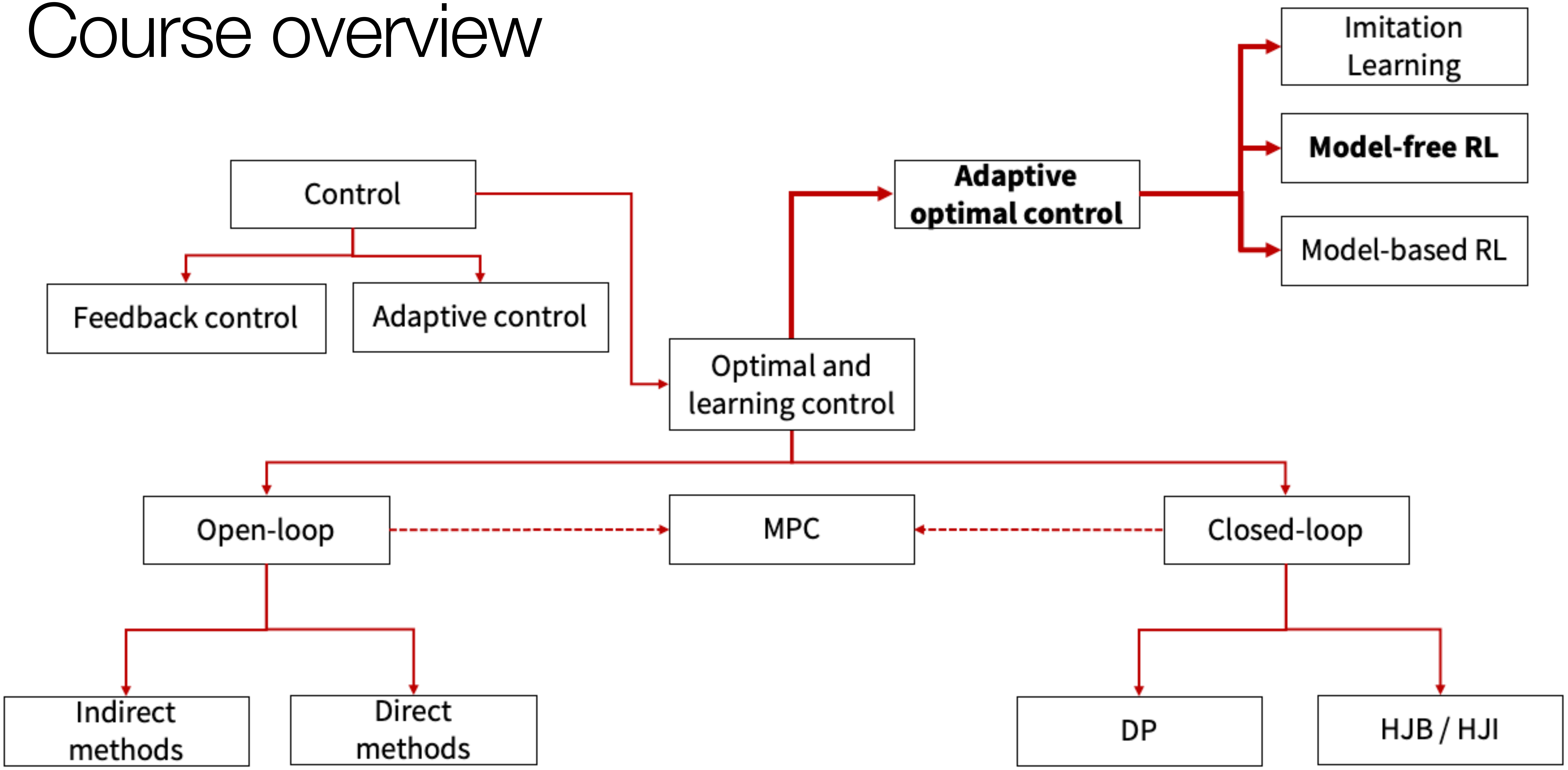
AA203
Optimal and Learning-based Control

Model-free Reinforcement Learning: Policy Optimization

Stanford University

Autonomous Systems Lab

Course overview



A taxonomy of RL

RL Algorithms

do not use dynamics $T(x_{t+1} \mid x_t, u_t)$

Model-free

Policy optimization

directly maximize the RL objective

$$\mathbb{E}_{\tau \sim p_\pi(\tau)}\left[\sum_{t=0}^{H} \gamma^t r\left(x_t, u_t\right)\right]$$

Value-based

policy implicitly defined via

$V(x)$ or $Q(x, u)$

set $\pi\left(\mathbf{x}_t\right) = \arg\max_u Q\left(\mathbf{x}_t, \mathbf{u}_t\right)$

Outline

Intro to policy gradients

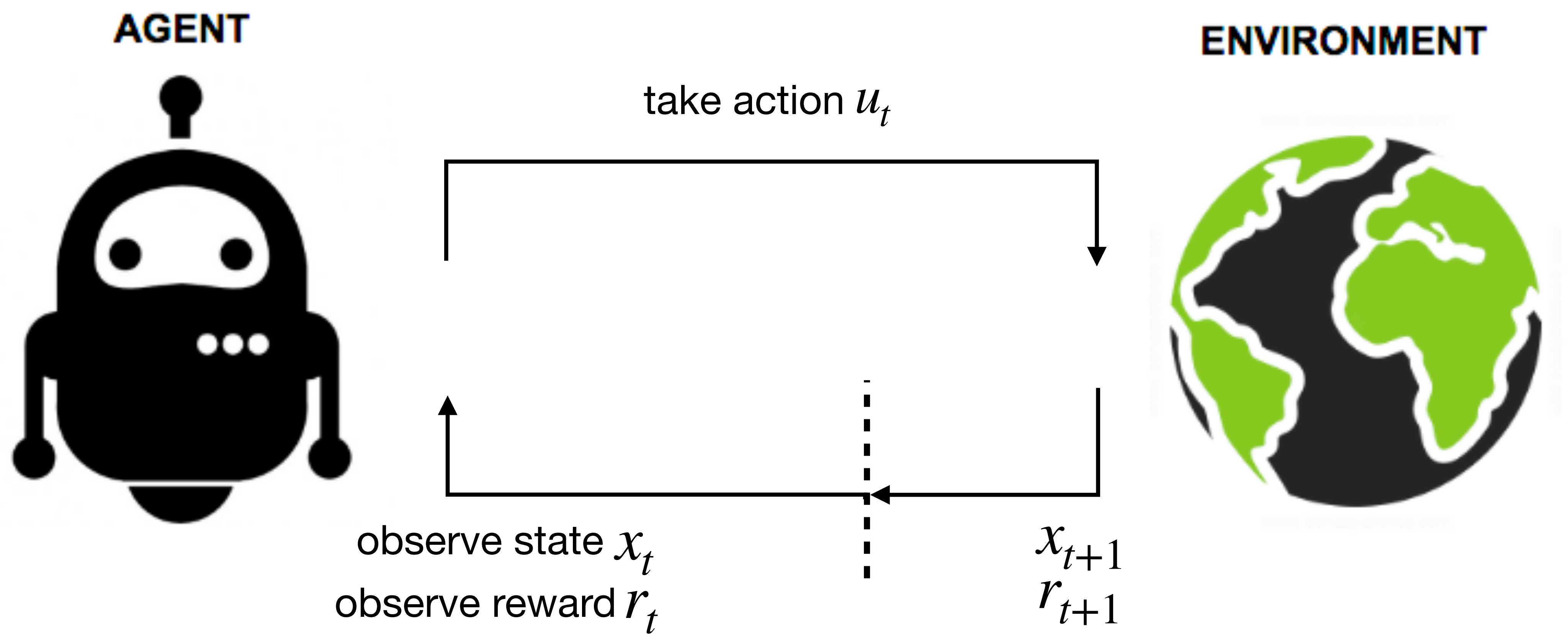
- REINFORCE algorithm
- Reducing variance of policy gradient

Actor-Critic methods

- Advantage
- Architecture design

Deep RL Algorithms & Applications

The goal of reinforcement learning



- The agent interacts with the environment to generate trajectories $\tau = (x_0, u_0, x_1, u_1, \ldots x_T)$
- We define the trajectory distribution

$$p(x_0, u_0, \ldots, x_T) = p(\tau) = p(x_0) \prod_{t=1}^{T} \pi(u_t \mid x_t) p(x_{t+1} \mid x_t, u_t)$$

- We can express the RL objective as an expectation under the trajectory distribution

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t \geq 0} \gamma^t R\left(x_t, u_t\right) \right]$$

Policy Optimization

- In policy optimization, we care about learning an (explicit) parametric policy π_θ, with parameters θ
- In light of this, we can re-write the Eqs from the previous slide w.r.t. θ:

$$p(x_0, u_0, \ldots, x_T) = p(\tau) = p(x_0)\prod_{t=1}^{T} \pi_\theta(u_t \mid x_t) p(x_{t+1} \mid x_t, u_t)$$

$$\theta^* = \arg\max_\theta \underbrace{\mathbb{E}_{\tau \sim p(\tau)}\left[\sum_{t \geq 0} \gamma^t R\left(x_t, u_t\right)\right]}_{J(\theta)}$$

To simplify the notation, we'll ignore discounting for now $(\gamma = 1)$ and consider

$$\theta^* = \arg\max_\theta \underbrace{\mathbb{E}_{\tau \sim p(\tau)}\left[\sum_{t \geq 0} R\left(x_t, u_t\right)\right]}_{J(\theta)}$$

Evaluating the objective

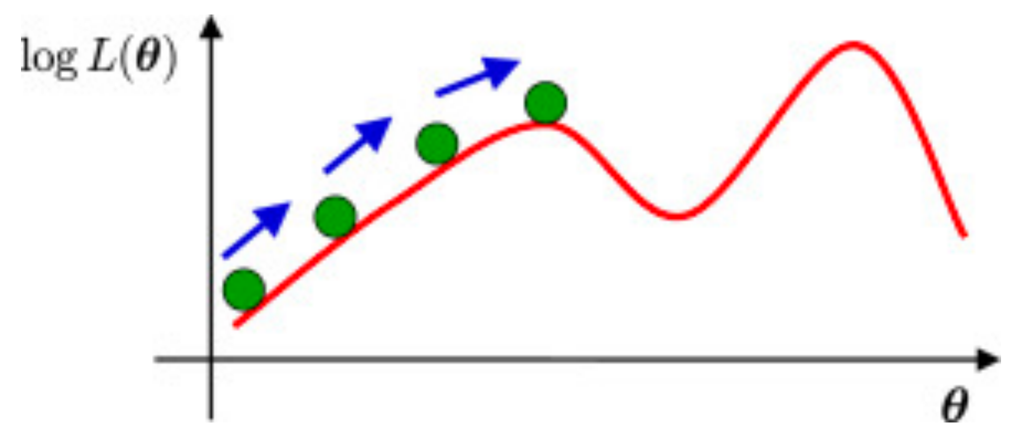
- Opposed to value-based methods, policy optimization attempts to learn the policy directly (i.e., optimize $J(\theta)$ w.r.t. θ)

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t \geq 0} R\left(x_t, u_t\right) \right]$$

One of the most direct ways to optimize this objective is to:

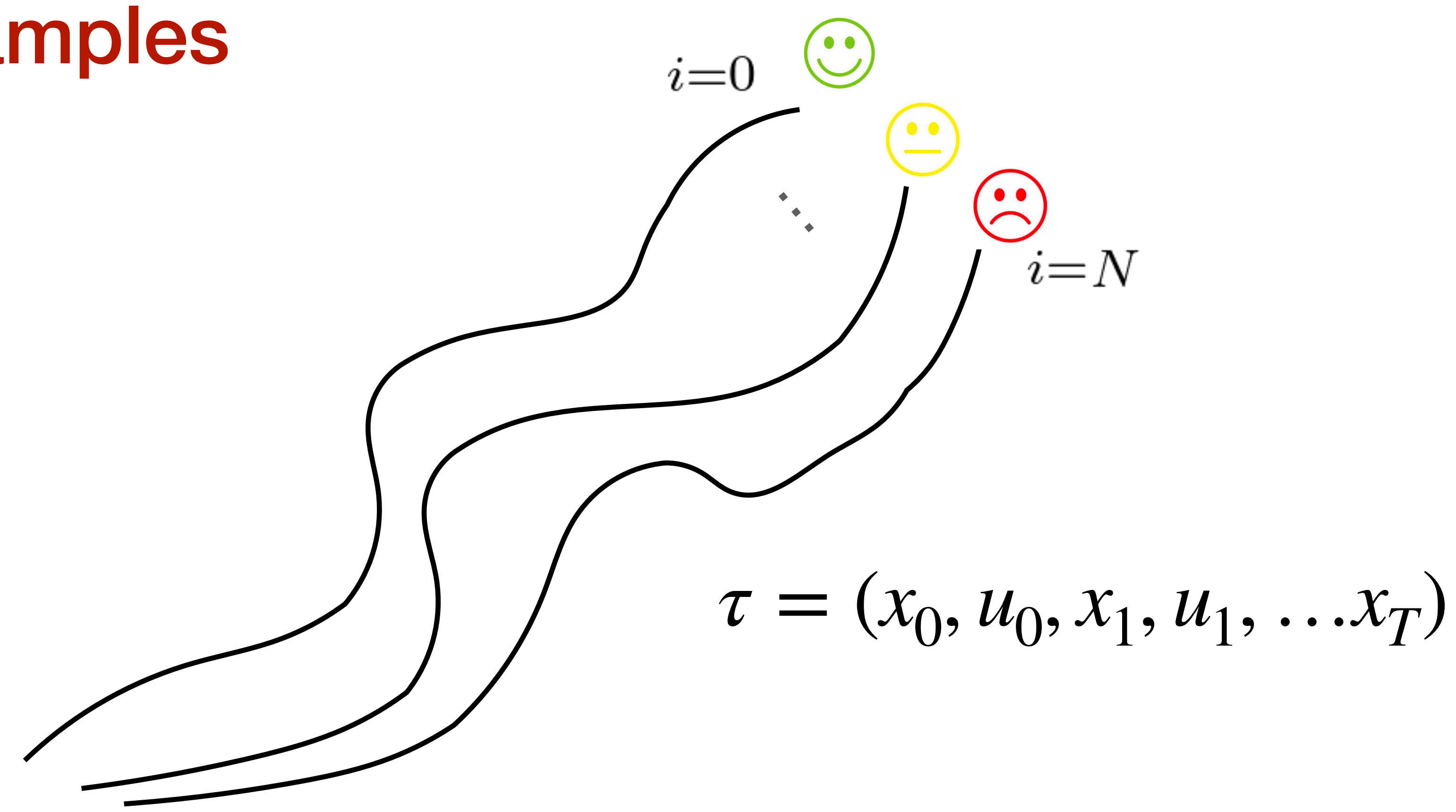
(1) estimate its gradient $\nabla_\theta J(\theta)$

(2) cast the learning process as approximate gradient ascent on $J(\theta)$ $\qquad \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



How can we evaluate the expectation in the objective? As usual in RL, **through samples**

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t \geq 0} R\left(x_t, u_t\right) \right] \approx \frac{1}{N} \sum_i \sum_t R(x_{i,t}, u_{i,t})$$



Direct policy gradient

- In order to solve the problem through gradient-based optimization we need to compute $\nabla_\theta J(\theta)$
- Let us define the compact notation $r(\tau) = \sum_{t=1}^{T} R(x_t, u_t)$
- By definition of expectation $J(\theta) = \mathbb{E}_{\tau \sim p(\tau)}\left[r(\tau)\right] = \int p_\theta(\tau) r(\tau) d\tau$
- We can then write the gradient $\nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau$

Problem: gradient depends on unknown dynamics and initial state distribution through $p_\theta(\tau)$

Useful identity:

$$p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) = p_\theta(\tau)\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = \nabla_\theta p_\theta(\tau)$$

$$\nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau = \int p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) r(\tau) d\tau = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\nabla_\theta \log p_\theta(\tau) r(\tau)\right]$$

On the right track since we can evaluate expectations through samples… but we still have $\nabla_\theta \log p_\theta(\tau)$ in the equation

Direct policy gradient

Let us recall the trajectory distribution

$$p(x_0, u_0, \ldots, x_T) = p(\tau) = p(x_0)\prod_{t=1}^{T}\pi(u_t \mid x_t)p(x_{t+1} \mid x_t, u_t)$$

$$\log p(\tau) = \log p(x_0) + \sum_{t=1}^{T}\log \pi_\theta(u_t \mid x_t) + \log p(x_{t+1} \mid x_t, u_t)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\nabla_\theta \log p_\theta(\tau) r(\tau)\right] = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\nabla_\theta\left[\cancel{\log p(x_0)} + \sum_{t=1}^{T}\log \pi_\theta(u_t \mid x_t) + \cancel{\log p(x_{t+1} \mid x_t, u_t)}\right] r(\tau)\right]$$

- When taking the gradient w.r.t. θ, $\log p(x_0), \log p(x_{t+1} \mid x_t, u_t)$ do not depend on θ
- While we can evaluate the log probability under our parametric policy π_θ
- This enable us to re-write the gradient $\nabla_\theta J(\theta)$ as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_t \mid x_t\right)\right)\left(\sum_{t=1}^{T}R\left(x_t, u_t\right)\right)\right]$$

Everything inside this expectation is known

Direct policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_t \mid x_t\right)\right)\left(\sum_{t=1}^{T} R\left(x_t, u_t\right)\right)\right]$$

Everything inside this expectation is known

- Recall how we use samples to evaluate the objective: $J(\theta) = \mathbb{E}_{\tau\sim p(\tau)}\left[\sum_{t\geq 0} R\left(x_t, u_t\right)\right] \approx \frac{1}{N}\sum_i \sum_t R(x_{i,t}, u_{i,t})$

- We can use the same idea to evaluate the gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_t \mid x_t\right)\right)\left(\sum_{t=1}^{T} R\left(x_t, u_t\right)\right)\right] \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right)\right)\left(\sum_{t=1}^{T} R\left(x_{i,t}, u_{i,t}\right)\right)\right]$$

REINFORCE algorithm

The procedure described so far gives us the basic policy gradient algorithms, a.k.a. REINFORCE:

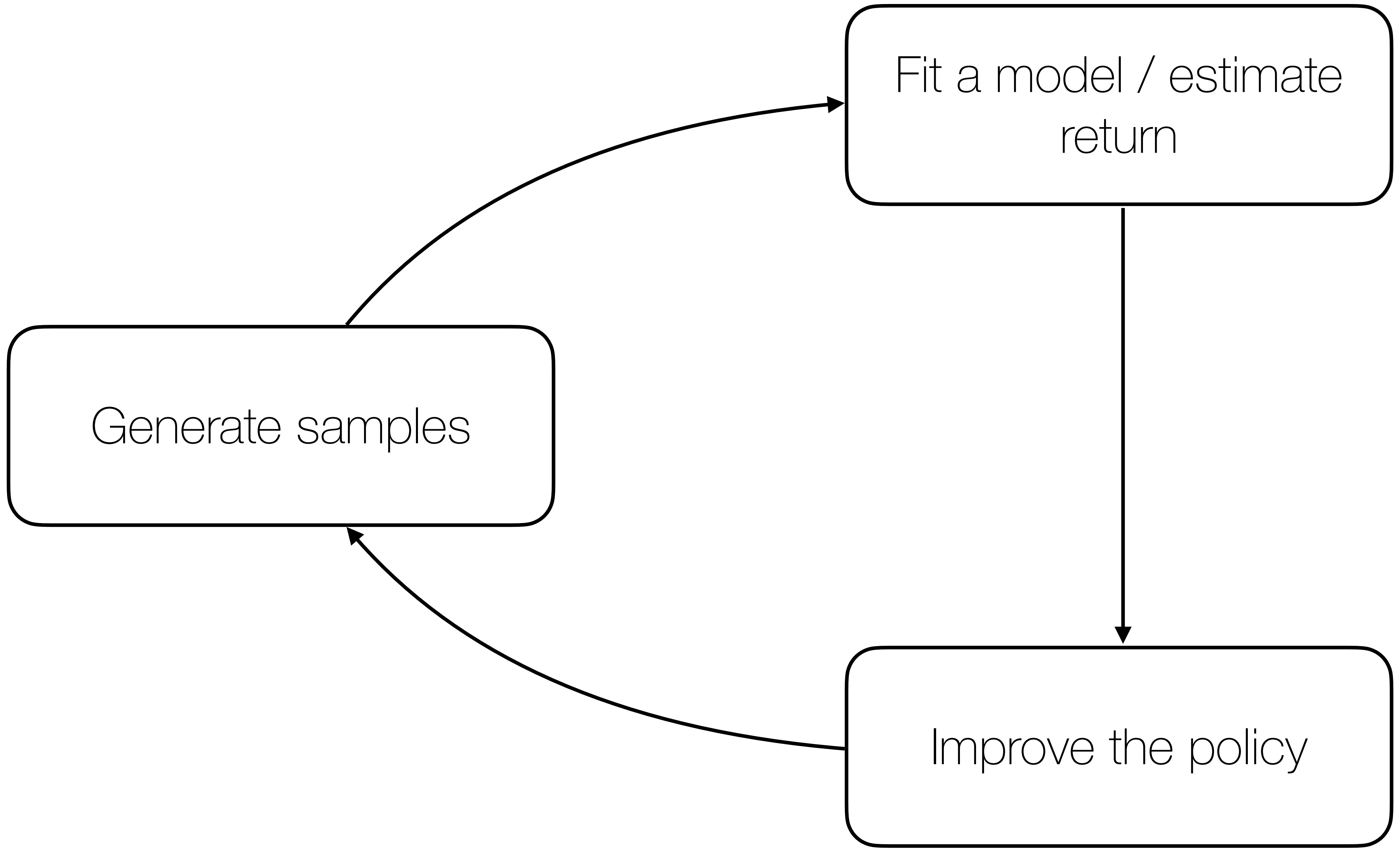
1. Sample trajectories $\{\tau_i\}_{i=1}^{N}$ from $\pi_\theta(u_t \mid x_t)$, i.e. run the policy in the environment
2. Evaluate the policy gradient $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right)\right)\left(\sum_{t=1}^{T}R\left(x_{i,t}, u_{i,t}\right)\right)\right]$
3. Take a gradient step to update the policy $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Estimate the return (via MC)

$$\sum_{t=1}^{T} R\left(x_{i,t}, u_{i,t}\right)$$

Run the policy and observe

$$\{\tau_i\}_{i=1}^{N}$$

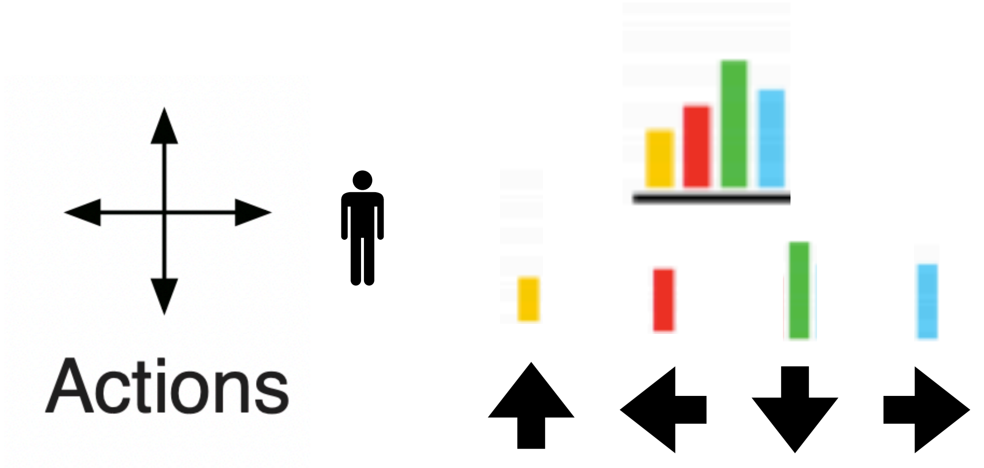


$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

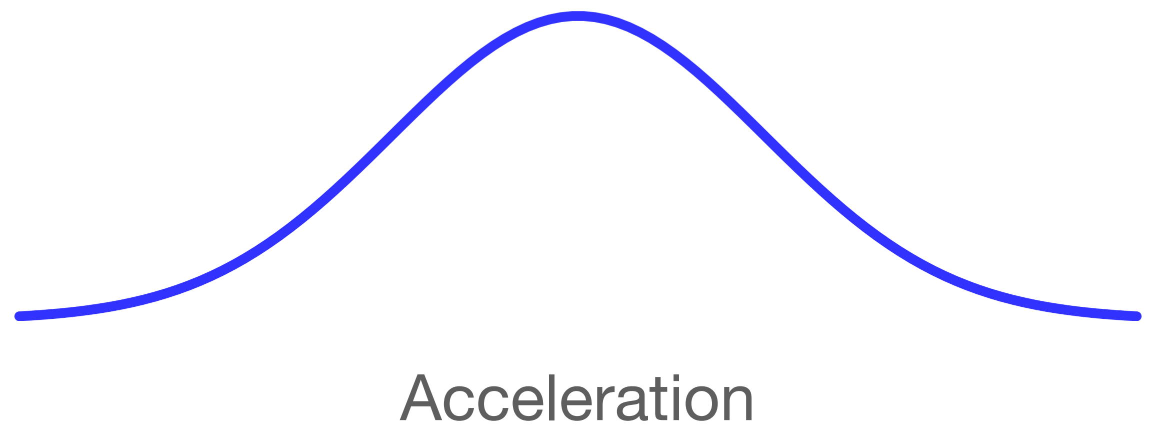
Intuition: “what is PG doing?”

Consider the expression we derived for the policy gradient $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta \boxed{\log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right)}\right)\left(\sum_{t=1}^{T} R\left(x_{i,t}, u_{i,t}\right)\right)\right]$

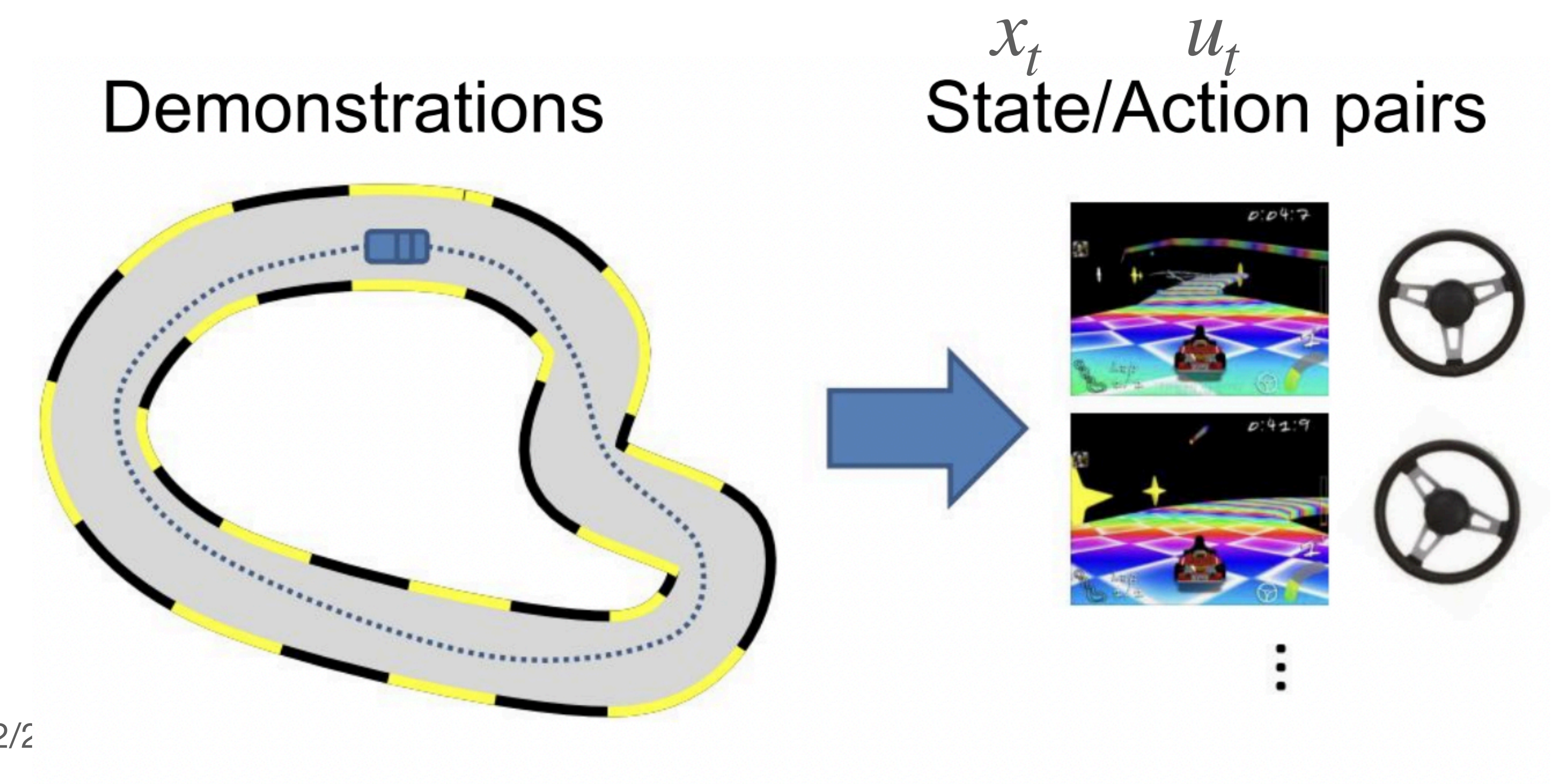
Discrete action space



Continuous action space



Let’s compare it with the expression of the gradient when performing maximum likelihood (e.g., supervised learning):



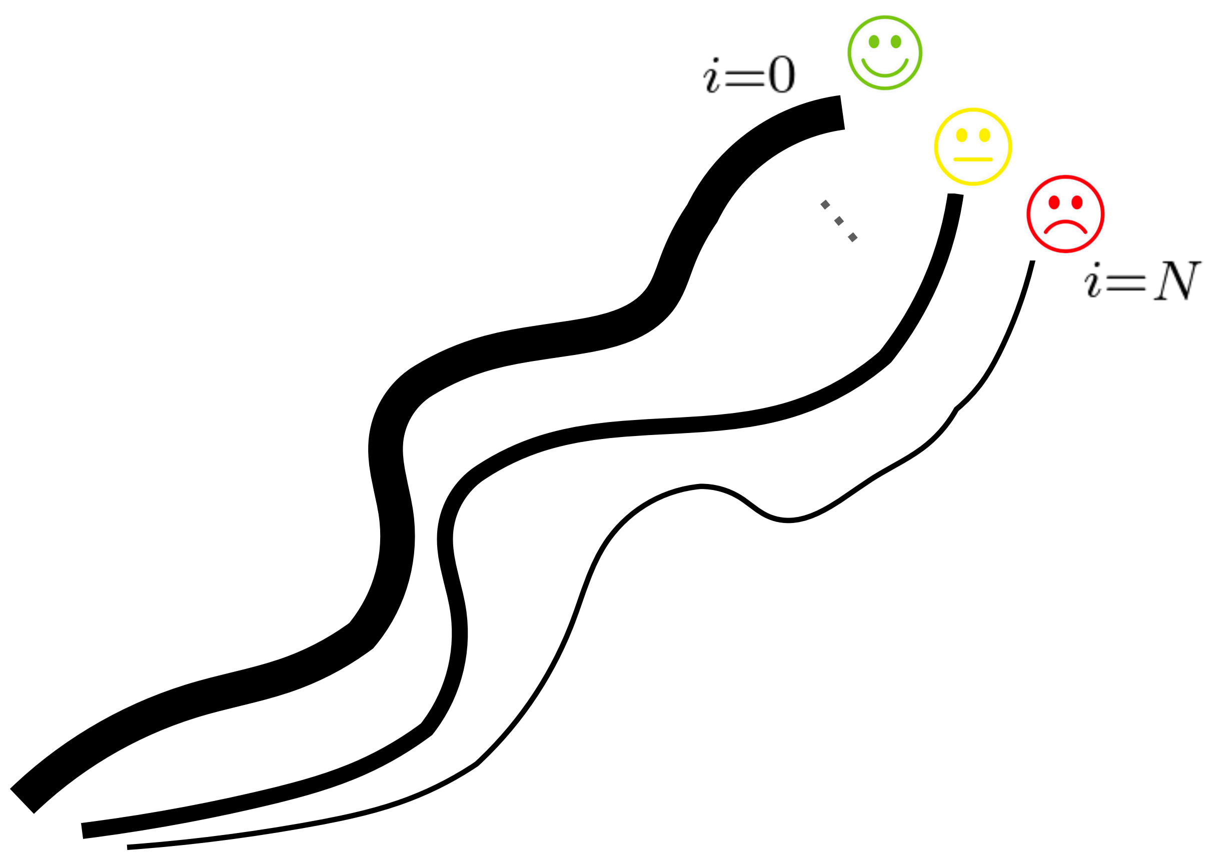
$$\nabla_\theta J_{MLE}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right)\right)\right]$$

The policy gradient is a weighted version of the MLE gradient

Intuition: "what is PG doing?"

Policy gradient: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right)\right)\left(\sum_{t=1}^{T} R\left(x_{i,t}, u_{i,t}\right)\right)\right]$

Maximum Likelihood: $\nabla_\theta J_{MLE}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right)\right)\right]$



Taking a step in the direction the policy gradient essentially means:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$$

"Change parameters $\boldsymbol{\theta}$ s.t. trajectories with higher reward have higher probability"

PG formalizes the idea of learning by "trial and error"

Outline

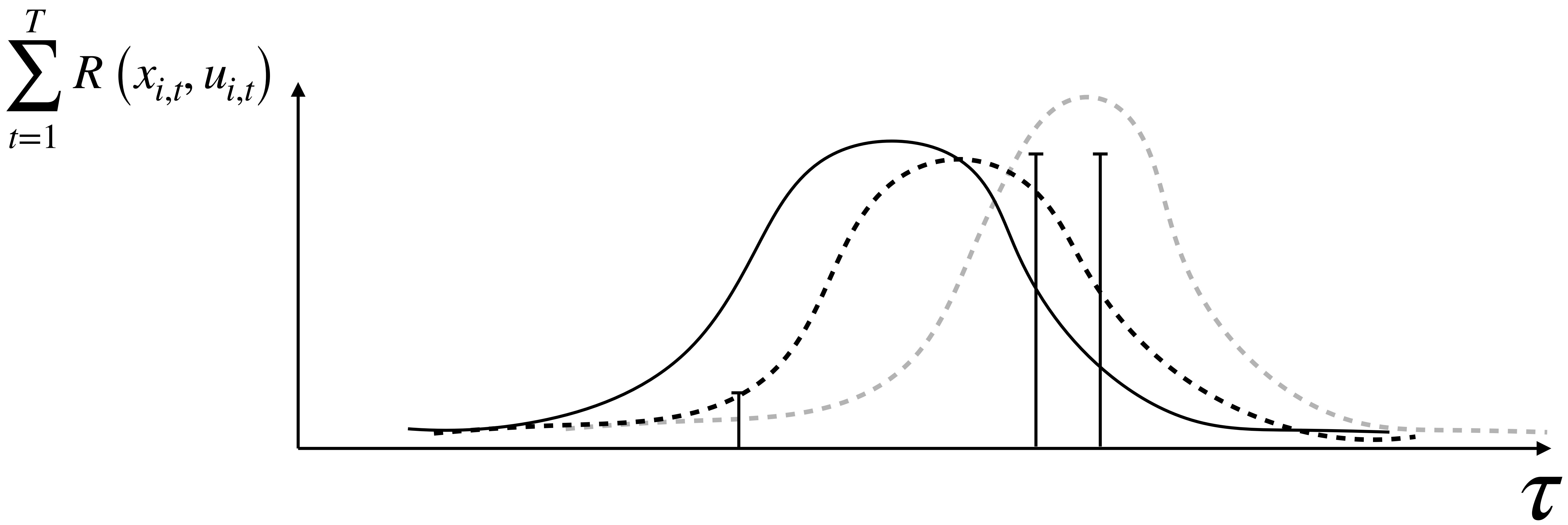
Intro to policy gradients

- REINFORCE algorithm
- Reducing variance of policy gradient

Problem: high variance of the PG

Policy gradient: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta \left(u_{i,t} \mid x_{i,t} \right) \right) \left(\sum_{t=1}^{T} R \left(x_{i,t}, u_{i,t} \right) \right) \right]$

Let's consider the following example:



- Depending on the sample, the policy gradient can vary wildly: PG estimator has **high variance**
- This negatively affects learning: worse performance, slower convergence

A lot of research in the domain of Policy Optimization revolves around finding ways to lower the variance of the policy gradient

Reducing the variance

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right)\right)\left(\sum_{t=1}^{T} R\left(x_{i,t}, u_{i,t}\right)\right)\right]$$

A first simple approach to reduce the variance entails using *causality*: "policy at time t' cannot affect reward at time $t < t'$"

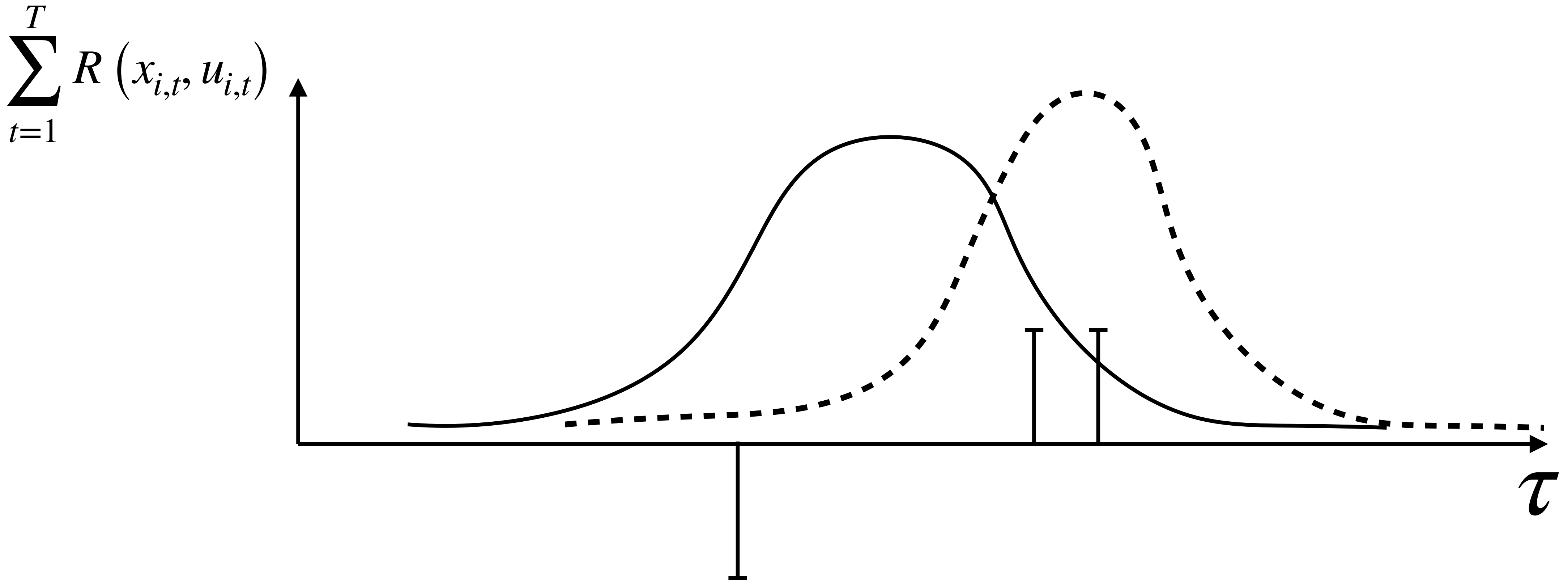
Consider this equivalent expression:

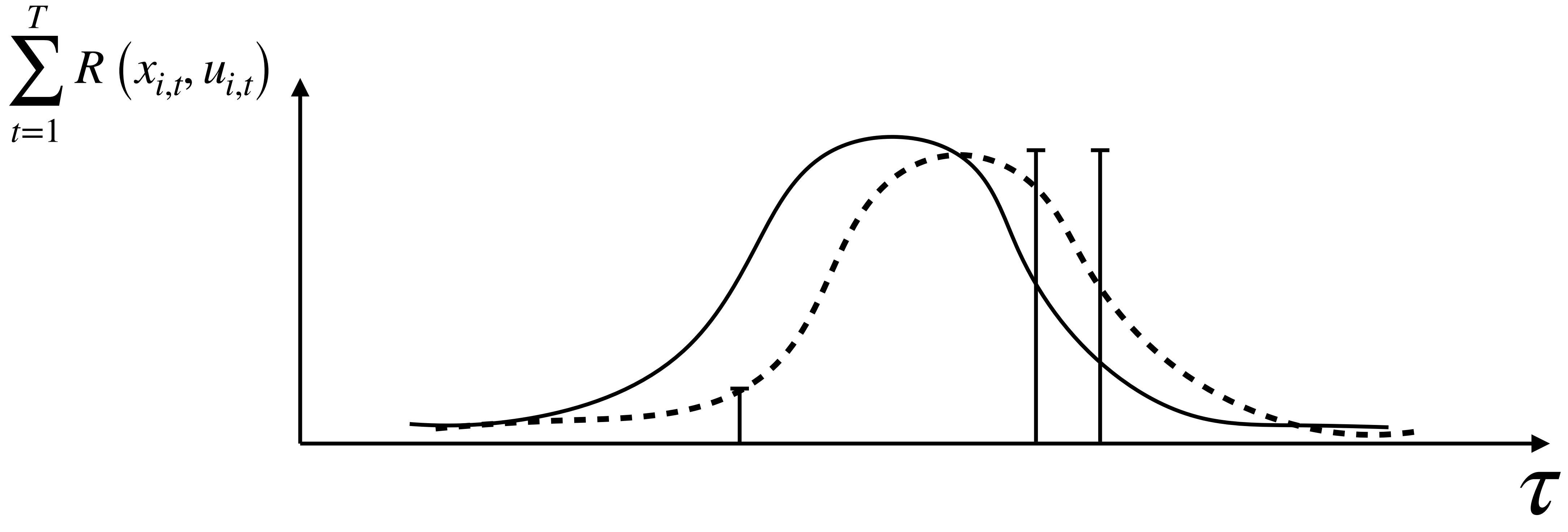
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right)\left(\sum_{t'=t}^{T} R\left(x_{i,t'}, u_{i,t'}\right)\right)$$

Baseline

A second (and extremely important) approach to reduce variance of PG estimators relates with the concept of **baseline**

Let's reconsider our intuition on PG, i.e., "making good behavior more likely"





However, PG will only do this **if the returns are centered** (e.g., consider the counter-example on the right)

Intuitively, we want to "center" our returns, such that:
- The probability of behavior that is better than average gets increased
- The probability of behavior that is worse than average gets decreased

We are going to subtract a baseline b from the expression of the PG $\quad \nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau)\left[r(\tau) - b\right]$

A closer look at the baseline

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\nabla_\theta \log p_\theta(\tau) r(\tau)\right] \rightarrow \nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\nabla_\theta \log p_\theta(\tau)\big(r(\tau) - b\big)\right]$$

Claim: adding the baseline does not change the value of the expected gradient

- To prove that, let's consider the following expectation:

Useful identity:

$$p_\theta(\tau)\,\nabla_\theta \log p_\theta(\tau) = p_\theta(\tau)\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = \nabla_\theta p_\theta(\tau)$$

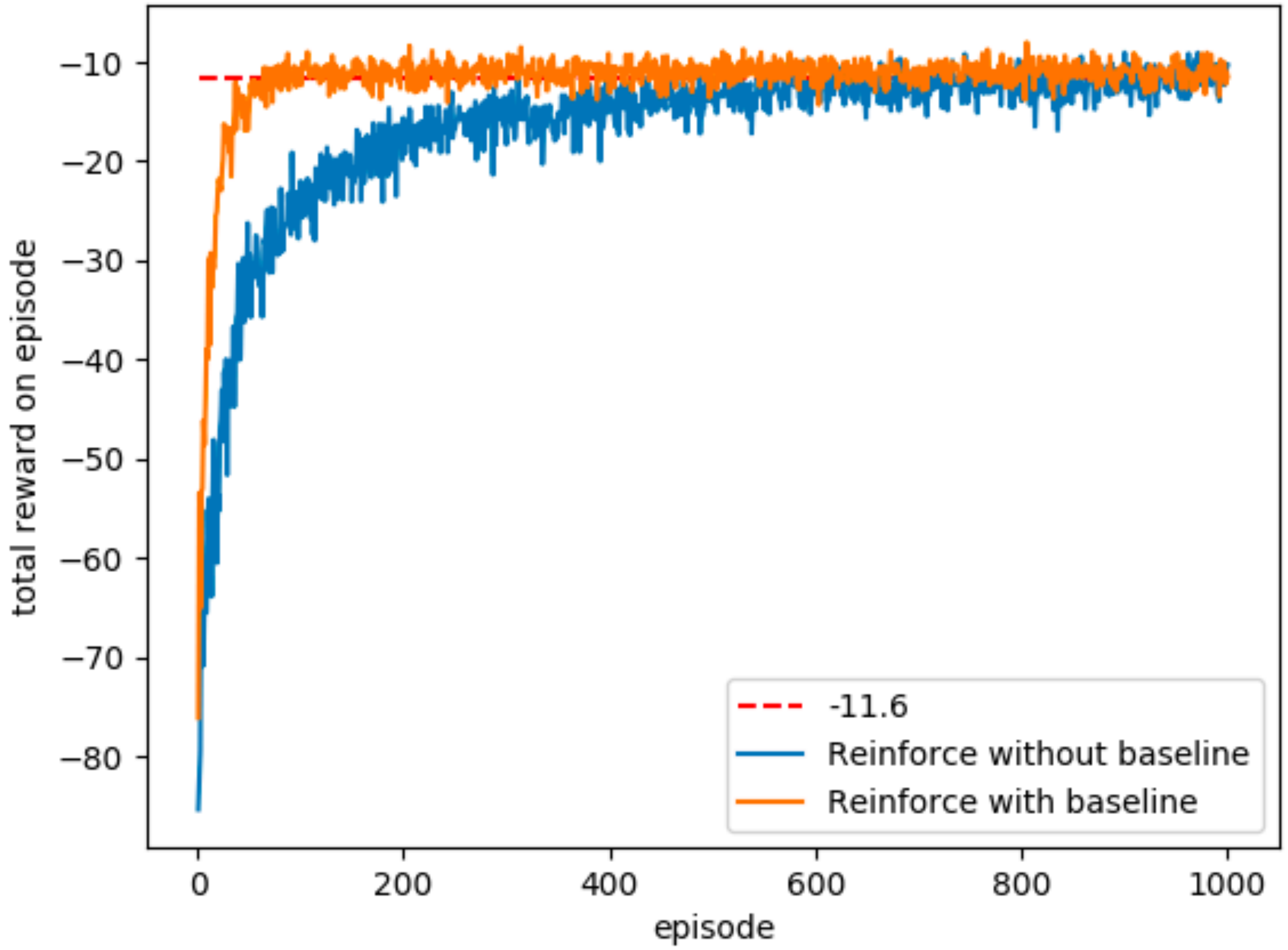
$$\mathbb{E}\left[\nabla_\theta \log p_\theta(\tau)\, b\right] = \int p_\theta(\tau)\,\nabla_\theta \log p_\theta(\tau) b\, d\tau = \int \nabla_\theta p_\theta(\tau) b\, d\tau = b\,\nabla_\theta \int p_\theta(\tau)\, d\tau = b\,\nabla_\theta 1 = 0$$

which makes our estimate of the gradient (with baseline) *unbiased* in expectation

- An extremely effective choice of the baseline is the **average return,** $b = \frac{1}{N}\sum_{i=1}^{N} r(\tau_i)$

(We'll see how this motivates many popular RL algorithms…)

Example



−10
−20
−30
−40
−50
−60
−70
−80
total reward on episode
0
200
400
600
800
1000
episode
-11.6
Reinforce without baseline
Reinforce with baseline

Properties of policy gradient

At a high-level, we've been defining a scheme where:

- Given the RL objective $J(\theta) = \mathbb{E}_{\tau \sim p(\tau)}\left[r(\tau)\right] = \int p_\theta(\tau) r(\tau) d\tau$
- We maximize the objective w.r.t. θ by:
  - Computing the gradient $\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\nabla_\theta \log p_\theta(\tau) r(\tau)\right]$
  - Taking a gradient step to update the policy $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Question:
Is this on- or off-policy? And why?

Outline

Actor-Critic methods

- Advantage
- Architecture design

From PG to Actor-Critic methods

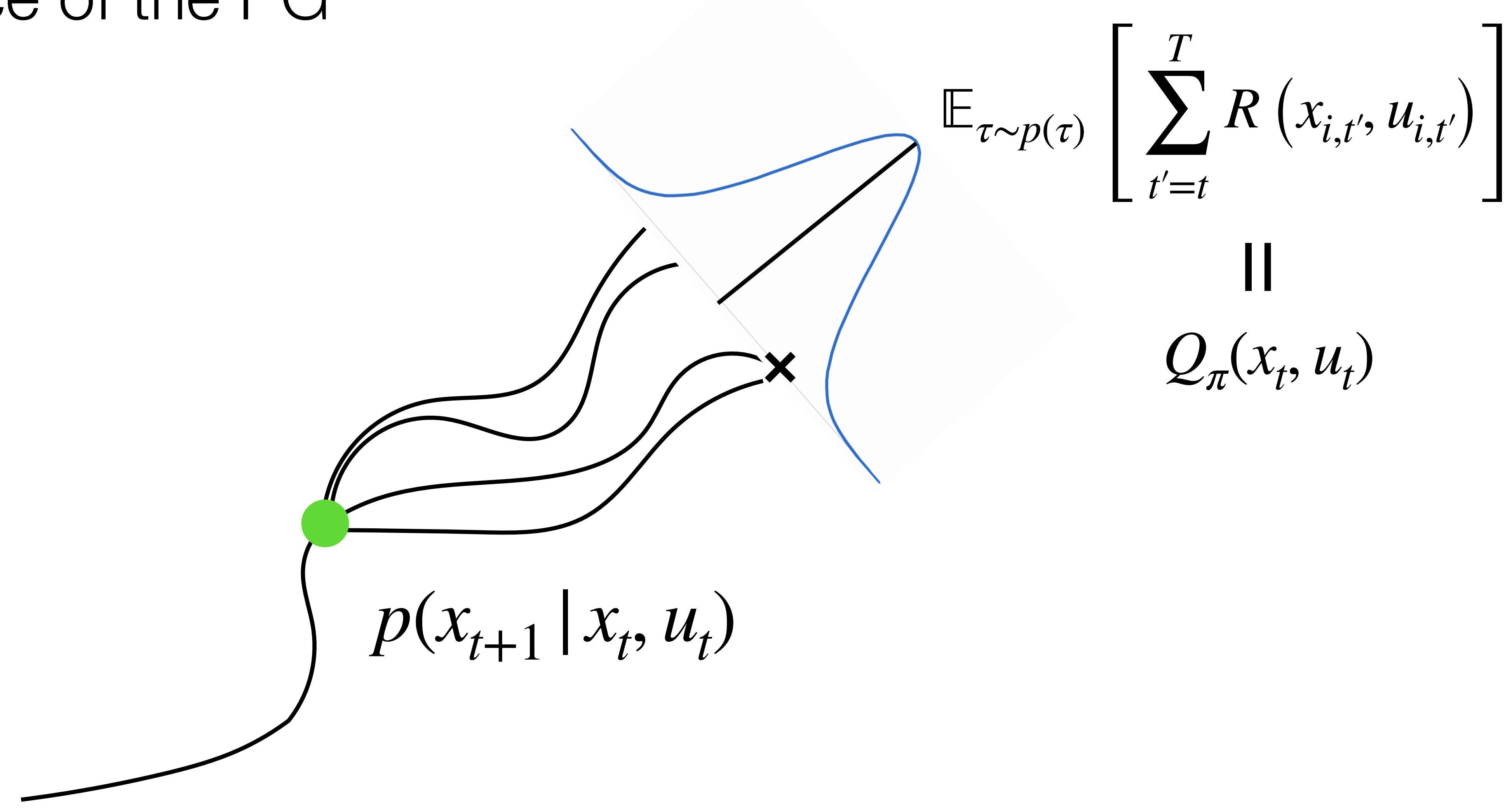
Once again, let's consider the policy gradient $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right)\left(\sum_{t'=t}^{T} R\left(x_{i,t'}, u_{i,t'}\right)\right)$

"reward-to-go"

This one-sample estimate of the reward-to-go contributes to the high variance of the PG

The idea of actor-critic methods is to define:

- An "actor", i.e., a policy $\pi_\theta(u_t \mid x_t)$
- A "critic" to better estimate the "reward-to-go",
  e.g., estimate Q-values through function approximation $Q_\phi(x_t, u_t)$



By using this better estimate of the reward-to-go we can get a lower variance policy gradient:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right) Q_\phi(x_t, u_t)$$

What about the baseline?

Can we use a baseline when using the approximate reward-to-go and reduce the variance even further?

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta \left(u_{i,t} \mid x_{i,t} \right) \left(Q_\phi(x_{i,t}, u_{i,t}) - b \right)$$

- An effective choice for b is a state-dependent baseline $b(x_t) = \mathbb{E}_{u_t \sim \pi(u_t|x_t)} \left[Q(x_t, u_t) \right] = V(x_t)$
- We can thus rewrite:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta \left(u_{i,t} \mid x_{i,t} \right) \left(Q_\phi(x_{i,t}, u_{i,t}) - V(x_{i,t}) \right)$$

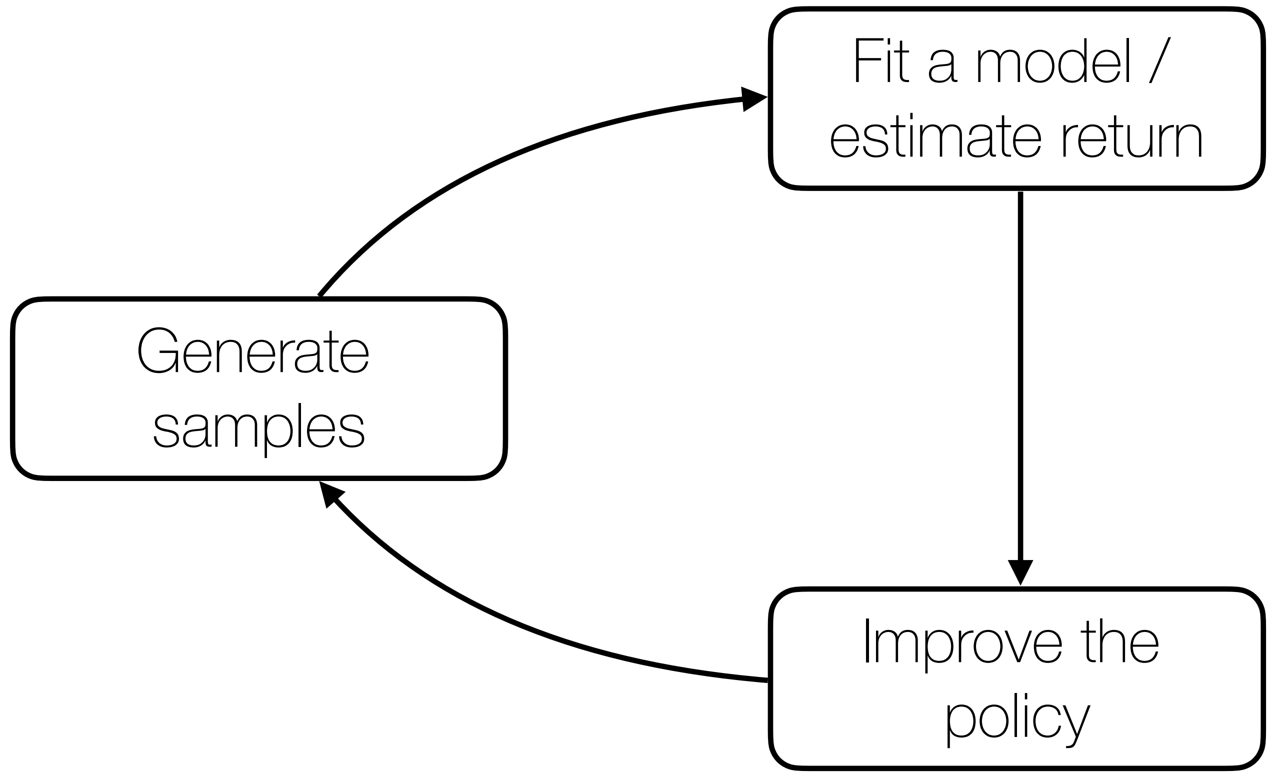
"How much u_t is better than the average action in x_t" $\longrightarrow$

Following this gradient:
- increases the probability of actions that have returns better than average
- decreases the probability of actions that have returns worse than average

- The function $A(x_t, u_t) = Q_\phi(x_t, u_t) - V(x_t)$ is usually referred to as **advantage function**

Fitted Q-learning:

Run policy and observe trajectories



Set target

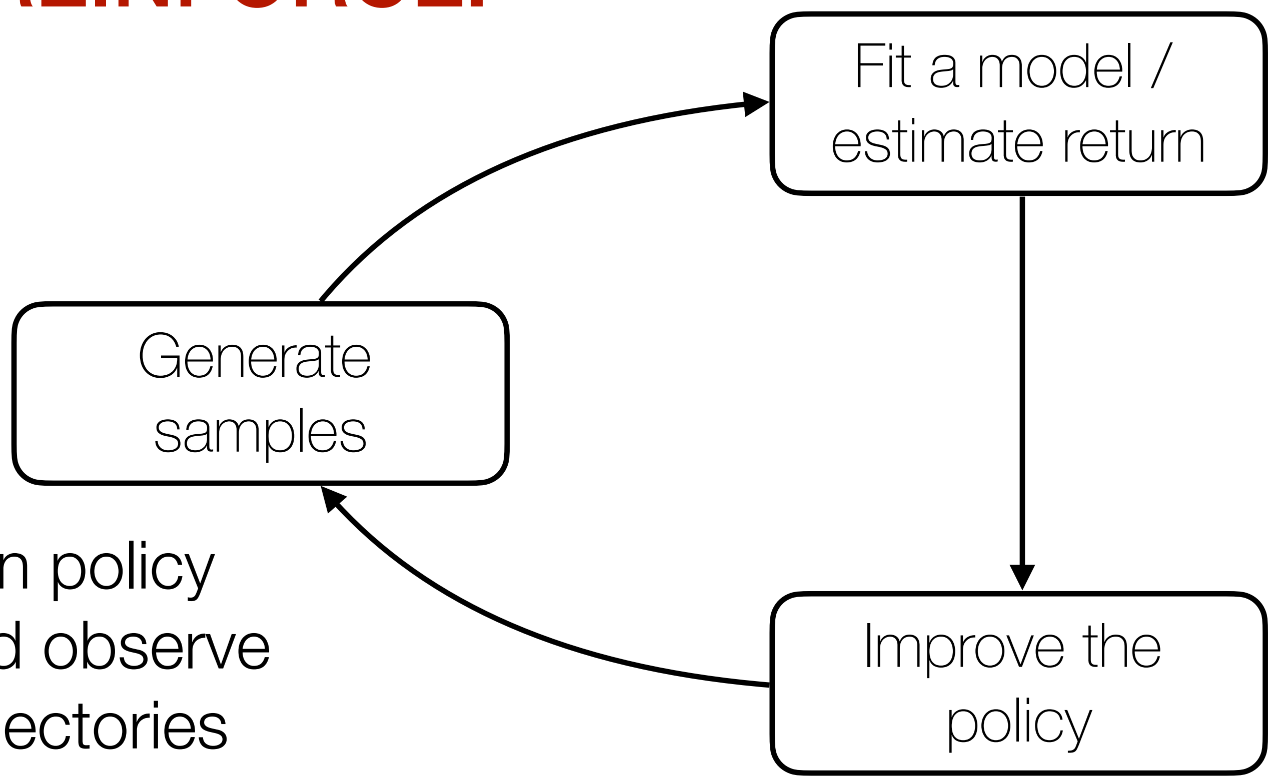
$$y_t \leftarrow r_t + \gamma \max_u Q_\phi(x_{t+1}, u)$$

Update ϕ to minimize

$$\min_\phi \mathbb{E}_\pi \left[y_t - Q_\phi(x_t, u_t) \right]$$

set $\pi\left(x_t\right) = \arg\max_u Q_\theta\left(x_t, u_t\right)$

REINFORCE:



Estimate the return (via MC)

$$\sum_{t=1}^{T} R\left(x_{i,t}, u_{i,t}\right)$$

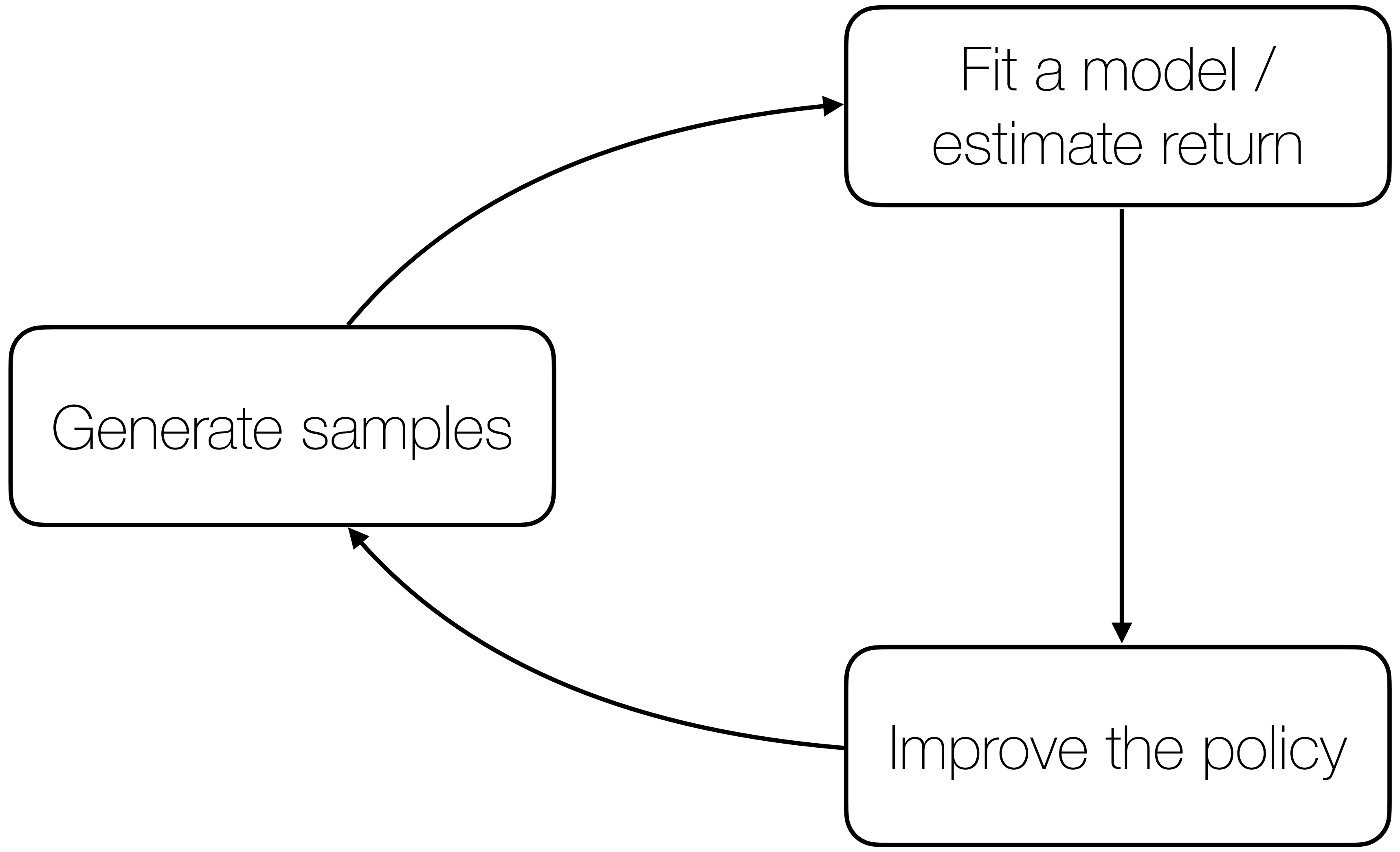
Run policy and observe trajectories

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right) \right) \left(\sum_{t=1}^{T} R\left(x_{i,t}, u_{i,t}\right) \right) \right]$$

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

Actor-Critic:

Run policy and observe trajectories

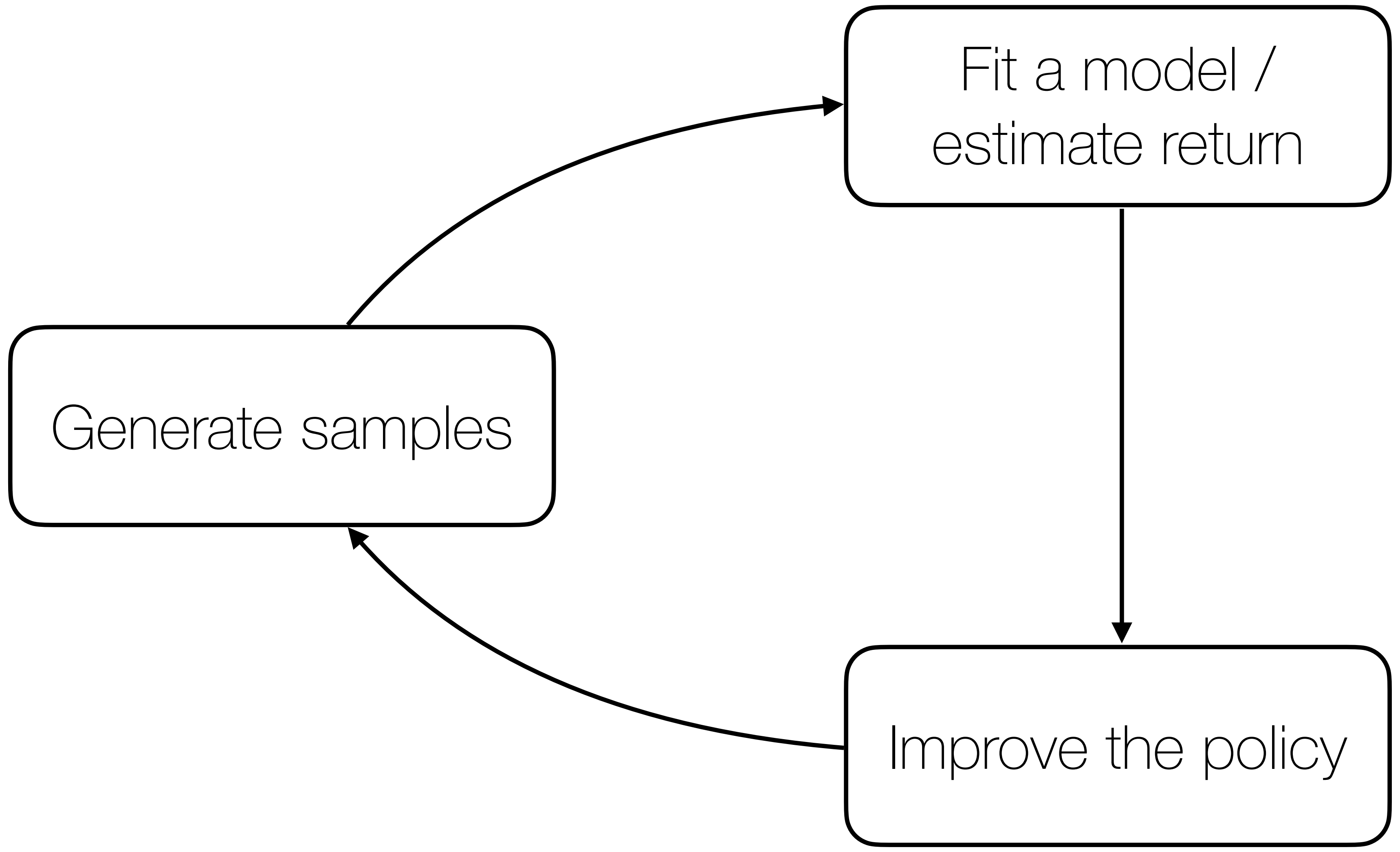


Fit V_π, Q_π or A_π

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

Actor-Critic:

Run policy and
observe trajectories



Fit V_π, Q_π or A_π

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

What quantity should we estimate? What are the trade-offs between estimating $V_\pi(x_t)$, $Q_\pi(x_t, u_t)$ or A_π? No wrong/right, answer, it depends. For now, let's consider the complexity of the estimation problem (i.e., fitting V_π is easier: only x_t as input)

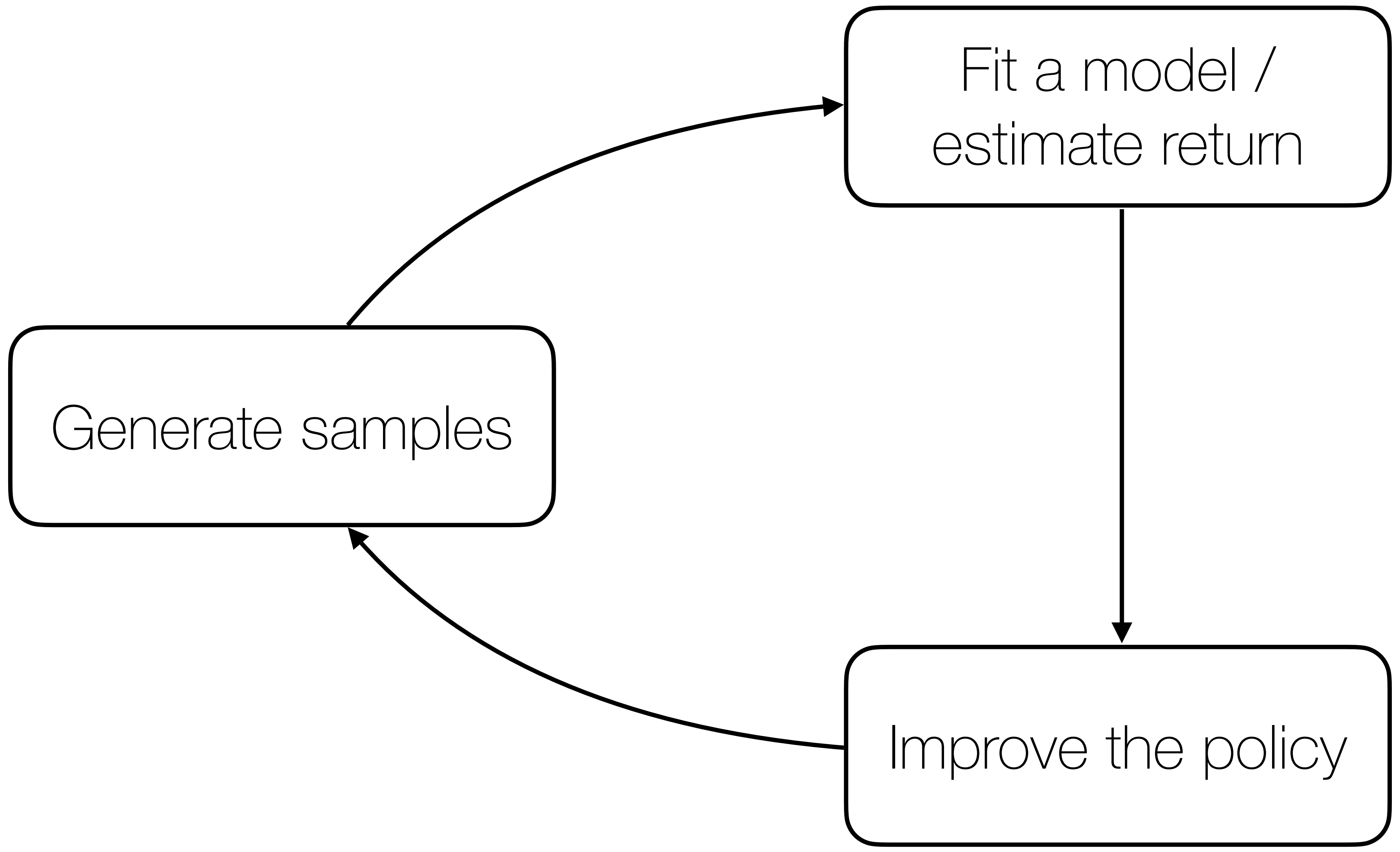
$$Q_\pi(x_t, u_t) = \mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t'=t}^{T} R\left(x_{i,t'}, u_{i,t'}\right) \right] = r\left(x_t, u_t\right) + \mathbb{E}_{x_{t+1} \sim p\left(x_{t+1} \mid x_t, u_t\right)} \left[V_\pi\left(x_{t+1}\right) \right] \approx r\left(x_t, u_t\right) + V_\pi\left(x_{t+1}\right)$$

$$A_\pi(x_t, u_t) = Q_\pi(x_t, u_t) - V_\pi(x_t) \approx r\left(x_t, u_t\right) + V_\pi\left(x_{t+1}\right) - V_\pi\left(x_t\right)$$

This enables us to "only" fit V_π

Advantage Actor-Critic (A2C):

Run policy and observe trajectories



Fit V_π

Compute

$$A_\pi(x_t, u_t) = r\left(x_t, u_t\right) + V_\pi\left(x_{t+1}\right) - V_\pi\left(x_t\right)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right) A_\phi(x_{i,t}, u_{i,t})$$

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

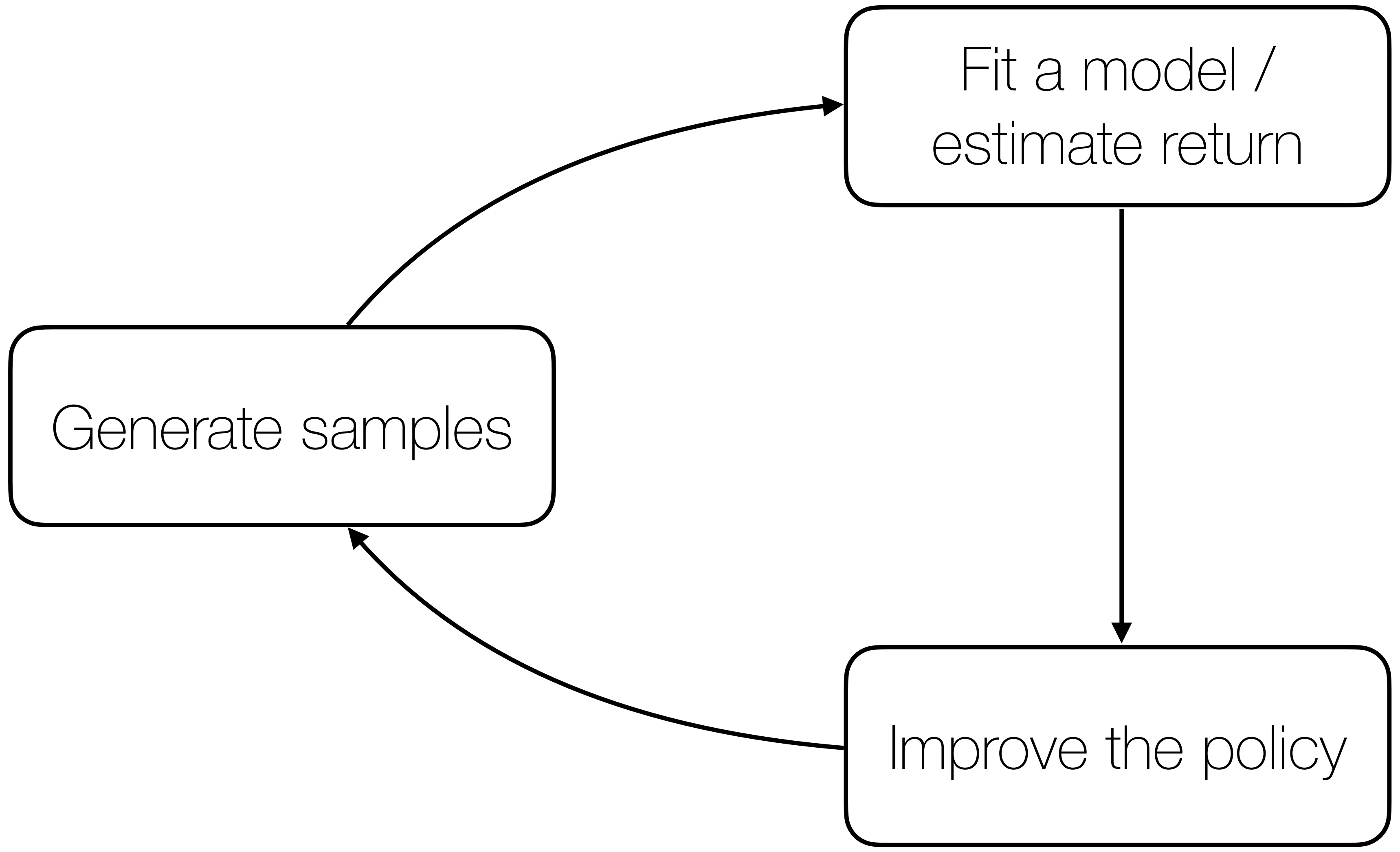
What quantity should we estimate? What are the trade-offs between estimating $V_\pi(x_t)$, $Q_\pi(x_t, u_t)$ or A_π? No wrong/right, answer, it depends. For now, let's consider the complexity of the estimation problem (i.e., fitting V_π is easier: only x_t as input)

$$Q_\pi(x_t, u_t) = \mathbb{E}_{\tau \sim p(\tau)}\left[\sum_{t'=t}^{T} R\left(x_{i,t'}, u_{i,t'}\right)\right] = r\left(x_t, u_t\right) + \mathbb{E}_{x_{t+1} \sim p\left(x_{t+1} \mid x_t, u_t\right)}\left[V_\pi\left(x_{t+1}\right)\right] \approx r\left(x_t, u_t\right) + V_\pi\left(x_{t+1}\right)$$

$$A_\pi(x_t, u_t) = Q_\pi(x_t, u_t) - V_\pi(x_t) \approx r\left(x_t, u_t\right) + V_\pi\left(x_{t+1}\right) - V_\pi\left(x_t\right)$$

This enables us to "only" fit V_π

Advantage Actor-Critic (A2C):



Run policy and observe trajectories

Fit V_π

Compute

$$A_\pi(x_t, u_t) = r\left(x_t, u_t\right) + V^\pi\left(x_{t+1}\right) - V^\pi\left(x_t\right)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right) A_\phi(x_{i,t}, u_{i,t})$$

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

When fitting V_π, we can use different *targets* to define the supervised learning labels

Question:

How to fit with MC target?

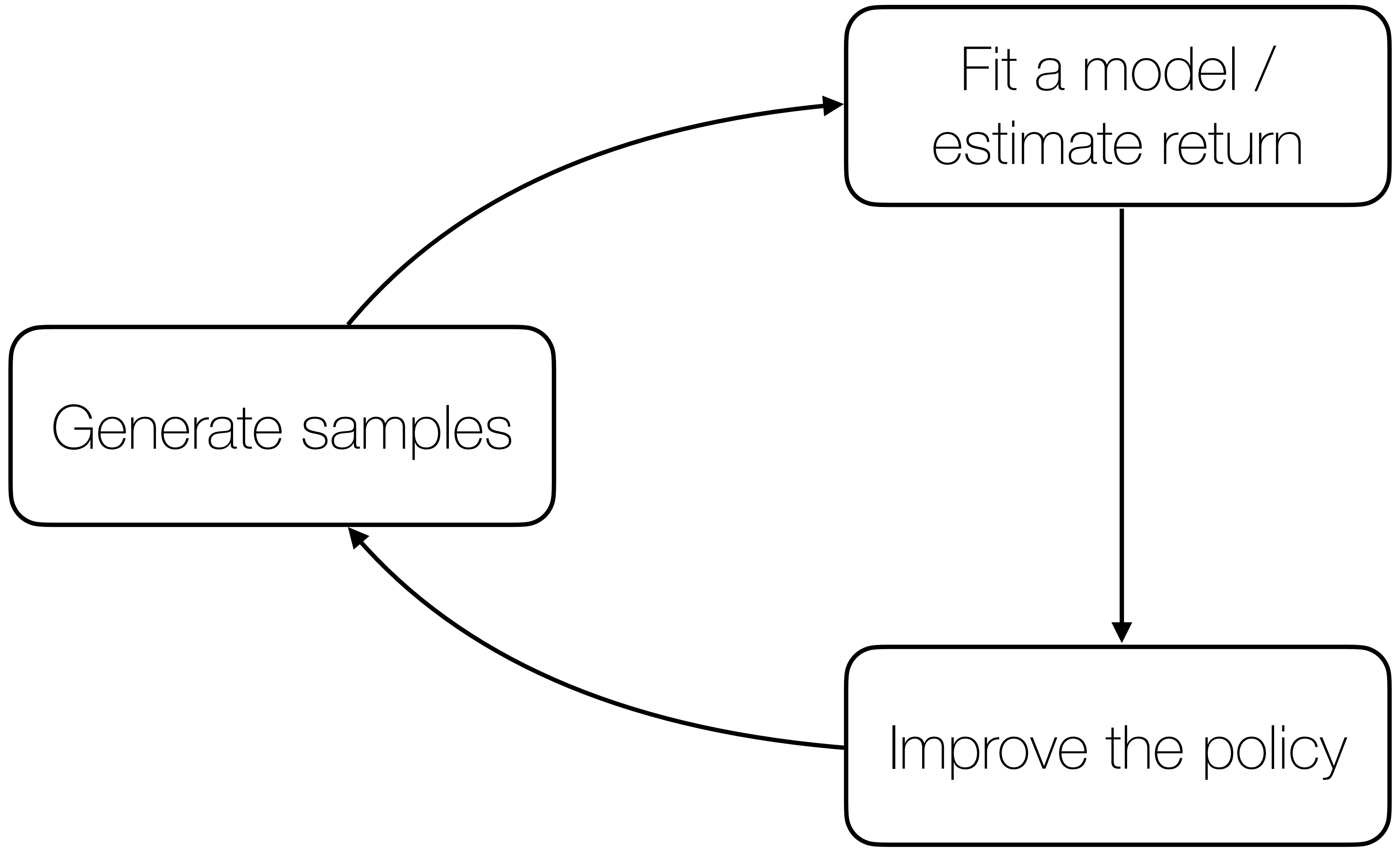
1. Collect dataset $\mathscr{D} = \{(x_t, G_t)\}, G_t = \sum_{t'=t}^{T} R(x_t, u_t)$
2. Supervised regression on $\mathscr{D}$

How to fit with TD target?

1. Collect dataset $\mathscr{D} = \{(x_t, r_t + \gamma \hat{V}_\theta(x_t))\}$
2. Supervised regression on $\mathscr{D}$

Architecture design

Run policy and observe trajectories



Fit V_π

Compute

$$A_\pi(x_t, u_t) = r\left(x_t, u_t\right) + V_\pi\left(x_{t+1}\right) - V_\pi\left(x_t\right)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right) A_\phi(x_{i,t}, u_{i,t})$$

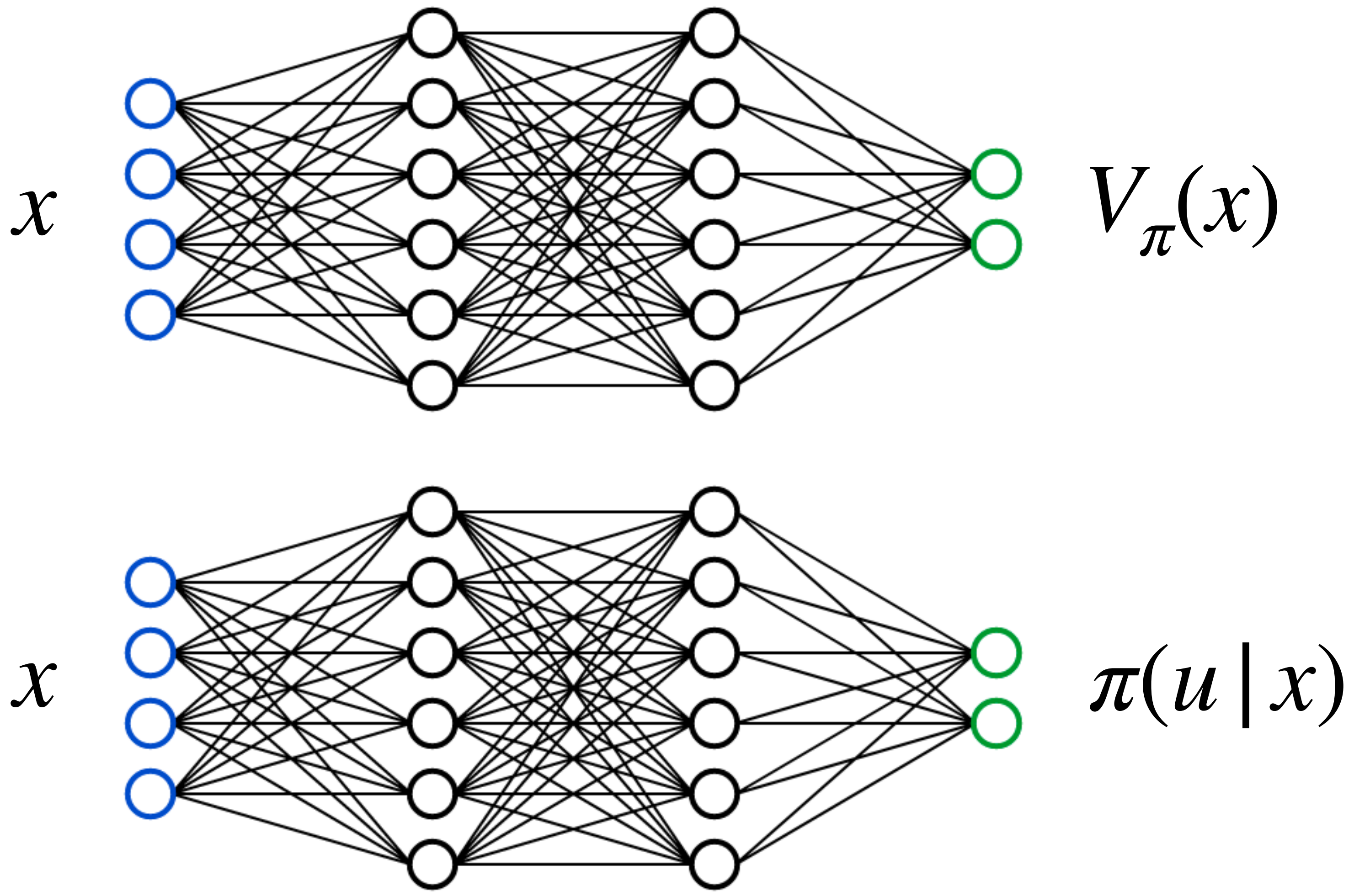
$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

In practice, one could opt for different designs of this same algorithms, e.g.,:

- Two network vs shared network
- Parallel processing: synchronized vs asynchronous

Simple, typically more stable

Does not share features

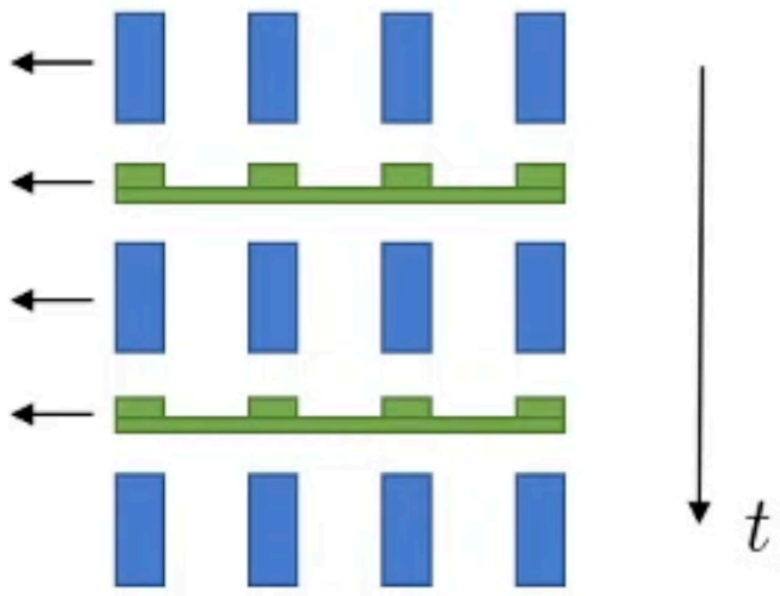


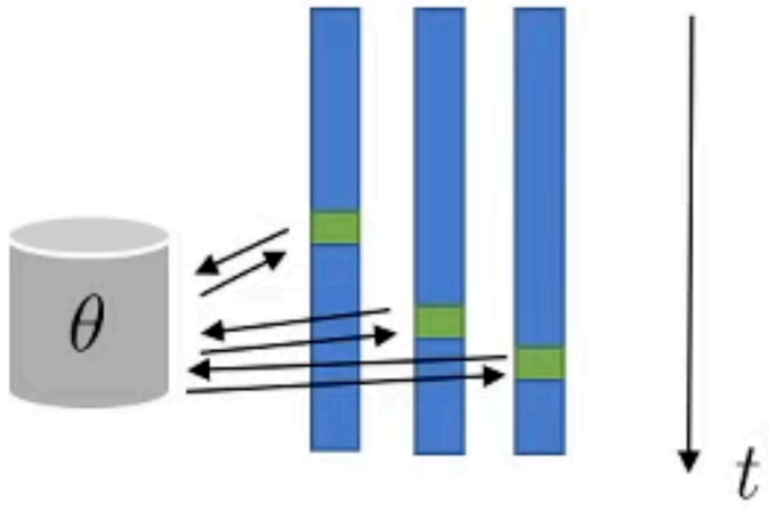
Get (x_t, u_t, x_{t+1}, r_t)

Update θ

Get (x_t, u_t, x_{t+1}, r_t)

Update θ





Outline

Deep RL Algorithms & Applications

Reducing RL to optimization

- Much of modern ML entails reducing learning to a numerical optimization problem
  - Supervised learning as *training error minimization*

- This is different from what we have seen so far in RL:
  - Q-learning: fixed-point iteration → can (in principle) include all transitions seen so far, however, it optimizes for the wrong objective
  - Policy gradient: yes, stochastic gradients of the RL objective, but no optimization problem

- We'll discuss approaches that define an optimization problem that allows us to do a small update to policy π, based on data sampled from π

Defining the objective

- We discussed how, in PO, we want to compute the following gradient $\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(u_t \mid x_t)\, A(x_t, u_t)\right]$
- To implement this using modern auto-diff tools (e.g., Torch, Jax, Tensorflow), this means writing the following loss function:

$$L^{PG}(\theta) = \mathbb{E}\left[\log \pi_\theta(u_t \mid x_t)\, A(x_t, u_t)\right]$$

- But we don't want to optimize it too far, since we are not working with the *true* advantage, rather with a noisy estimate
- Equivalently differentiate

$$L^{IS}(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\frac{\pi_\theta(u_t \mid x_t)}{\pi_{\theta_{old}}(u_t \mid x_t)} A(\tau)\right]$$

- If we take the derivative of L^{IS} and evaluate at $\theta = \theta_{old}$, we get the same gradient

$$\nabla_\theta \log f(\theta)\Big|_{\theta_{\text{old}}} = \frac{\nabla_\theta f(\theta)\Big|_{\theta_{\text{old}}}}{f(\theta_{\text{old}})} = \nabla_\theta \left(\frac{f(\theta)}{f(\theta_{\text{old}})}\right)\Bigg|_{\theta_{\text{old}}}$$

Trust Region Policy Optimization (TRPO)

$$\underset{\theta}{\text{maximize}}\ \hat{\mathbb{E}}_t\left[\frac{\pi_\theta\left(u_t \mid x_t\right)}{\pi_{\theta_{old}}\left(u_t \mid x_t\right)}\hat{A}_t\right]$$

$$\text{subject to } \hat{\mathbb{E}}_t\left[\mathrm{KL}[\pi_{\theta_{old}}\left(\cdot \mid x_t\right), \pi_\theta\left(\cdot \mid x_t\right)\right] \leq \delta$$

- Main idea: use trust region to constrain change **in distribution space** (opposed to e.g., parameter space)

- Hard to use with architectures with multiple outputs, e.g., policy and value function
- Empirically performs poorly on tasks requiring deep nets, e.g., deep CNNs, RNNs
- Conjugate gradient makes implementation more complicated

Proximal Policy Optimization (PPO)

- Can we solve the problem defined in TRPO without second-order optimization?

PPO v1 - Surrogate loss with Lagrange multipliers

$$\underset{\theta}{\text{maximize}}\ \hat{\mathbb{E}}_t \left[\frac{\pi_\theta\left(u_t \mid x_t\right)}{\pi_{\theta_{old}}\left(u_t \mid x_t\right)} \hat{A}_t \right] + \beta \left(\hat{\mathbb{E}}_t \left[\mathrm{KL}[\pi_{\theta_{old}}\left(\cdot \mid x_t\right), \pi_\theta\left(\cdot \mid x_t\right) \right] - \delta \right)$$

- Run SGD on the above objective
- Do dual descent update for β

PPO v2 - Clipped surrogate loss

$$r(\theta) = \frac{\pi_\theta\left(u_t \mid x_t\right)}{\pi_{\theta_{old}}\left(u_t \mid x_t\right)}, \quad r(\theta_{old}) = 1$$

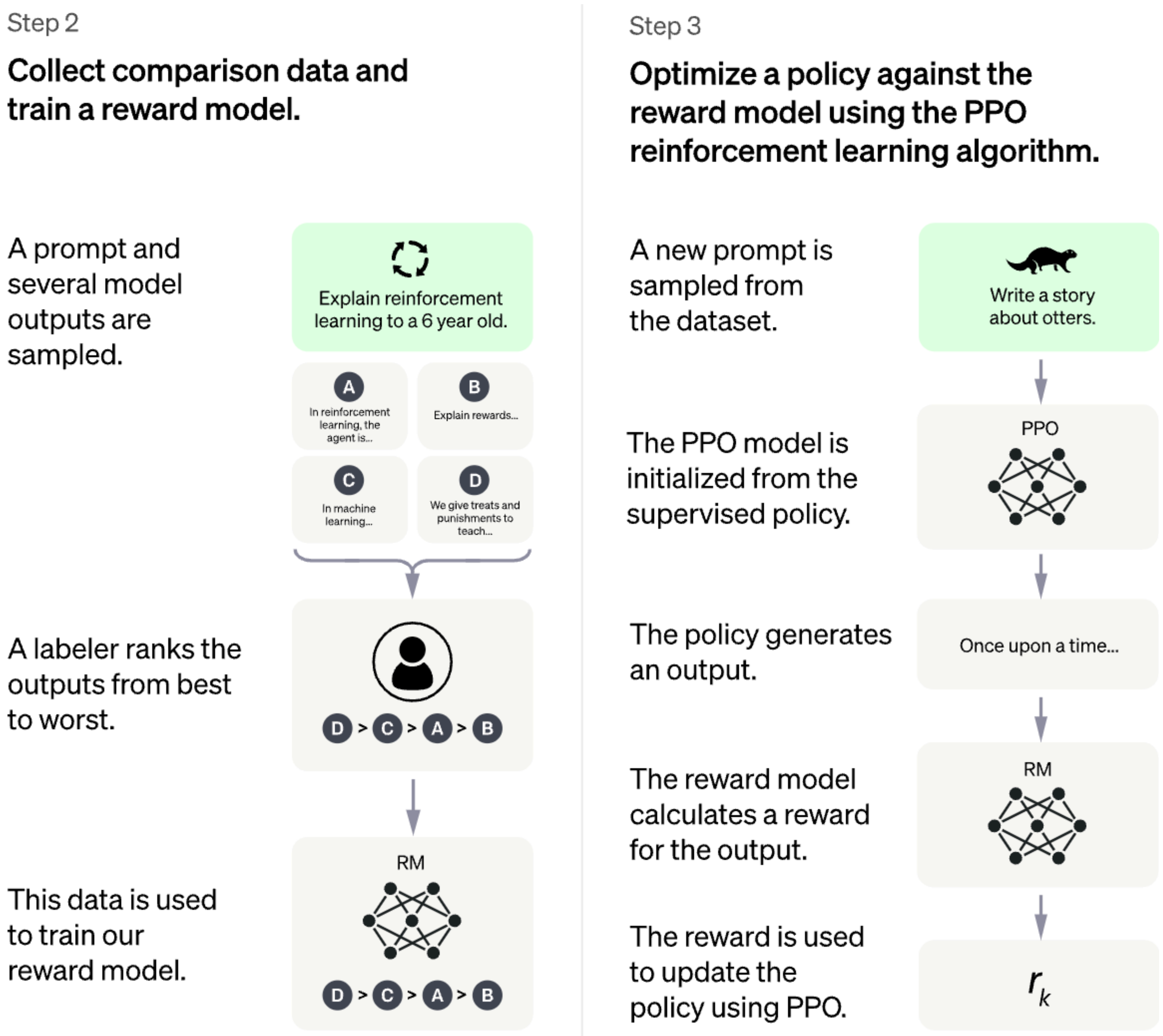
$$\underset{\theta}{\text{maximize}}\ \hat{\mathbb{E}}_t \left[\min(r(\theta)A(\tau), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)A(\tau) \right]$$

- Heuristically replicates constraint in the objective
- One of the (if not the) most popular PO algorithm

Examples: Maze Navigation

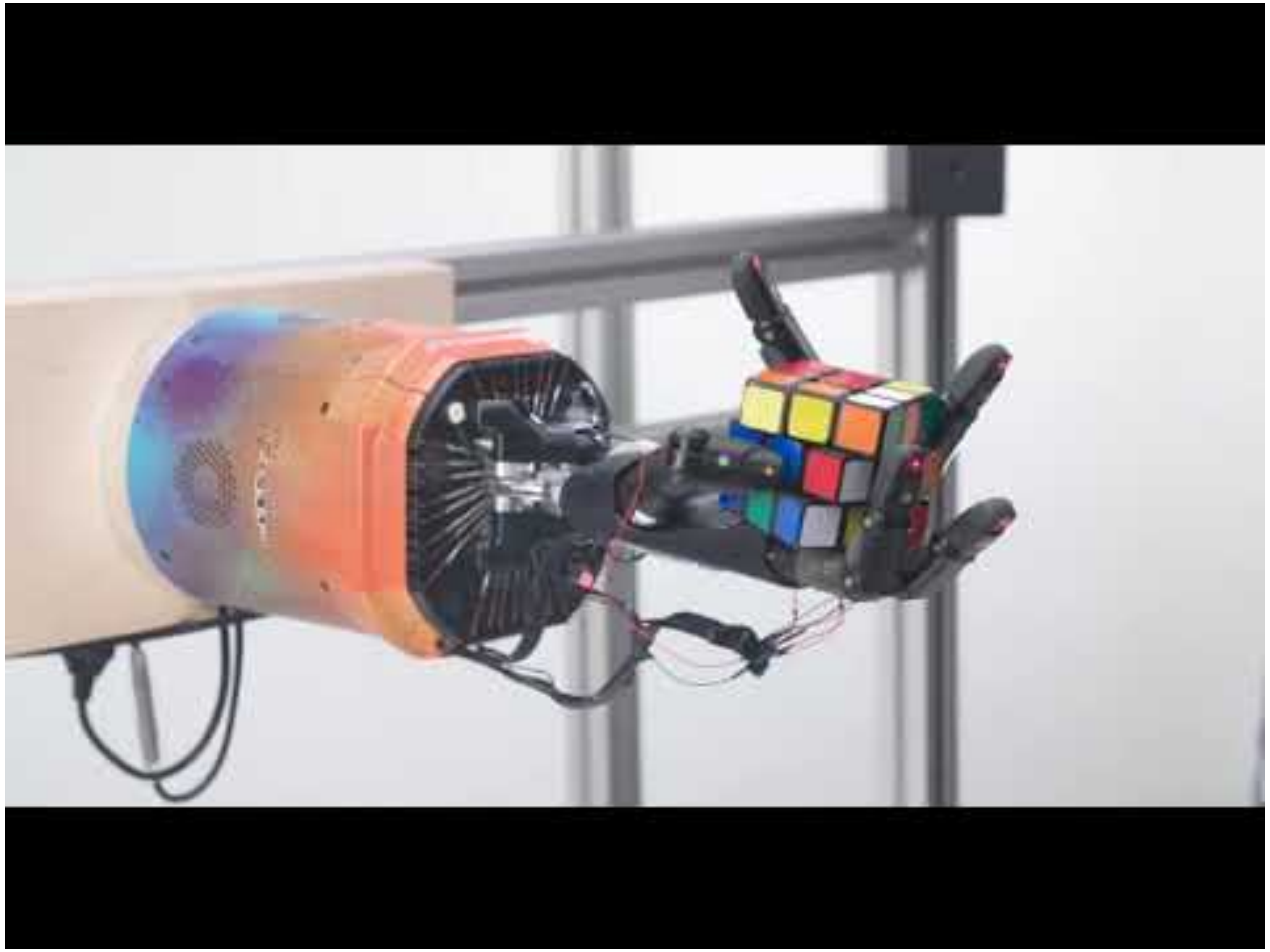
- Mnih et al. 2016 “Asynchronous Methods for Deep Reinforcement Learning"
- Advantage Actor-Critic
- Asynchronous parallel workers
- Policy and Value networks: CNNs & RNNs

Examples: Alignment of ChatGPT



Examples: Robot manipulation

- PPO
- Trained entirely in Sim



Next time

- Model-based RL