Actor Critic

Sanjiban Choudhury

Cornell Bowers CIS
Computer Science

Vanilla REINFORCE

Start with an arbitrary initial policy $\pi_\theta(a \mid s)$

while *not converged* **do**

Roll-out $\pi_\theta(a \mid s)$ to collect trajectories $D = \{s_0^i, a_0^i, r_0^i, \ldots, s_{T-1}^i, a_{T-1}^i, r_{T-1}^i\}_{i=1}^N$

Compute reward-to-go for each timestep for each trajectory $\hat{Q}^{\pi_\theta}(s_t^i, a_t^i) = \sum_{t'=t}^{T-1} r(s_{t'}^i, a_{t'}^i)$

Compute gradient

$$\nabla_\theta J(\theta) = \frac{1}{N}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\, \hat{Q}^{\pi_\theta}(s_t^i, a_t^i)\right]$$

Update parameters $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Three major nightmares with policy gradients

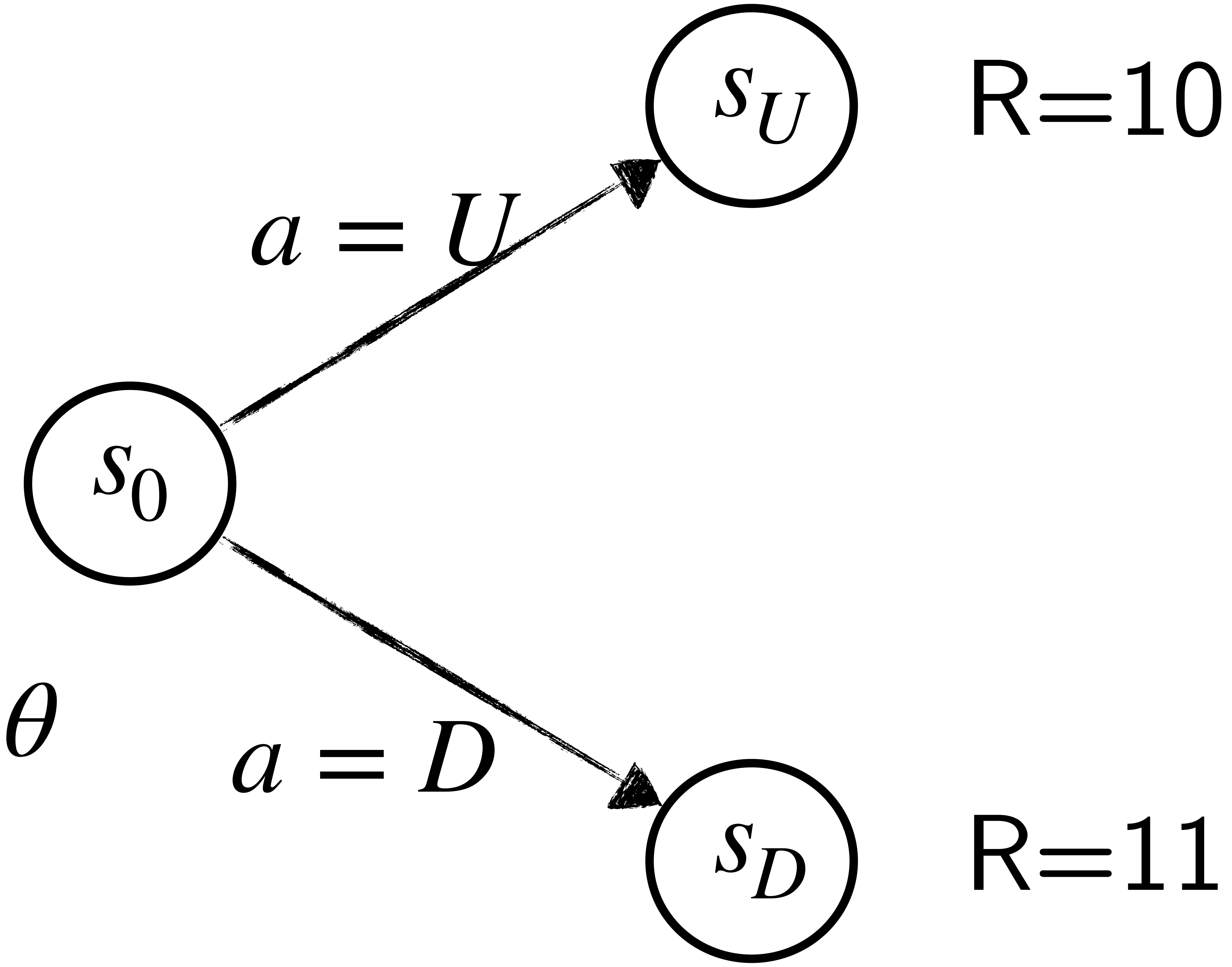
Nightmare 1:

High Variance

Consider the following MDP

$\pi_\theta(a = U \mid s_0) = \theta$

$\pi_\theta(a = D \mid s_0) = 1 - \theta$



Suppose we init $\theta = 0.5$, and draw 4 samples with our policy

And then apply PG

When Q values for all rollouts in a batch are high?

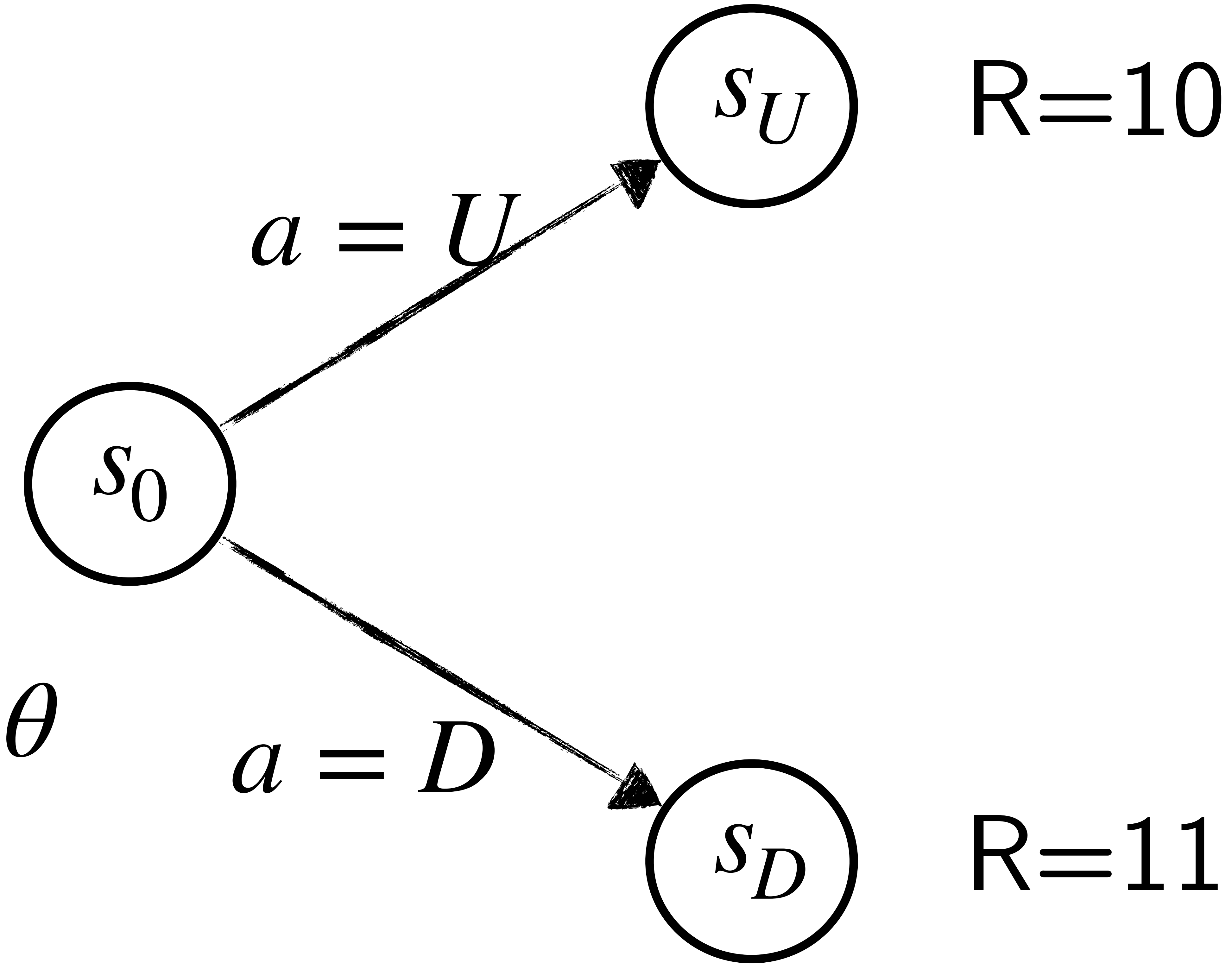
$$\boxed{\nabla_\theta J \; = \; E_{s\sim d^{\pi_\theta}(s), a\sim\pi_\theta(a|s)} \left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)\right]}$$

Recall that one of the reasons for the high variance is that the algorithm does not know how well the trajectories perform compared to other trajectories. Therefore, by introducing a baseline for the total reward (or reward to go), we can update the policy based on how well the policy performs compared to a baseline

Solution: Subtract a baseline!

$\pi_\theta(a = U \,|\, s_0) = \theta$

$\pi_\theta(a = D \,|\, s_0) = 1 - \theta$



Suppose we subtracted of $V^\pi(s_0) = 10.5$ from the reward to go

$$\nabla_\theta J = E_{d^{\pi_\theta}(s)} E_{\pi_\theta(a|s)} \left[\nabla_\theta \log(\pi_\theta(a|s) \left(Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \right) \right]$$

Solution: Subtract a baseline!

$$\nabla_\theta J = E_{d^{\pi_\theta}(s)} E_{\pi_\theta(a|s)} \left[\nabla_\theta \log(\pi_\theta(a|s) \left(Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \right) \right].$$

We can prove that this does not change the gradient

Solution: Subtract a baseline!

$$\nabla_\theta J = E_{d^{\pi_\theta}(s)} E_{\pi_\theta(a|s)} \left[\nabla_\theta \log(\pi_\theta(a|s) \left(Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \right) \right].$$

We can prove that this does not change the gradient

$$\nabla_\theta J = E_{d^{\pi_\theta}(s)} E_{\pi_\theta(a|s)} \left[\nabla_\theta \log(\pi_\theta(a|s) A^{\pi_\theta}(s,a) \right]$$

But turns Q values into advantage (which is lower variance)

Solution: Subtract a baseline!

$$\nabla_\theta J = E_{d^{\pi_\theta}(s)} E_{\pi_\theta(a|s)} \left[\nabla_\theta \log(\pi_\theta(a|s) \left(Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \right) \right].$$

We can prove that this does not change the gradient

$$\nabla_\theta J = E_{d^{\pi_\theta}(s)} E_{\pi_\theta(a|s)} \left[\nabla_\theta \log(\pi_\theta(a|s) A^{\pi_\theta}(s,a) \right]$$

But turns Q values into advantage (which is lower variance)

Can we justify this move using the PDL?

Vanilla REINFORCE

Start with an arbitrary initial policy $\pi_\theta(a \mid s)$

while *not converged* **do**

Roll-out $\pi_\theta(a \mid s)$ to collect trajectories $D = \{s_0^i, a_0^i, r_0^i, \ldots, s_{T-1}^i, a_{T-1}^i, r_{T-1}^i\}_{i=1}^N$

Compute reward-to-go for each timestep for each trajectory $\hat{Q}^{\pi_\theta}(s_t^i, a_t^i) = \sum_{t'=t}^{T-1} r(s_{t'}^i, a_{t'}^i)$

Compute gradient

$$\nabla_\theta J(\theta) = \frac{1}{N} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\ \hat{Q}^{\pi_\theta}(s_t^i, a_t^i) \right]$$

Update parameters $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Fix #1: Subtract baseline

Start with an arbitrary initial policy $\pi_\theta(a \mid s)$

while *not converged* **do**

Roll-out $\pi_\theta(a \mid s)$ to collect trajectories $D = \{s_0^i, a_0^i, r_0^i, \ldots, s_{T-1}^i, a_{T-1}^i, r_{T-1}^i\}_{i=1}^N$

Compute reward-to-go for each timestep for each trajectory $\hat{Q}^{\pi_\theta}(s_t^i, a_t^i) = \sum_{t'=t}^{T-1} r(s_{t'}^i, a_{t'}^i)$

Fit value function $\hat{V}^{\pi_\theta}(s_t^i) \approx \sum_{t'=t}^{T-1} r(s_{t'}^i, a_{t'}^i)$

How??

Compute advantage $\hat{A}^{\pi_\theta}(s_t^i, a_t^i) = \hat{Q}^{\pi_\theta}(s_t^i, a_t^i) - \hat{V}^{\pi_\theta}(s_t^i)$

Compute gradient

$$\nabla_\theta J(\theta) = \frac{1}{N}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\, \hat{A}^{\pi_\theta}(s_t^i, a_t^i)\right]$$

Update parameters $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Two ways to fit critic

Monte-Carlo

$$\left(V(s_t) - \sum_{t'=t}^{T} \gamma^{t'-t} r_t' \right)^2$$

Needs full time-horizon trajectories

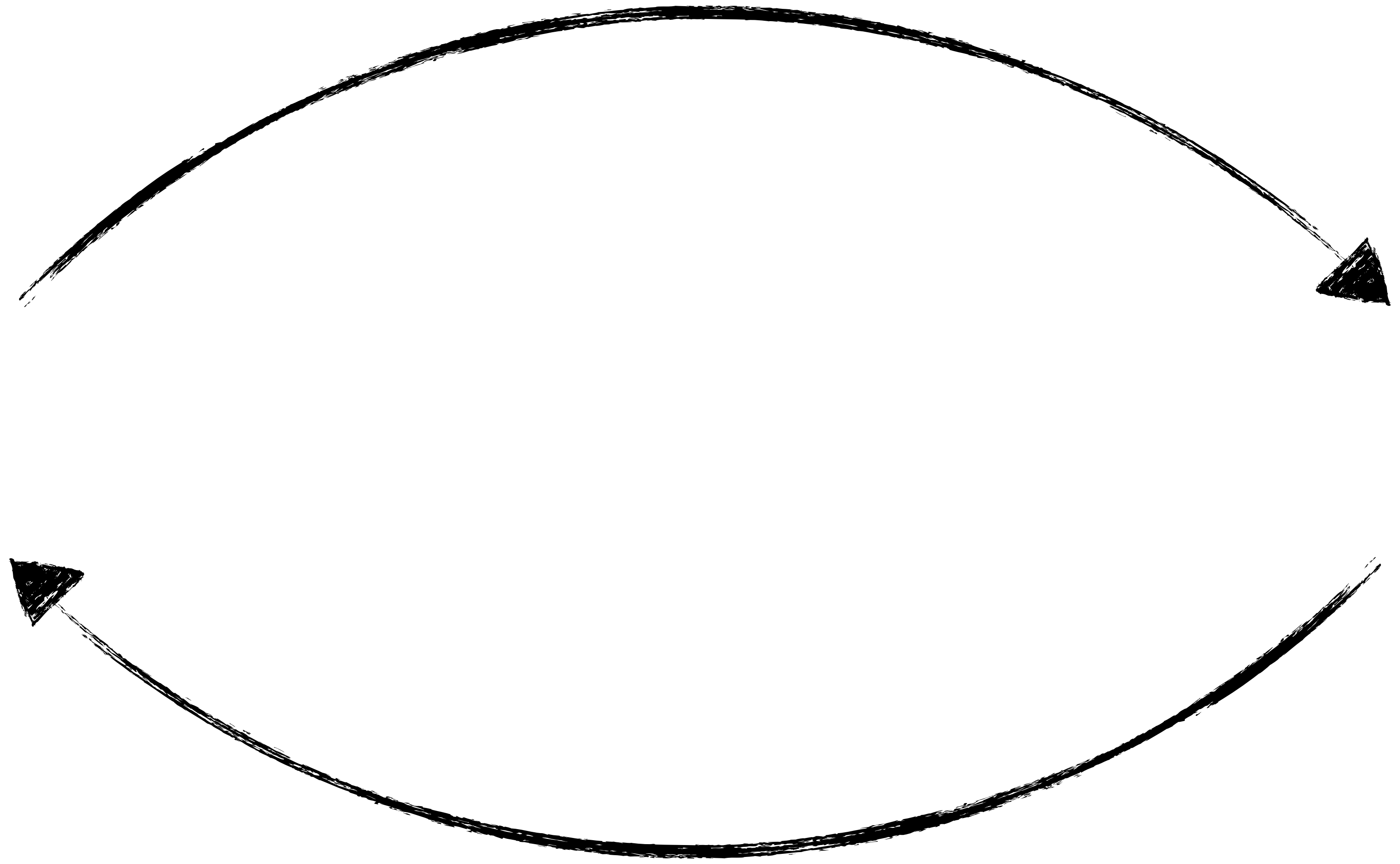
Temporal Difference

$$\Big(V(s_t) - \big[r_t + \gamma V(s_t) \big] \Big)^2$$

Works with partial segments! (s,a,r,s')

Actor-Critic Framework

Actor



Critic

Policy improvement of π

Estimates value functions V^{π}

Actor-Critic Framework (Infinite Horizon)

Start with an arbitrary initial policy $\pi_\theta(a \mid s)$

while *not converged* **do**

Roll-out $\pi_\theta(a \mid s)$ to collect trajectories $D = \{s^i, a^i, r^i, s^i_+\}_{i=1}^N$

Fit value function $\hat{V}^{\pi_\theta}(s^i)$ using TD, i.e. minimize $(r^i + \gamma \hat{V}^{\pi_\theta}(s^i_+) - \hat{V}^{\pi_\theta}(s^i))^2$

Compute advantage $\hat{A}^{\pi_\theta}(s^i, a^i) = r(s^i, a^i) + \gamma \hat{V}^{\pi_\theta}(s^i_+) - \hat{V}^{\pi_\theta}(s^i)$

Compute gradient

$$\nabla_\theta J(\theta) = \frac{1}{N}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a^i_t \mid s^i_t)\, \hat{A}^{\pi_\theta}(s^i, a^i)\right]$$

Update parameters $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Important Actor Critic Algorithms

1. Soft Actor-Critic
   - Stochastic policy
   - Off-policy algorithm
   - Adds entropy to reward to encourage exploration

2. TD3
   - Deterministic policy
   - Off-policy algorithm
   - Trains two Q networks to combat overestimation

3. PPO
   - Stochastic policy
   - On-policy algorithm
   - We will cover this next!

Demonstrating a Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning

Laura Smith* [1], Ilya Kostrikov* [1], Sergey Levine[1]
*Equal contribution [1]Berkeley AI Research, UC Berkeley
{smithlaura, kostrikov}@berkeley.edu, svlevine@eecs.berkeley.edu

Details

State-Action Space: The state is 30 dimensional containing the joint positions (12 values), joint velocities (12 values), roll and pitch of the torso and binary foot contact indicators (4 values). The action space is 12 dimensional corresponding to the target joint position for the 12 robot joints. The predicted target joint angles $a = \hat{\mathbf{q}} \in \mathbb{R}^{12}$ is converted to torques $\boldsymbol{\tau}$ using a PD controller with target joint velocities set to 0.

Reward:

$$r(s,a) = r_v(s,a) - 0.1 v_{yaw}^2$$

where v_{yaw} is an angular yaw velocity and

$$r_v(s,a) = \begin{cases} 1, & \text{for } v_x \in [v_t, 2v_t] \\ 0, & \text{for } v_x \in (-\infty, -v_t] \cup [4v_t, \infty) \\ 1 - \frac{|v_x - v_t|}{2v_t}, & \text{otherwise.} \end{cases}$$

Algorithm:

Soft Actor-Critic

Nightmare 2:

Distribution Shift

What happens if your step-size is large?

$$\nabla_\theta J = E_{d^{\pi_\theta}(s)} E_{\pi_\theta(a|s)} \left[\nabla_\theta \log(\pi_\theta(a|s) A^{\pi_\theta}(s,a) \right]$$

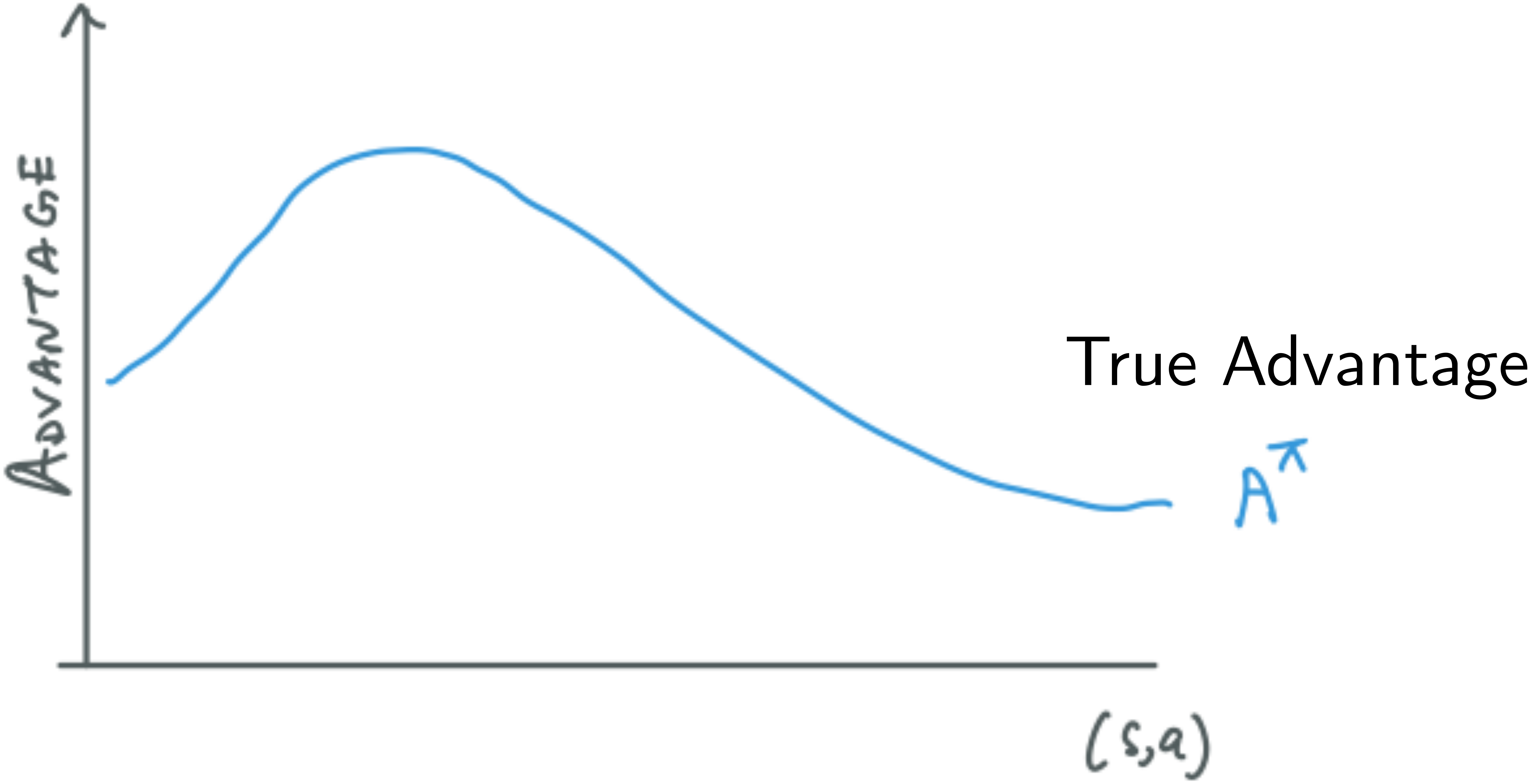
What happens if your step-size is large?

$$\nabla_\theta J = E_{d^{\pi_\theta}(s)} E_{\pi_\theta(a|s)} \left[\nabla_\theta \log(\pi_\theta(a|s) \cancel{A^{\pi_\theta}}(s,a)\right]$$

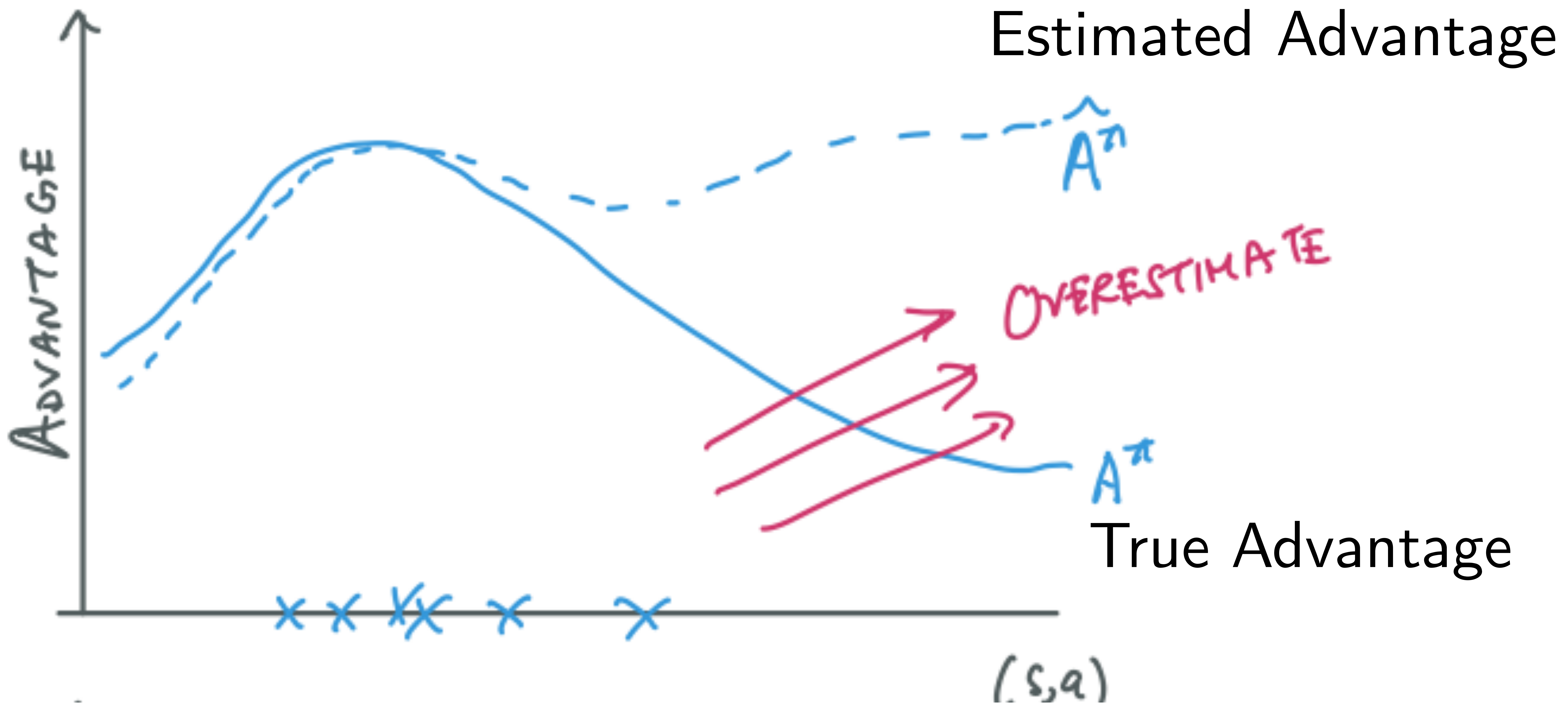
$$\hat{A}^{\pi_\theta}(s,a)$$

We are *estimating* the advantage from roll-outs

The problem of distribution shift

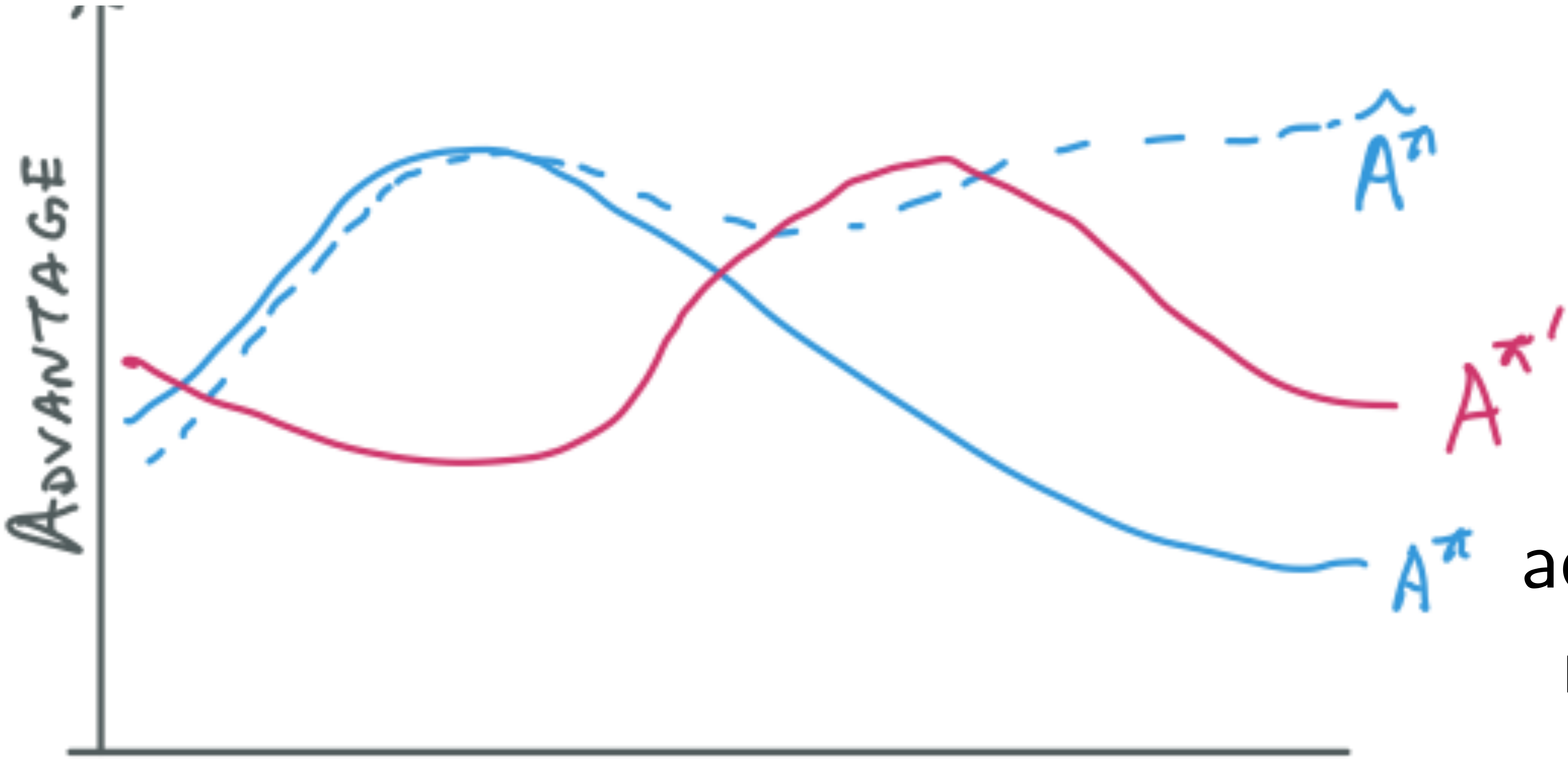


The problem of distribution shift



Our new policy wants to go all the way to the RIGHT

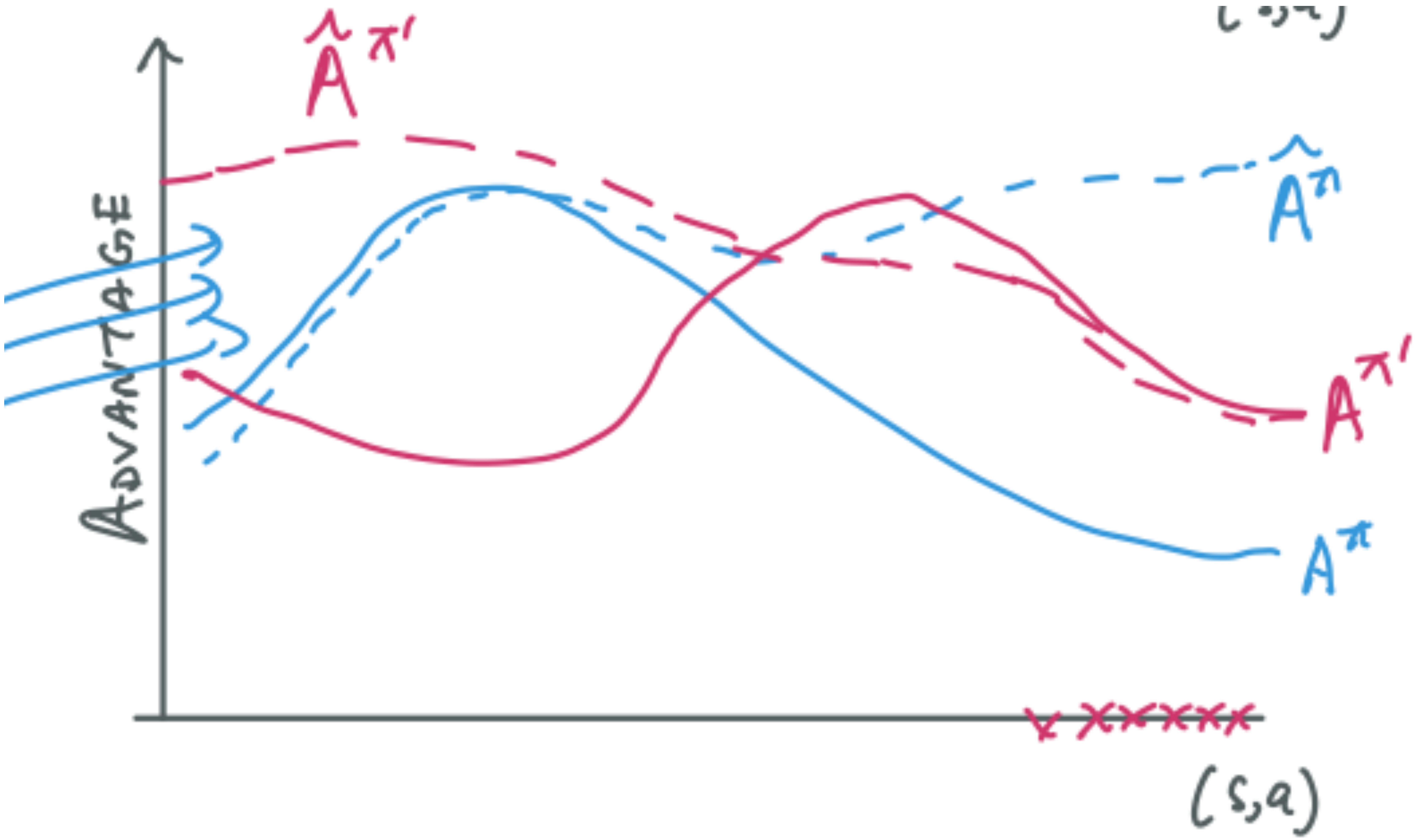
The problem of distribution shift



True advantage of new policy

The problem of distribution shift

Estimated Advantage



True advantage of new policy

Our new policy wants to go all the way to the LEFT

Recap: Problem with Approximate Policy Iteration

$$V^{\pi^+}(s_0) - V^{\pi}(s_0) = \sum_{t=0}^{T-1} \mathbb{E}_{s_t \sim d_t^{\pi^+}} A^{\pi}(s_t, \pi^+)$$

PDL requires accurate Q_θ^π on states that π^+ will visit! ($d_t^{\pi^+}$)

But we only have states that π visits (d_t^π)

If π^+ changes drastically from π, then $|d_t^{\pi^+} - d_t^\pi|$ is big!

Be stable

Slowly change policies

Keep $d_t^{\pi^+}$ close to d_t^{π}

Goal: Change distributions slowly

$$\max_{\Delta\theta} J(\theta + \Delta\theta)$$

s.t. $d^{\pi_{\theta+\Delta\theta}}$ is close to d^{π_θ}

How do we measure distance between distributions?

Goal: Change distributions slowly

$$\max_{\Delta\theta} J(\theta + \Delta\theta)$$

s.t. $KL(d^{\pi_{\theta+\Delta\theta}} || d^{\pi_\theta}) \leq \epsilon$

Fix #2: Take small steps

Start with an arbitrary initial policy $\pi_\theta(a \mid s)$

while *not converged* **do**

Roll-out $\pi_\theta(a \mid s)$ to collect trajectories $D = \{s^i, a^i, r^i, s^i_+\}_{i=1}^N$

Fit value function $\hat{V}^{\pi_\theta}(s^i)$ using TD, i.e. minimize $(r^i + \gamma \hat{V}^{\pi_\theta}(s^i_+) - \hat{V}^{\pi_\theta}(s^i))^2$

Compute advantage $\hat{A}^{\pi_\theta}(s^i, a^i) = r(s^i, a^i) + \gamma \hat{V}^{\pi_\theta}(s^i_+) - \hat{V}^{\pi_\theta}(s^i)$

Compute gradient

$$\nabla_\theta J(\theta) = \frac{1}{N} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a^i_t \mid s^i_t)\, \hat{A}^{\pi_\theta}(s^i, a^i) \right]$$

s.t. $KL(\pi(\theta + \Delta\theta) \,||\, \pi(\theta)) \leq \epsilon$

Update parameters $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

How??

Natural Gradient Descent (rediscovered as TRPO)

Start with an arbitrary initial policy $\pi_\theta(a\,|\,s)$

while *not converged* **do**

Roll-out $\pi_\theta(a\,|\,s)$ to collect trajectories $D = \{s^i, a^i, r^i, s_+^i\}_{i=1}^N$

Fit value function $\hat{V}^{\pi_\theta}(s^i)$ using TD, i.e. minimize $(r^i + \gamma \hat{V}^{\pi_\theta}(s_+^i) - \hat{V}^{\pi_\theta}(s^i))^2$

Compute advantage $\hat{A}^{\pi_\theta}(s^i, a^i) = r(s^i, a^i) + \gamma \hat{V}^{\pi_\theta}(s_+^i) - \hat{V}^{\pi_\theta}(s^i)$

Compute gradient

$$\nabla_\theta J(\theta) = \frac{1}{N}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i\,|\,s_t^i)\, \hat{A}^{\pi_\theta}(s^i, a^i)\right]$$

$$\cancel{\text{s.t. } KL(\pi(\theta+\Delta\theta)\,||\,\pi(\theta)) \le \epsilon}$$

$$\approx \Delta\theta^T G(\theta)\Delta\theta \le \epsilon$$

Update parameters $\quad \theta \leftarrow \theta + \alpha G(\theta)^{-1} \nabla_\theta J(\theta)$

$G(\theta)$ is Fischer Information Matrix

$$G(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta \nabla_\theta \log \pi_\theta^T\right]$$

Proximal Policy Optimization (PPO)

Computing Fischer Information matrix is expensive and slow!

Idea: Instead of taking small steps, change the loss function so there is no benefit in taking large steps!

Proximal Policy Optimization (PPO)

Computing Fischer Information matrix is expensive and slow!

Idea: Instead of taking small steps, change the loss function so there is no benefit in taking large steps!

Instead of defining gradient, we will define a surrogate loss function (Lets say we are at iteration k)

$$\mathscr{L}(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta}{\pi_{\theta_k}} A^{\pi_{\theta_k}}(s,a)\right]$$

Proximal Policy Optimization (PPO)

Computing Fischer Information matrix is expensive and slow!

Idea: Instead of taking small steps, change the loss function so there is no benefit in taking large steps!

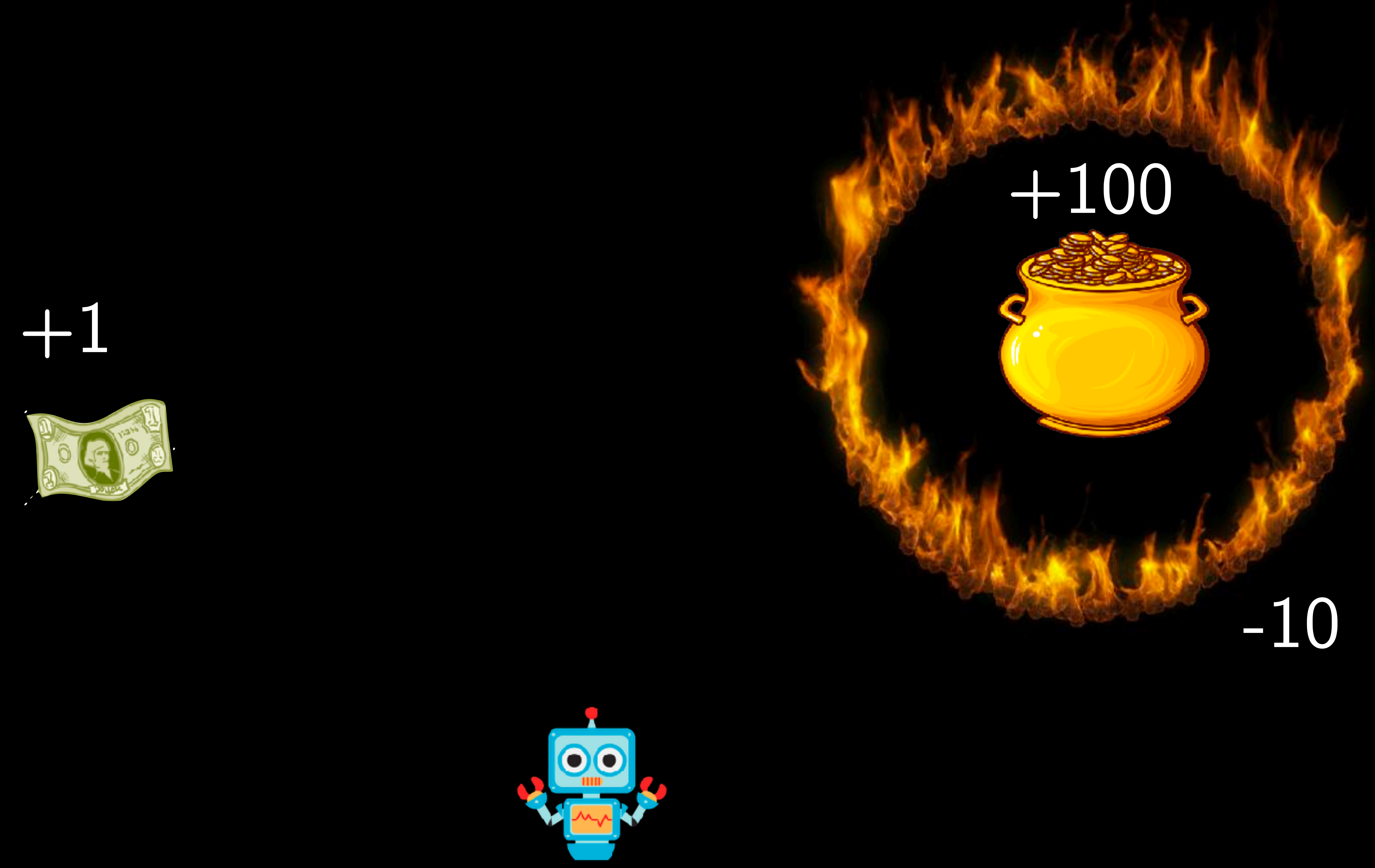
Clip the loss if the policy π_θ deviates too much from π_{θ_k}

$$\mathscr{L}(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_k}}\left[\min\left(\frac{\pi_\theta}{\pi_{\theta_k}}A^{\pi_{\theta_k}}(s,a), \text{clip}\left(\frac{\pi_\theta}{\pi_{\theta_k}}, 1-\epsilon, 1+\epsilon\right)A^{\pi_{\theta_k}}(s,a)\right)\right]$$

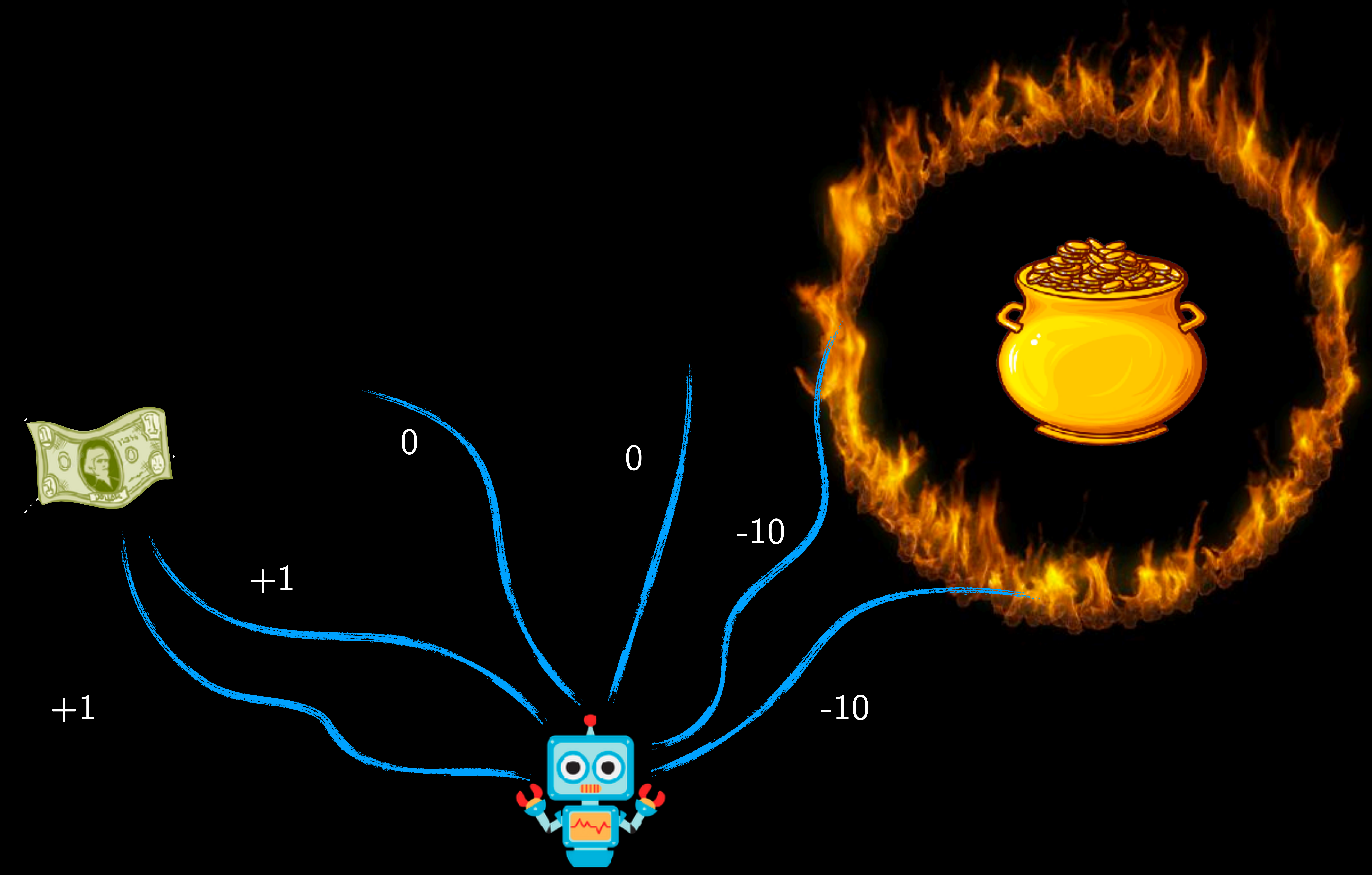
Nightmare 3:

Local Optima

The Ring of Fire

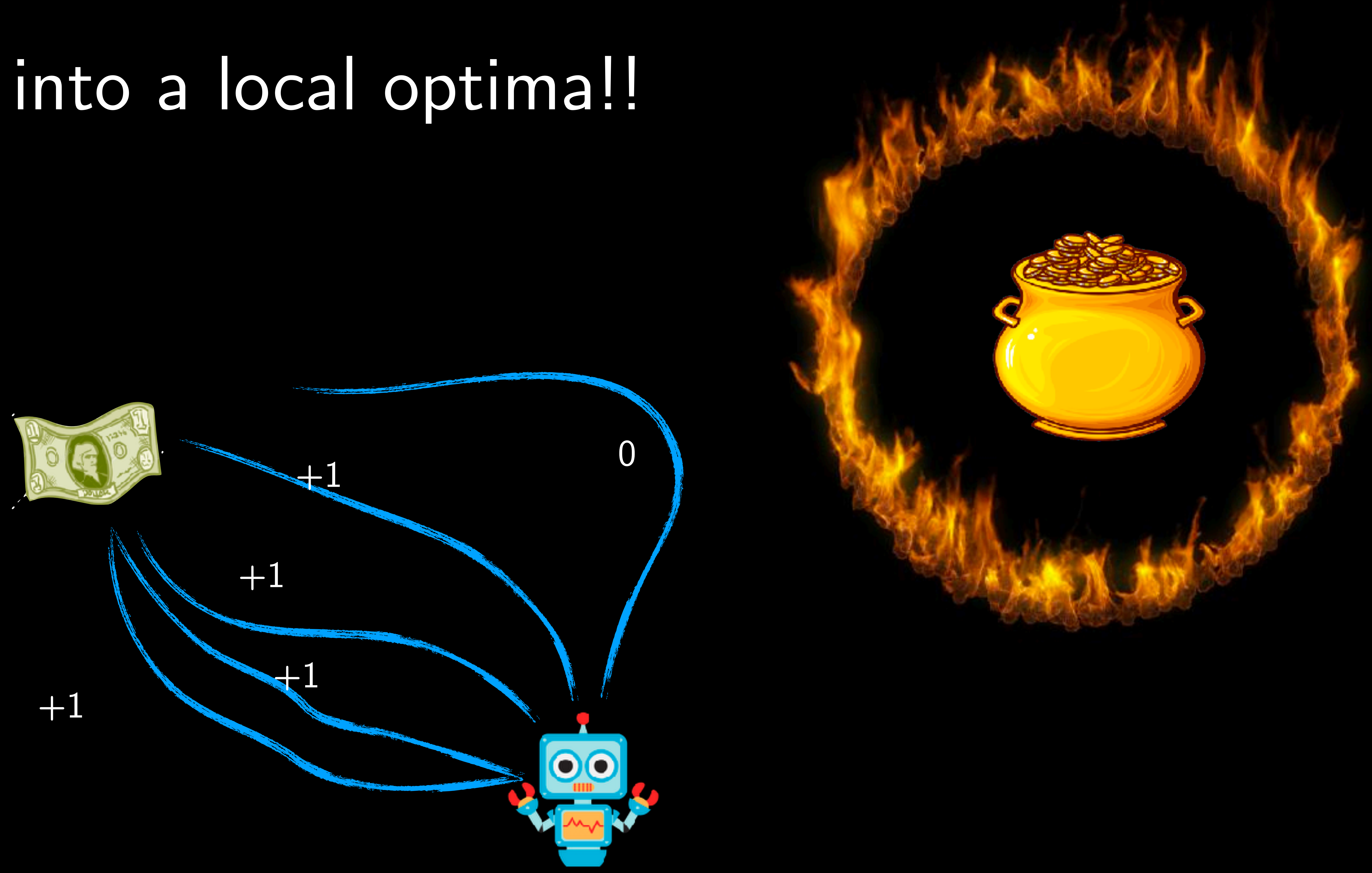


The Ring of Fire



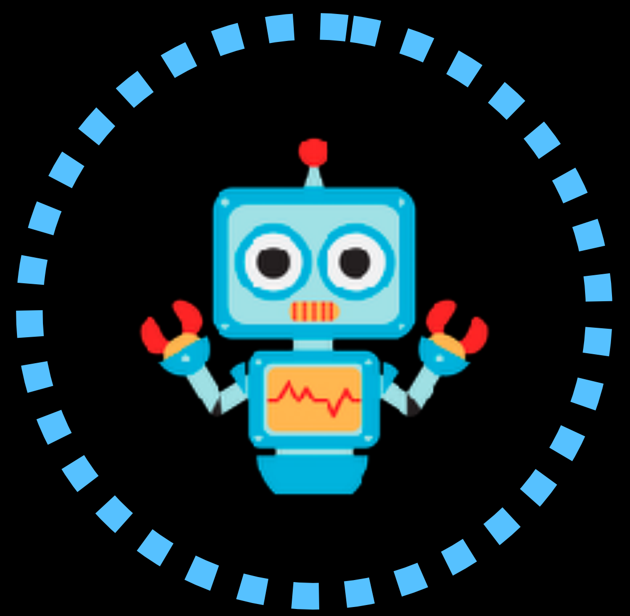
The Ring of Fire

Get's sucked into a local optima!!

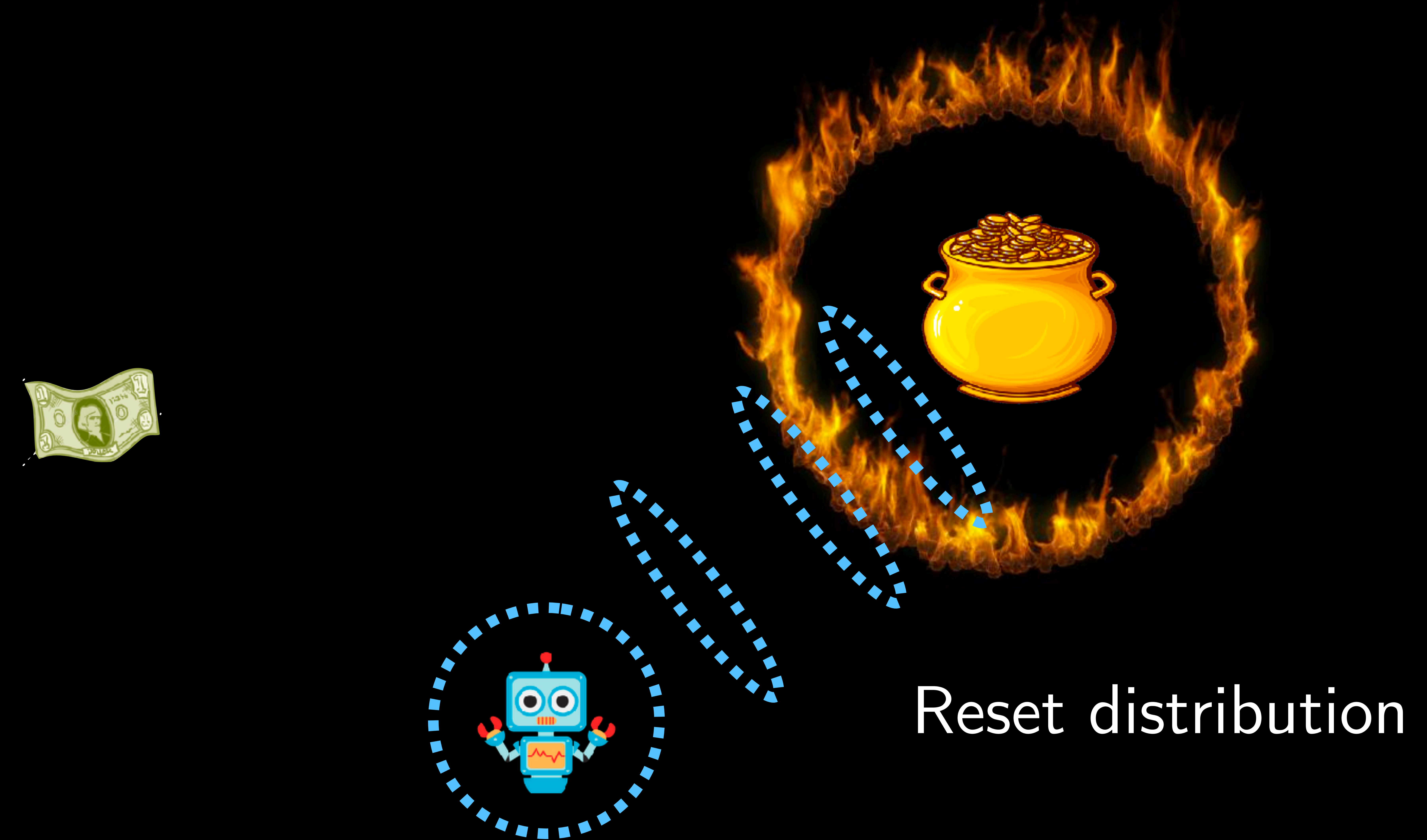


Idea: What if we had a “good reset distribution?”

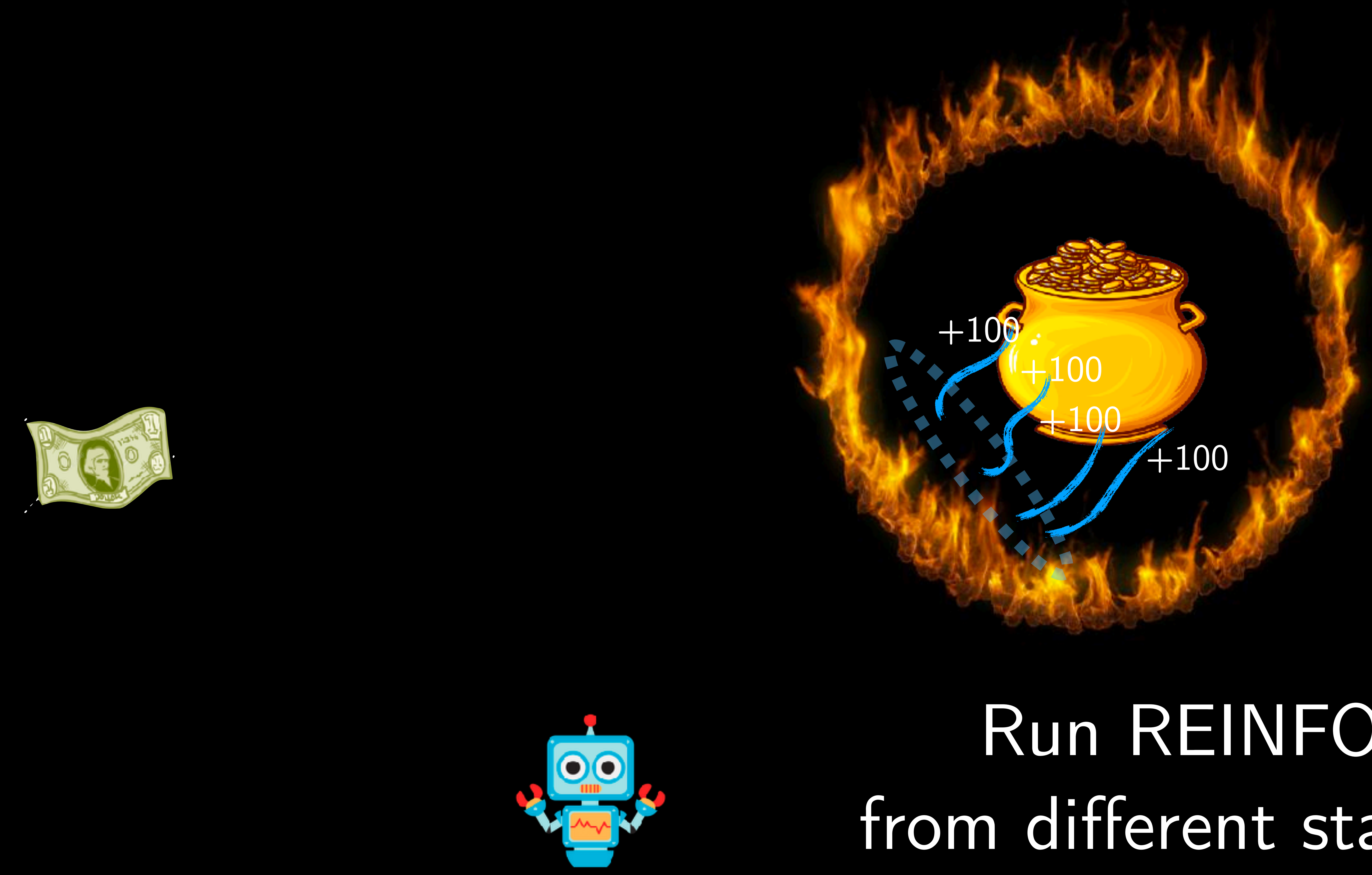
Start distribution



Idea: What if we had a “good reset distribution?”

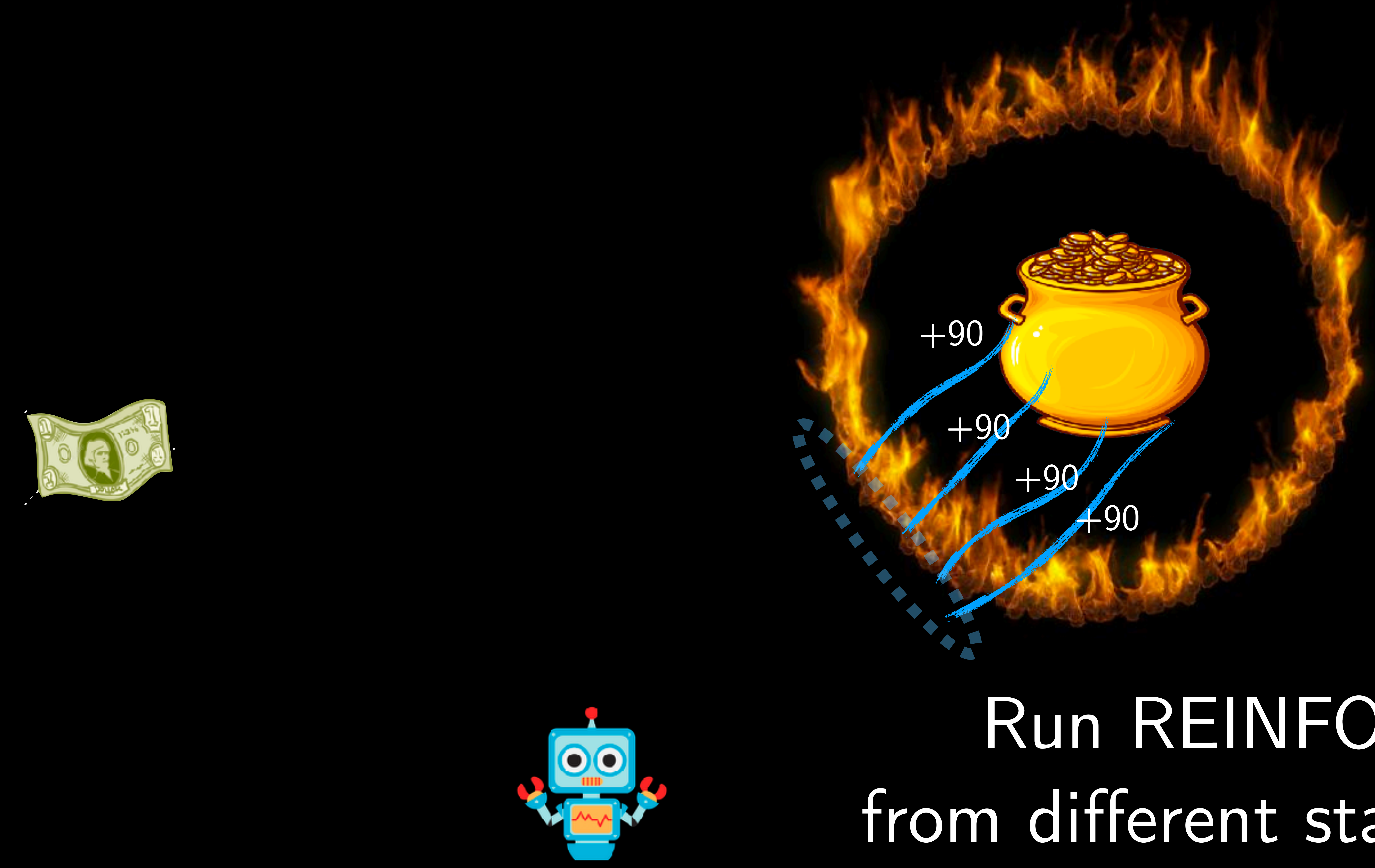


Idea: What if we had a "good reset distribution?"



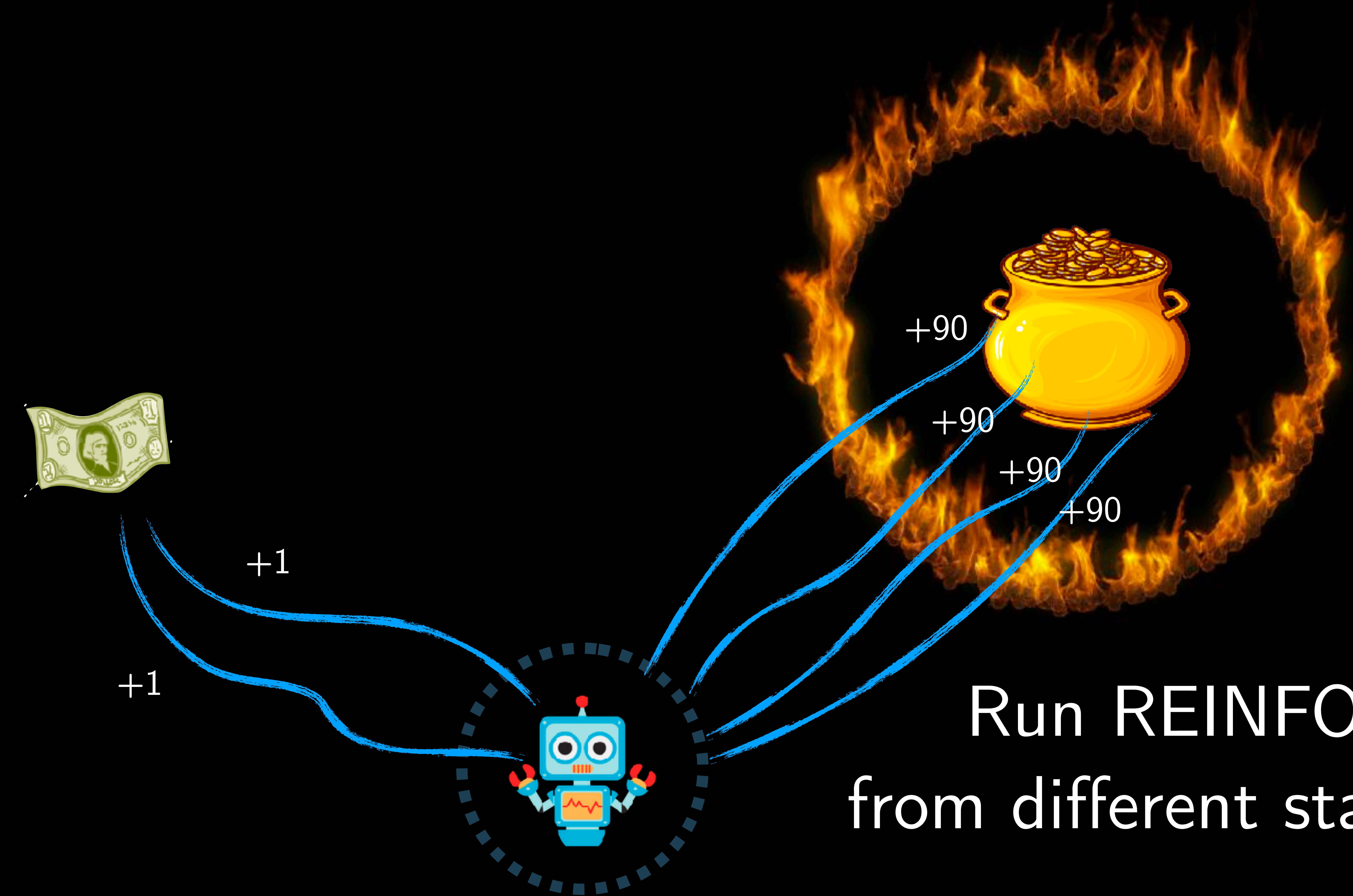
Run REINFORCE
from different start states

Idea: What if we had a “good reset distribution?”



Run REINFORCE
from different start states

Idea: What if we had a "good reset distribution?"



Run REINFORCE
from different start states

Fix #3: Use a reset distribution

Start with an arbitrary initial policy $\pi_\theta(a \mid s)$

while *not converged* **do**

Roll-out $\pi_\theta(a \mid s)$ to collect trajectories $D = \{s^i, a^i, r^i, s_+^i\}_{i=1}^N$

Fit value function $\hat{V}^{\pi_\theta}(s^i)$ using TD, i.e. minimize $(r^i + \gamma \hat{V}^{\pi_\theta}(s_+^i) - \hat{V}^{\pi_\theta}(s^i))^2$

Compute advantage $\hat{A}^{\pi_\theta}(s^i, a^i) = r(s^i, a^i) + \gamma \hat{V}^{\pi_\theta}(s_+^i) - \hat{V}^{\pi_\theta}(s^i)$

Compute gradient

$$\nabla_\theta J(\theta) = \frac{1}{N}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\, \hat{A}^{\pi_\theta}(s^i, a^i)\right] \quad \text{s.t. } KL(\pi(\theta + \Delta\theta) \,||\, \pi(\theta)) \leq \epsilon$$

Update parameters $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Instead of rolling out from the start state, rollout from states expert visits