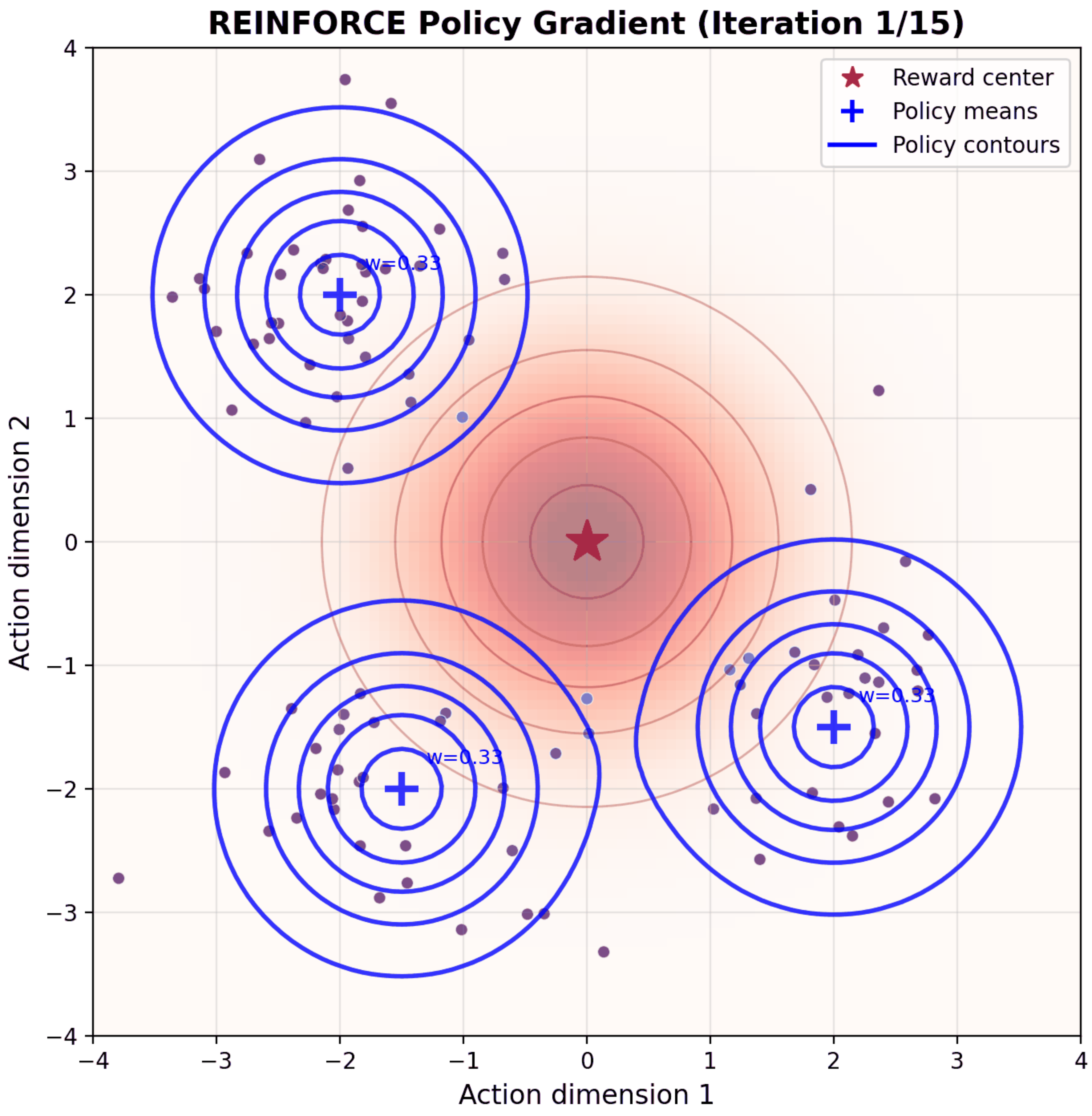
- In policy-based, simply set $\pi = \pi_{prior}$

A supervised learning model can always be interpreted as an imitative prior!

Policy-based deep RL

Improvement loop:

1. Sample from policy π in environment, record trajectories
2. Make good trajectories more probable and bad trajectories less probable



Theory of policy-based RL

Goal: set $\pi_\theta : S \to \Delta A$ to maximize expected return, without estimating Q, V, A

Note: policy parameterized by θ, allows for infinite actions

$J(\pi_\theta) = \mathbb{E}_{s_0,a_0,r_1,\dots}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}\right]$, where $\tau = (s_0, a_0, r_1, \dots)$ is a trajectory sampled from π_θ

Warm up: gradient descent on one-step episode

Single-step case

General: $J(\pi_\theta) = \mathbb{E}_{s_0, a_0, r_1, \ldots} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right]$

Single-step: $J(\pi_\theta) = \mathbb{E}_{s_0, a_0, r_1} r_{t+1} = \sum_{a \in A} \sum_{s \in S} P(s) \pi_\theta(a \mid s) \mathbb{E}[R(s, a)]$. We want:

$\nabla_\theta J(\pi_\theta)$

We assume that $\nabla_\theta \pi_\theta(a \mid s)$ is known, e.g., because π_θ is a neural network

Exercise: MC estimator for $\nabla_\theta(J(\theta))$

Create an **unbiased estimator** for

$\nabla_\theta J(\pi_\theta) = \nabla_\theta \sum_{a \in A} \sum_{s \in S} P(s)\pi_\theta(a \mid s)\mathbb{E}[R(s,a)]$ from a sample of (s, a, r)

Hint: If $x \sim X$, then $f(x)$ is an unbiased estimator for $\mathbb{E}_X[f(X)] = \sum_{x \in X} p(x)f(x)$

Hint: use the log derivative trick:

$$\nabla_\theta \pi_\theta(a \mid s) = \pi_\theta(a \mid s)\frac{\nabla_\theta \pi_\theta(a \mid s)}{\pi_\theta(a \mid s)} = \pi_\theta(a \mid s)\,\nabla_\theta \log(\pi_\theta(a \mid s))$$

MC estimator for $\nabla_\theta(J(\theta))$

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \sum_{a \in A} \sum_{s \in S} P(s)\pi_\theta(a \mid s)\mathbb{E}[R(s,a)]$$

$$= \sum_{s \in S} \sum_{a \in A} P(s)\mathbb{E}[R(s,a)]\, \nabla_\theta \pi_\theta(a \mid s) = \sum_{s \in S} \sum_{a \in A} P(s_0)\mathbb{E}[R(s,a)]\pi_\theta(a \mid s)\, \nabla_\theta \log(\pi_\theta(a \mid s))$$

$$= \sum_{s \in S} \sum_{a \in A} P(s_0)\pi_\theta(a \mid s)\mathbb{E}[R(s,a)]\, \nabla_\theta \log(\pi_\theta(a \mid s)) = \mathbb{E}_{s,a,r}\left[r\, \nabla_\theta \log(\pi_\theta(a \mid s))\right]$$

Estimator: $r\, \nabla_\theta \log(\pi_\theta(a \mid s))$

Log derivative trick allows us to create an extra π_θ outside the gradient

Interpretation: “weighted” MLE

Suppose a were the correct class label for s. The supervised learning update (maximize log prob of action), would be $\nabla_\theta \log \pi_\theta(a \mid s)$

One-step policy gradient update: $r \nabla_\theta \log \pi_\theta(a \mid s)$
- Reward-weighted MLE, where reward sign determines whether log prob is increased or decreased

Single- vs. multi-step

Contextual bandit setting: assume start state is independent of the policy

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \sum_{a \in A} \sum_{s \in S} P(s)\pi_\theta(a \mid s)\mathbb{E}[R(s, a)]$$

- Can pass ∇_θ through $P(s)$

General case: start state distribution is still independent, but future states are not

- If we naively try, e.g., by unrolling, we end up having to different through the Q or V functions 😔

How do we avoid this issues?

Trajectory perspective

Zoom out → *treat θ like an action*

$$J(\pi_\theta) = \mathbb{E}_\tau \left[G(\tau)\right] = \sum_\tau P(\tau \mid \theta) G(\tau)$$

- $P(\tau \mid \theta)$ is the distribution of trajectories given the policy params (**we don't know how to differentiate this yet**)
- $G(\tau)$ is the **return** of trajectory τ, i.e., $\sum_{t=0} \gamma^t R(s_t, a_t, s_{t+1})$ for $\tau = (s_0, a_0, r_1, s_1 \ldots)$

We can use the log derivative trick again: $\nabla_\theta J(\pi_\theta) = \nabla_\theta \sum_\tau P(\tau \mid \theta) G(\tau)$

$$= \sum_\tau G(\tau) P(\tau \mid \theta) \nabla_\theta \log P(\tau \mid \theta) = \mathbb{E}_\tau \left[G(\tau) \nabla_\theta \log P(\tau \mid \theta)\right]$$

Exercise: $\nabla_\theta \log P(\tau \mid \theta)$

How to compute $\nabla_\theta \log P(\tau \mid \theta)$?

Chain rule! Try it—simplify as much as possible

$$\nabla_\theta \log P(\tau \mid \theta) = \nabla_\theta \log \left(P(s_0) \prod_{t=0} \pi(a_t \mid s_t) \prod_{t=0} P(s_{t+1} \mid s_t, a_t) \right)$$

$$= \nabla_\theta \log P(s_0) + \nabla_\theta \sum_{t=0} \log \pi(a_t \mid s_t) + \nabla_\theta \log \sum_{t=0} P(s_{t+1} \mid s_t, a_t)$$

$$= \sum_{t=0} \nabla_\theta \log \pi(a_t \mid s_t)$$

Naive policy gradient (PG) algorithm

$\nabla_\theta J(\pi_\theta) = \nabla_\theta \sum_\tau P(\tau \mid \theta) G(\tau) = \mathbb{E}_\tau \left[G(\tau) \nabla_\theta \log P(\tau \mid \theta) \right]$ (log derivative trick)

$= \mathbb{E}_\tau \left[G(\tau) \sum_{t=0} \nabla_\theta \log \pi(a_t \mid s_t) \right]$ (chain rule)

This is an executable algorithm (**naive PG**). Repeat:

1. Sample τ (or a batch of τ)
2. Take a gradient **ascent** step $\theta \leftarrow \theta + \alpha G(\tau) \sum_{t=0} \nabla_\theta \log \pi(a_t \mid s_t)$

(in practice, Adam and variants are used rather than vanilla SGD)

Intuition: naive PG

$$\theta \leftarrow \theta + \alpha G(\tau) \sum_{t=0} \nabla_\theta \log \pi(a_t \mid s_t)$$

If a trajectory has high reward, make **all** actions in the trajectory more likely

Pros:
1. We have a PG algorithm!
2. Uses all trajectories: makes good trajectories more likely than bad trajectories (intuitive trial and error)

What’s bad about it?

Problems with naive PG

1. *All actions in* τ are upweighted if $G(\tau)$ is good

- Even for the part of $G(\tau)$ that preceded this action 🤔

2. “Racing”: if rewards are positive, all *trajectories get upweighted*

- Bad training dynamics

3. If the learning rate is too high, policy can be destroyed

- Even if the change in θ is small, change in π may be large

1. Causality and REINFORCE

We can just drop the part of the return from the past

- Intuition: at time t, return-so-far is independent of return-to-go given s_t
- Dropping return-so-far has expectation zero, but reduces variance
- If we do this, get the first classic algorithm for PG: REINFORCE

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\tau \left[G(\tau) \sum_{t=0} \nabla_\theta \log \pi(a_t \,|\, s_t) \right] = \mathbb{E}_\tau \left[\sum_{t=0} G(s_t) \, \nabla_\theta \log \pi(a_t \,|\, s_t) \right]$$

2. “Racing”

What function would represent the “right” amount of “reward credit” to give to a_t from s_t?

Answer: the advantage: $A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$

- Intuition: how much more reward in expectation for a rather than a random action from π?
- Why correct? Upweights actions that are *better than average*, downweights actions that are *worse than average* → no more racing!

Recall: the return-to-go $G(s_t)$ is an unbiased estimate of $Q_\pi(s_t, a_t)$

- Just need to subtract $V_\pi(s)$ somehow, without introducing bias 🤔

Subtracting the value function

Let's just try it

$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\tau \left[\sum_{t=0} [G(s_t) - b(s_t)] \nabla_\theta \log \pi(a_t \mid s_t) \right]$ where $b(s_t)$ is an arbitrary function we call a **baseline**

$$= \mathbb{E}_\tau \left[\sum_{t=0} G(s_t) \nabla_\theta \log \pi(a_t \mid s_t) \right] - \mathbb{E}_\tau \left[\sum_{t=0} b(s_t) \nabla_\theta \log \pi(a_t \mid s_t) \right]$$

Left term is the original REINFORCE expectation. Hopefully the right term is zero?

Proof: baselines

$\mathbb{E}_\tau \left[\sum_{t=0} b(s_t) \nabla_\theta \log \pi(a_t \mid s_t) \right]$. Let’s look at a single t

$$\mathbb{E}_\tau \left[b(s_t) \nabla_\theta \log \pi(a_t \mid s_t) \right] = \mathbb{E}_{s_t} \left[\mathbb{E}_{a_t} \left[b(s_t) \nabla_\theta \log \pi(a_t \mid s_t) \right] \right]$$

$= \mathbb{E}_{s_t} \left[\sum_a \pi(a \mid s_t) b(s_t) \nabla_\theta \log \pi(a_t \mid s_t) \right] = \mathbb{E}_{s_t} \left[b(s_t) \sum_a \nabla_\theta \pi(a_t \mid s_t) \right]$ using the reversed log derivative trick

$$= \mathbb{E}_{s_t} \left[b(s_t) \nabla_\theta \sum_a \pi(a_t \mid s_t) \right] = \mathbb{E}_{s_t} \left[b(s_t) \nabla_\theta 1 \right] = 0$$

Actor-critic methods

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\tau \left[\sum_{t=0} [Q_\pi(s_t, a_t) - V_\pi(s_t)] \nabla_\theta \log \pi(a_t \,|\, s_t) \right]$$

- Use the advantage of each target to reduce the variance of policy gradient (as long as baseline is estimated well)

Actor: π, critic: V_π

How to estimate $Q_\pi(s_t, a_t)$ and $V_\pi(s)$?

- Use some combination of networks and MC or TD target (PPO, etc.)
- Use raw MC (no network) for entire advantage (GRPO)
- Using networks → usually more bias (from networks being wrong) in PG update, but reduces variance
- More networks = more memory and compute, but environment interactions

Actor-critic spectrum

MC/pure data (high variance, no bias)

More belief (high bias, less variance)

REINFORCE (w/ baseline):
- MC for $Q_\pi(s, a)$
- no baseline: no $V_\pi(s)$
- w/baseline: uses a critic without introducing bias

GRPO:
- MC for $Q_\pi(s, a)$
- no critic, but tabular MC estimate for $V_\pi(s)$

PPO / GAE:
- GAE averages MC and TD to estimate $Q_\pi(s, a)$
- Learned network critic for $V_\pi(s)$ using MC target

A2C/A3C
- 1-step TD for $Q_\pi(s, a)$ and $V_\pi(s, a)$

3. Avoiding policy destruction

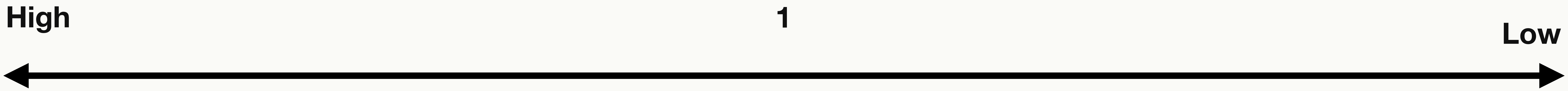
Recall: if the learning rate is too high, policy can be destroyed
- Even if the change in θ is small, change in π may be large

Theoretical solution (*natural policy gradient*): limit KL divergence in π at every update
- Problem: KL divergence is expensive if action space is large

Various hacks designed to “approximate” NPG
- TRPO: check KL for each step, reduce if too high
- PPO: check if $\pi(s, a)$ has changed too much since this trajectory was collected
- GRPO uses PPO-style, but usually data is not reused in practice

RL algorithms are shaped by their environment

Ratio of environment cost / policy cost



High

1

Low

Environment is expensive
- Physical world (e.g., robotics, medicine)
- Expensive sim (fluid dynamics)

Policy is cheap
- Needs to be real-time
- Needs to run on-device

Need to be able to learn off-policy and on old data
- Has a computational cost but worth it

Model-based may be worth it

Algs: DQN, SAC

“Moderate regime”

Cheaper simulations:
Atari, Go, Starcraft

Policy ≈ environment

May not reuse samples

Algs: PPO, A2C

“LLM regime”
- Extremely complicated and large policy
- Environment is just running policy

Compute gradients on an LLM costs 3-4x memory vs. inference

No critic → saves 20-30% memory

Don’t reuse samples

Algs: GRPO

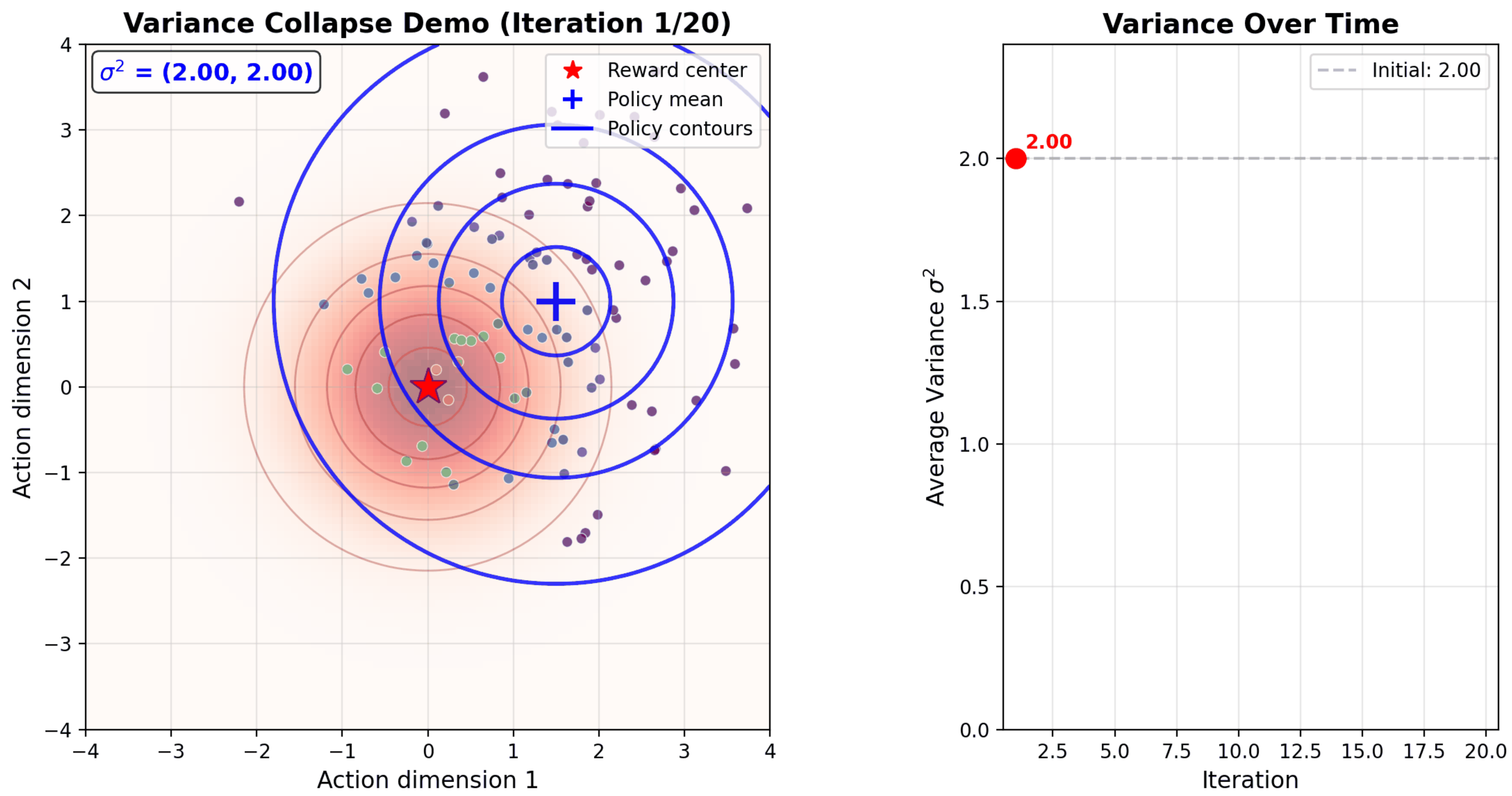
Questions?

Bonus: problems with infinite action PG

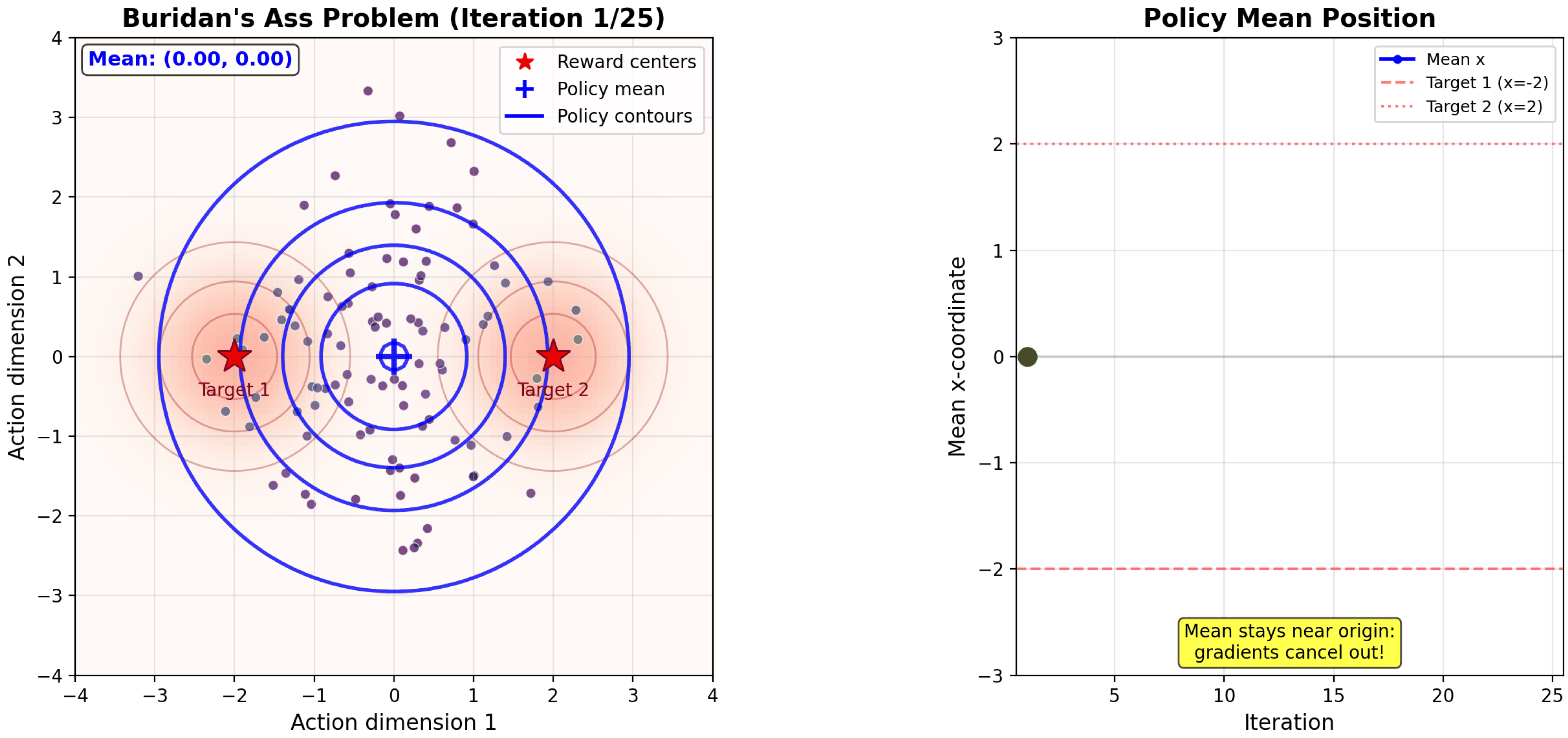
Historical #1 limitation: infinite, continuous action spaces (joint angles, forces in robotics)
- Value-based RL must discretize

Policy-based approach can, e.g., output a Gaussian with mean μ, variance σ
- In principle, elegant solution to infinite actions, but not great in practice 😔

Problems with policy-based: σ collapses



Problems with policy-based: unimodality



Buridan's Ass Problem (Iteration 1/25)

Mean: (0.00, 0.00)

Policy Mean Position

Mean stays near origin:
gradients cancel out!

Discretizing is good?

Actually, discretizing continuous actions turns out to be a good strategy!
- Avoids averaging problem of unimodality
- Can get around precision issues with learned “tokens”
- Theme of last ~5 years: replace regression with classification