

# Policy Gradients

CS 224R

# Course reminders

- Start forming final project groups (survey due next Wednesday)
- Homework 1 out, due Fri April 18

# Recap

state  $\mathbf{s}_t$  - the state of the “world” at time  $t$

action  $\mathbf{a}_t$  - the decision taken at time  $t$

trajectory  $\tau$  - sequence of states/observations and actions

$$(\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \mathbf{a}_2, \dots, \mathbf{s}_T, \mathbf{a}_T)$$

reward function  $r(\mathbf{s}, \mathbf{a})$  - how good is  $\mathbf{s}, \mathbf{a}$ ?

policy  $\pi(\mathbf{a} \mid \mathbf{s})$  or  $\pi(\mathbf{a} \mid \mathbf{o})$  - behavior, usually what we are trying to learn

**Goal:** learn policy  $\pi_\theta$  that maximizes expected sum of rewards:

$$\max_{\theta} \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$\text{where } p(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_t \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$$

## Imitation learning from demonstrations $\mathcal{D}$

$$\min_{\theta} - \mathbb{E}_{(\mathbf{s}, \mathbf{a}) \sim \mathcal{D}} [\log \pi_{\theta}(\mathbf{a} \mid \mathbf{s})]$$

- + Simple, scalable approach for learning performant behavior
- Cannot outperform demonstrator, doesn't allow improvement from practice

### Definitions.

*offline*: using only an existing dataset,  
no new data from learned policy

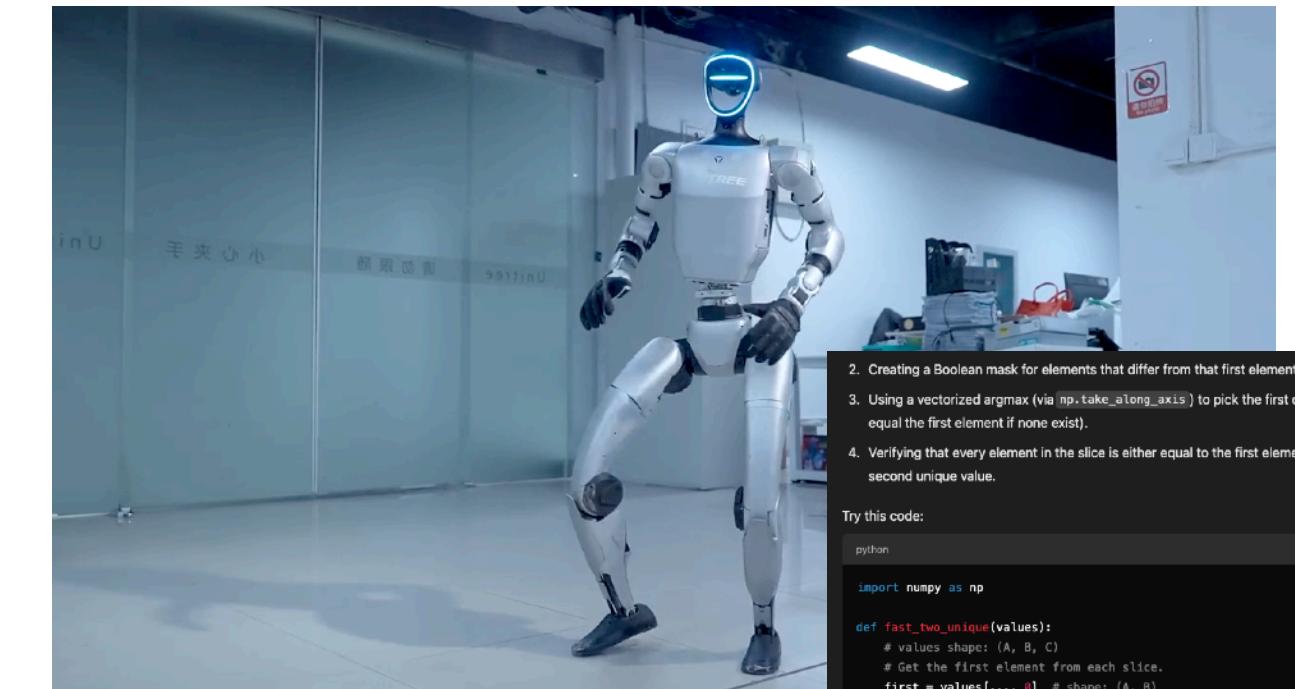
*online*: using new data from learned  
policy

# The plan for today

the basis for:

## Policy gradients: our first online RL algorithm

1. On-policy policy gradient
  - a. Derivation and intuition of policy gradients
  - b. Full algorithm
  - c. How to make it better - causality and baselines
2. Off-policy policy gradients
  - a. Importance sampling
  - b. KL constraints



Source: Unitree

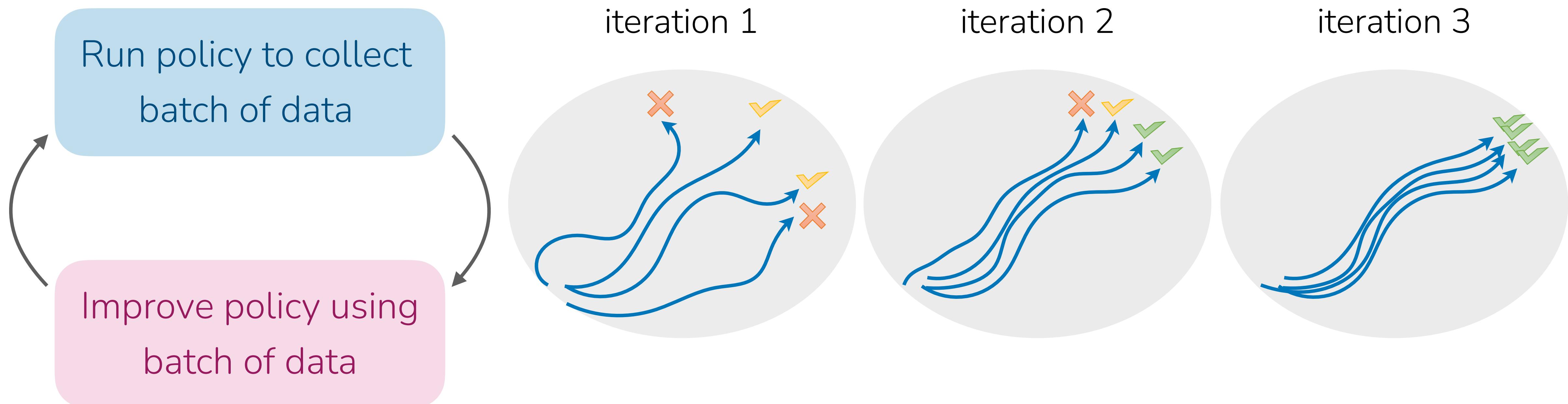
+ part of  
default project

## Key learning goals:

- Key intuition behind policy gradients
- How to implement, when to use policy gradients

# Online RL Outline

First: Initialize the policy (randomly, with imitation learning, with heuristics)

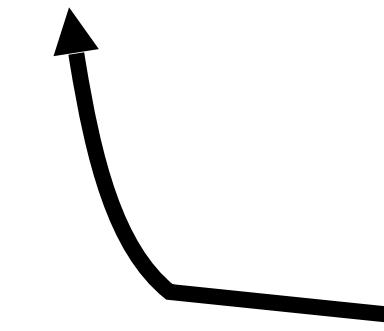


# Evaluating the RL objective

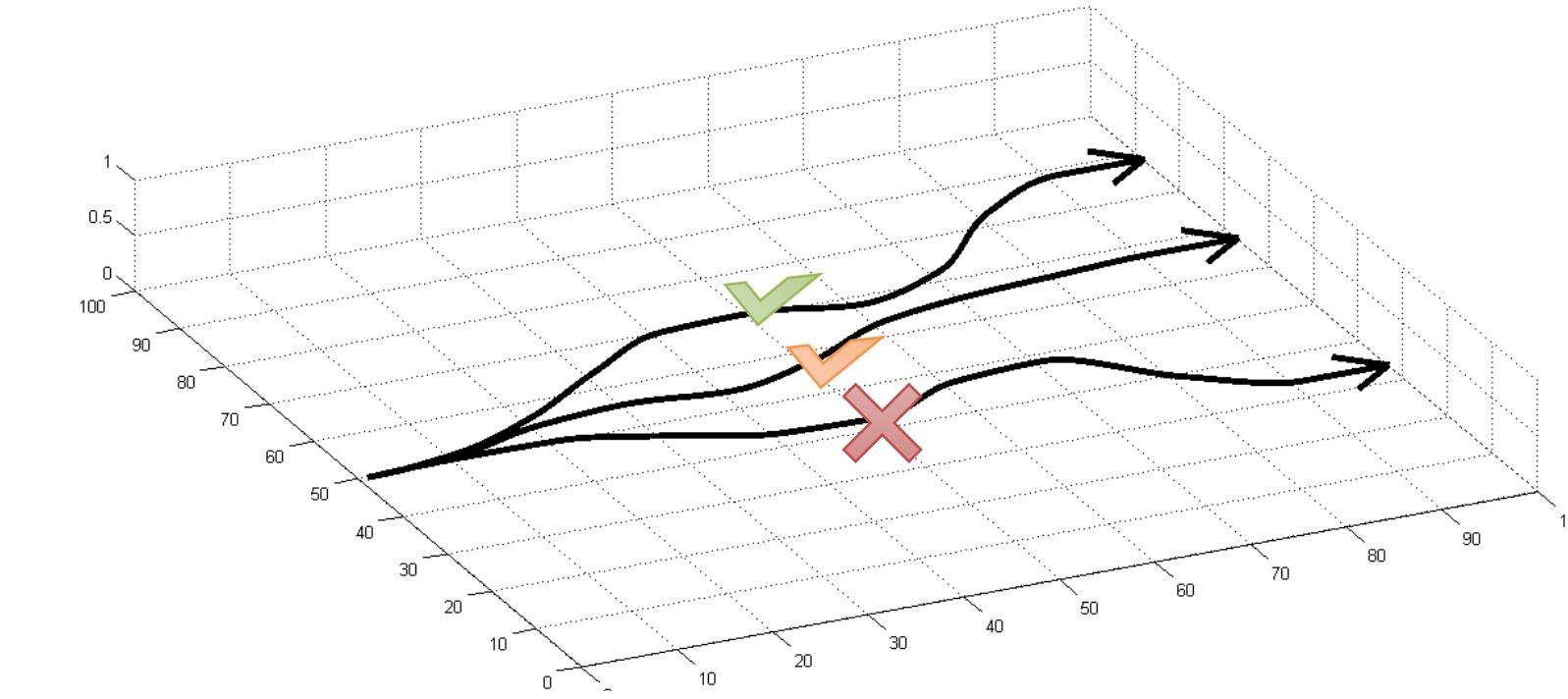
$$\theta^* = \arg \max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[ \sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$J(\theta)$

$$J(\theta) = E_{\tau \sim p_{\theta}(\tau)} \left[ \sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$



sum over samples from  $\pi_{\theta}$



# Can we get the gradient of the RL objective?

Let's start in terms of trajectories.

$$\theta^* = \arg \max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[ \underbrace{\sum_t r(\mathbf{s}_t, \mathbf{a}_t)}_{J(\theta)} \right]$$

a convenient identity

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

$$J(\theta) = E_{\tau \sim p_{\theta}(\tau)} \left[ \underbrace{r(\tau)}_{\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t)} \right] = \int p_{\theta}(\tau) r(\tau) d\tau$$

$$\nabla_{\theta} J(\theta) = \int \underbrace{\nabla_{\theta} p_{\theta}(\tau) r(\tau)}_{\nabla_{\theta} p_{\theta}(\tau) r(\tau)} d\tau = \int \underbrace{p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) r(\tau)}_{p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) r(\tau)} d\tau = E_{\tau \sim p_{\theta}(\tau)} [\nabla_{\theta} \log p_{\theta}(\tau) r(\tau)]$$

# Can we get the gradient of the RL objective?

From trajectories to final form.

$$\nabla_{\theta} J(\theta) = E_{\tau \sim p_{\theta}(\tau)} [\nabla_{\theta} \underbrace{\log p_{\theta}(\tau)}_{\text{green}} r(\tau)]$$

$$\nabla_{\theta} \left[ \log p(\mathbf{s}_1) + \sum_{t=1}^T \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) + \log p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \right]$$

$$p_{\theta}(\tau) = p(\mathbf{s}_1) \prod_{t=1}^T \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\log p_{\theta}(\tau) = \underbrace{\log p(\mathbf{s}_1)}_{\text{green}} + \sum_{t=1}^T \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) + \log p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\nabla_{\theta} J(\theta) = E_{\tau \sim p_{\theta}(\tau)} \left[ \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) \right) \left( \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

# Estimating the gradient

$$\nabla_{\theta} J(\theta) = E_{\tau \sim p_{\theta}(\tau)} \left[ \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) \right) \left( \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

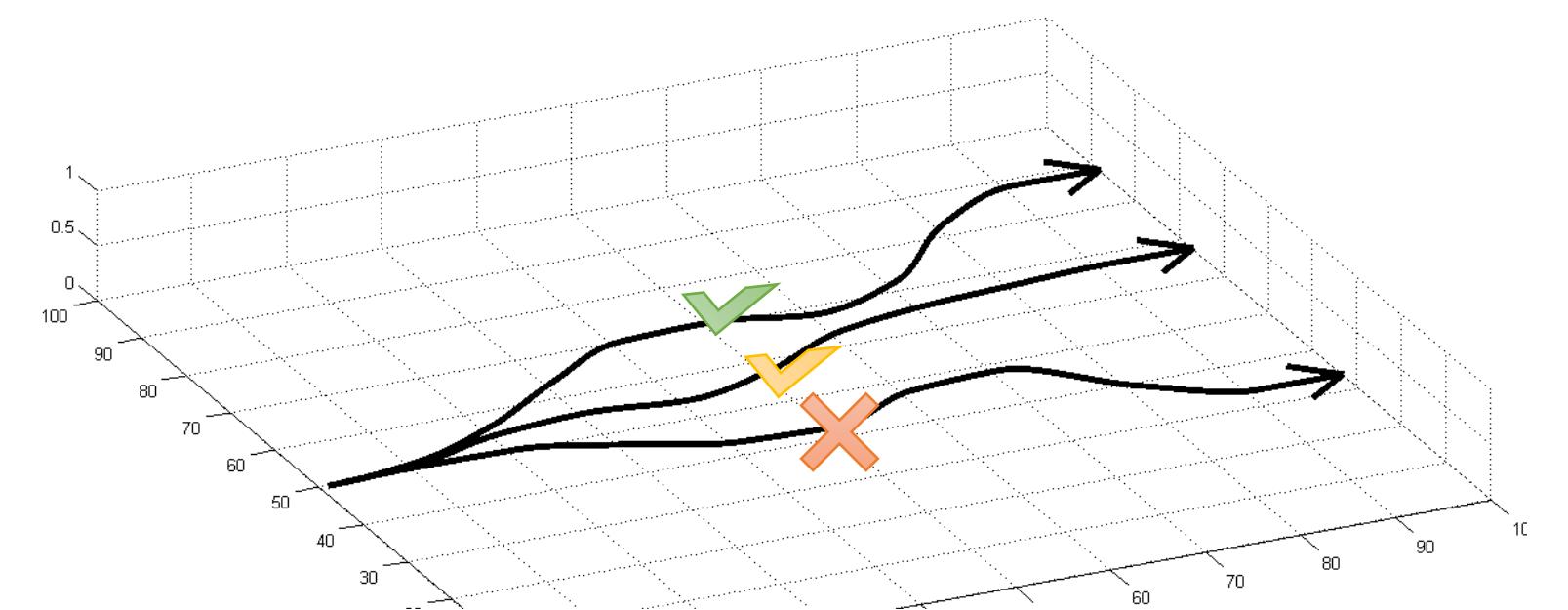
$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left( \sum_{t=1}^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

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

Full algorithm:

1. sample  $\{\tau^i\}$  from  $\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$
2.  $\nabla_{\theta} J(\theta) \approx \sum_i \left( \sum_t \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t^i | \mathbf{s}_t^i) \right) \left( \sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) \right)$
3.  $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$

“REINFORCE algorithm”, vanilla policy gradient



Run policy to collect  
batch of data

Improve policy using  
batch of data

# What does the gradient mean?

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left( \sum_{t=1}^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

Imitation gradient, but weighted by reward

Recall: imitation learning

$$\min_{\theta} -\mathbb{E}_{(\mathbf{s}, \mathbf{a}) \sim \mathcal{D}} [\log \pi_{\theta}(\mathbf{a} | \mathbf{s})]$$

$$\nabla_{\theta} J_{\text{BC}}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right)$$

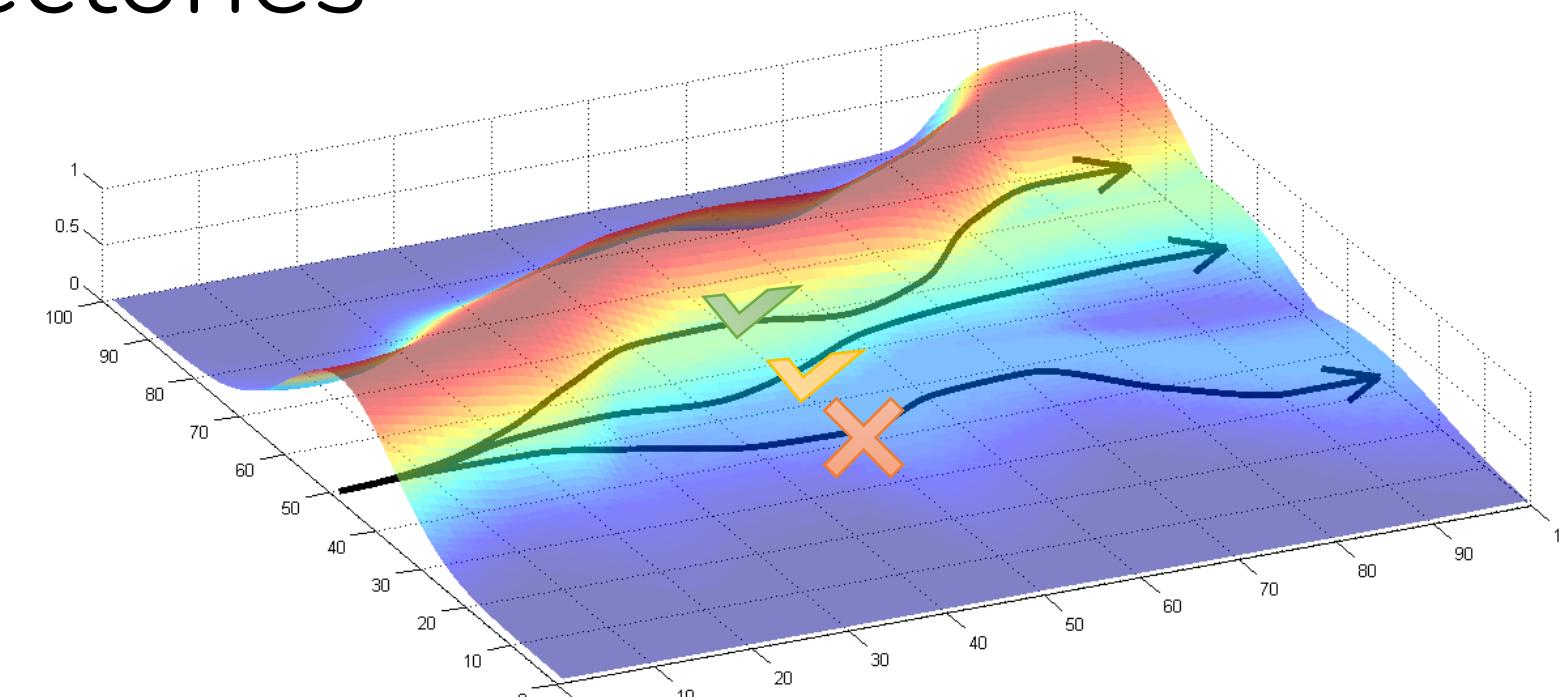
Intuition:

Increase likelihood of actions you took in high reward trajectories.

Decrease likelihood of actions you took in negative reward trajectories

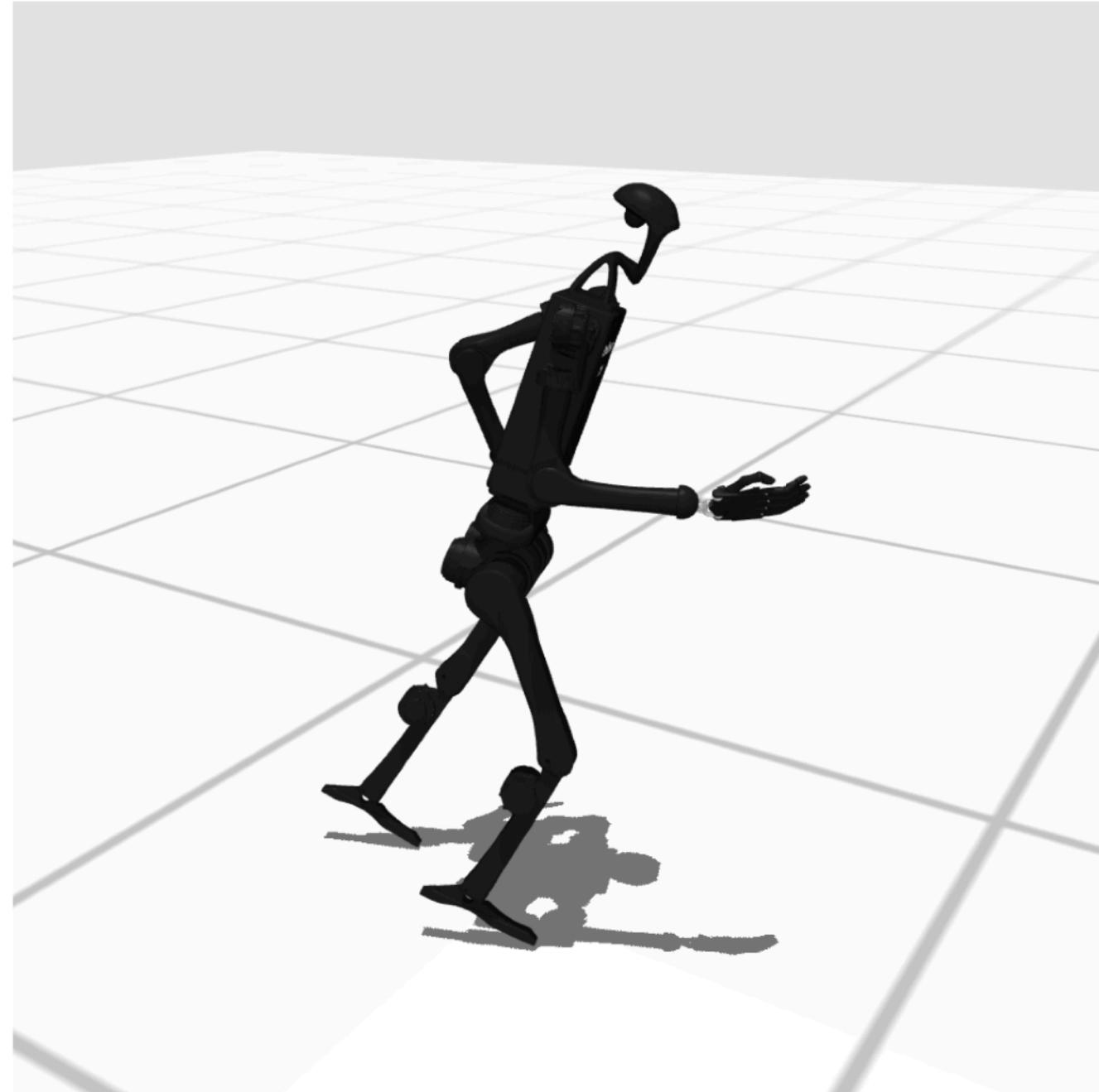
i.e. do more of the good stuff, less of the bad stuff.

formalization of “trial-and-error”



# What does the gradient do?

Example: learning humanoid walking in simulation



reward:  $r(\mathbf{s}, \mathbf{a})$  = forward velocity of robot

(can be negative if robot goes backwards)

1. sample  $\{\tau^i\}$  from  $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$

$\tau^1$ : falls backwards ✗

$\tau^2$ : falls forwards ✓

$\tau^3$ : manages to stand still ✓

$\tau^4$ : one small step forward then falls backwards ✗

$\tau^5$ : one large step backwards then small step forwards ✗

2.  $\nabla_\theta J(\theta) \approx \sum_i \left( \sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i | \mathbf{s}_t^i) \right) \left( \sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) \right)$

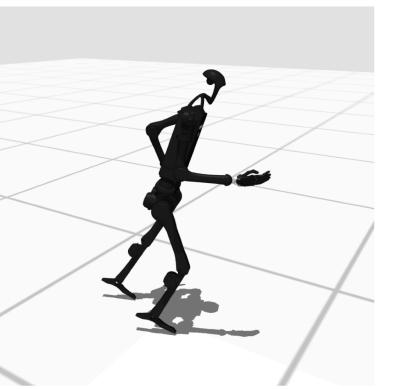
**Question:** what will the gradient encourage the policy to do?

-> will encourage policy to fall forward, and not take step forward 😞

# Improving the gradient

Using causality

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left( \sum_{t=1}^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$



$\tau^5$ : one large step backwards  
then small step forwards X

Policy behavior at time  $t$  does not affect rewards at time  $t' < t$

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left( \sum_{t' \neq t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

sum of *future* rewards

# What does the gradient do?

Example: learning humanoid walking in simulation



reward:  $r(\mathbf{s}, \mathbf{a})$  = forward velocity of robot

(can be negative if robot goes backwards)

1. sample  $\{\tau^i\}$  from  $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$

$\tau^1$ : falls forwards

$\tau^2$ : slowly stumbles forwards

$\tau^3$ : steadily walks forwards

$\tau^4$ : runs forwards

2.  $\nabla_\theta J(\theta) \approx \sum_i \left( \sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i | \mathbf{s}_t^i) \right) \left( \sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) \right)$

Question: what will the gradient encourage the policy to do?

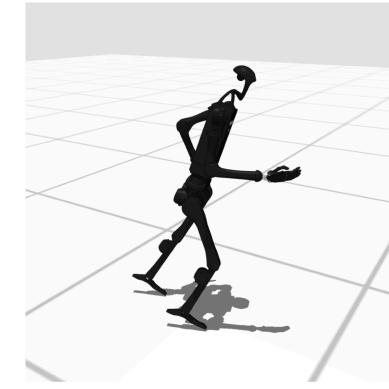
-> encourages policy to fall, stumble forward some of the time

Policy gradient is **noisy / high-variance**  
**sensitive to reward scale**

# Improving the gradient

Introducing baselines

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



$\tau^1$ : falls forwards ✓

$\tau^4$ : runs forwards ✓

# Improving the gradient

Introducing baselines

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

$$b = \frac{1}{N} \sum_{i=1}^N r(\tau)$$

subtracting a **constant**  
“baseline”



a convenient identity

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

$\tau^1$ : falls forwards ✓

$\tau^4$ : runs forwards ✓

If we subtract average reward, we get negative gradients for below-average behavior. 🤦

But, can we even do that? 🤔

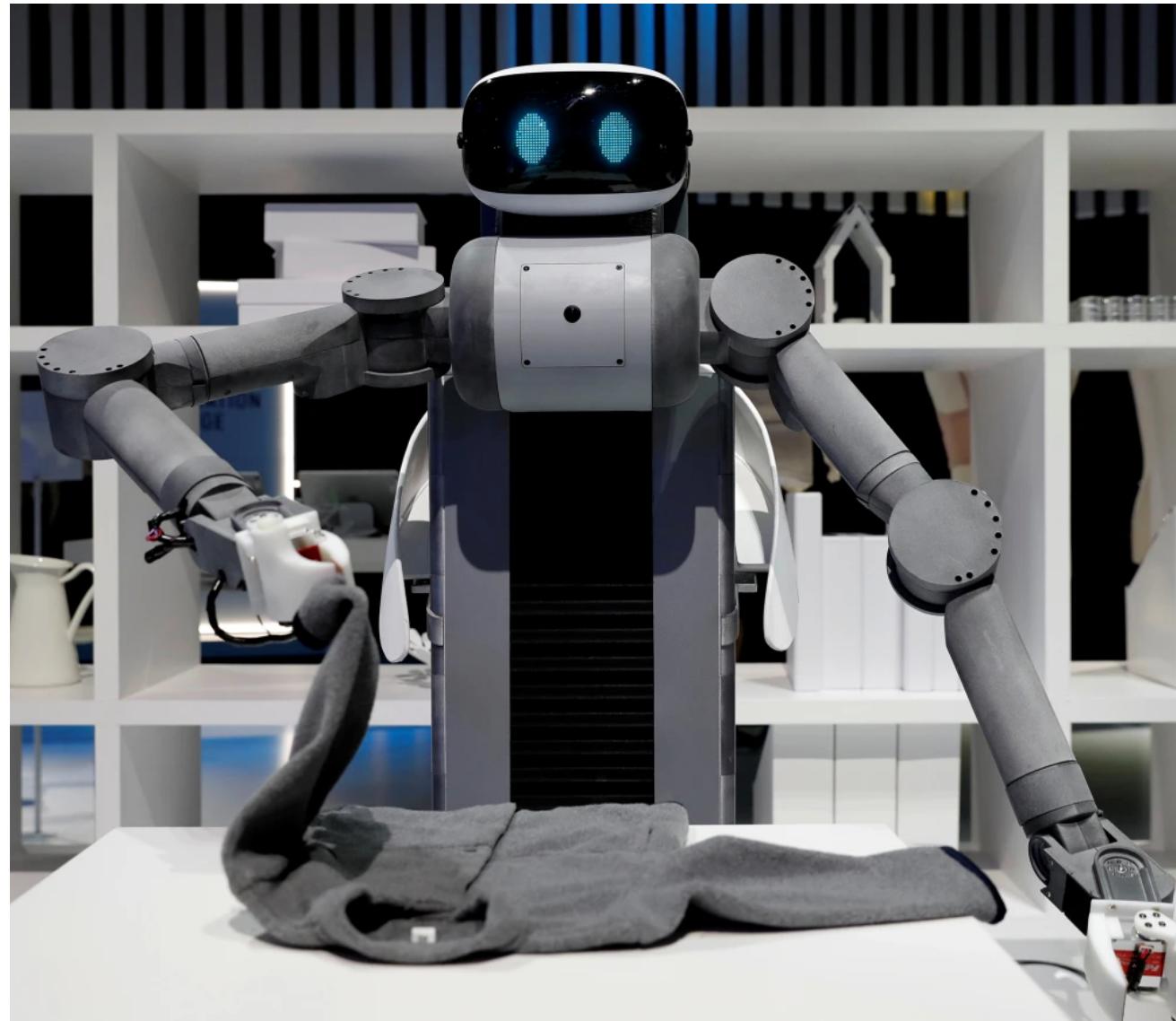
$$E[\nabla_{\theta} \log p_{\theta}(\tau) b] = \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) b d\tau = \int \nabla_{\theta} p_{\theta}(\tau) b d\tau = b \nabla_{\theta} \int p_{\theta}(\tau) d\tau = b \nabla_{\theta} 1 = 0$$

Subtracting a constant baseline does not change the gradient in expectation. It is **unbiased**.  
and can reduce **variance** of the gradient

Average reward is a pretty good baseline.

# What does the gradient do? (baseline edition)

Example: learning to fold a jacket



reward:  $r(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{neatly folded} \\ 0.5 & \text{folded with some wrinkles} \\ 0 & \text{not folded} \end{cases}$

1. sample  $\{\tau^i\}$  from  $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$   
 $\tau^1$ : doesn't touch the jacket       $\tau^3$ : flattens the jacket but does not fold it  
 $\tau^2$ : folds only the sleeves       $\tau^4$ : folds the jacket
2.  $\nabla_\theta J(\theta) \approx \sum_i \left( \sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i | \mathbf{s}_t^i) \right) \left( \sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) - b \right)$

**Question: what will the gradient encourage the policy to do?**

-> will encourage folding, but gradient is constant for all but one trajectory 😰

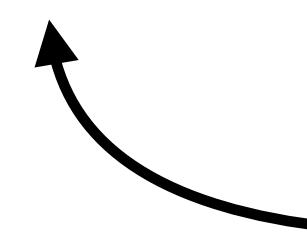
Policy gradient is still **noisy / high-variance**

Best with **dense rewards, large batches.**

# How to implement policy gradients?

Our gradient

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left( \left( \sum_{t'=t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$



Computing these individually is inefficient.  
( $N*T$  backwards passes)

Can we use automatic differentiation on full objective?

Implement “surrogate objective” whose gradient is the same as  $\nabla J$

$$\tilde{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=1}^T \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left( \left( \sum_{t'=t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

weighted maximum likelihood



Cross-entropy for discrete action policy, squared error for Gaussian policy

# Summary so far

## Estimating gradient of RL objective

- log gradient trick
- weigh policy likelihood by future rewards
- subtract baseline (e.g. average reward)
- even with tricks, gradient is **noisy**

## First reinforcement learning algorithm

- collect batch of data, **improve policy by applying gradient**
- formalizes trial-and-error learning

Key intuition: do more high reward stuff, less low reward stuff



# What else is troublesome about policy gradients?

Latest version of our gradient:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[ \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) \right) \left( \left( \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) - b \right) \right]$$

attribute of online RL algorithms

assumes samples from current policy  $\pi_{\theta}$

Full algorithm:



1. sample  $\{\tau^i\}$  from  $\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$
2. compute  $\nabla_{\theta} J(\theta)$  using  $\{\tau^i\}$
3.  $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$     <- we change  $\theta$  right here

Definitions.

on-policy: update uses only data from current policy

off-policy: update can reuse data from other, past policies

Need to recollect data every gradient step! 🍺

Vanilla policy gradient is **on-policy**.

# Off-policy version of policy gradient?

Importance sampling

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}[r(\tau)]$$

What if we want to use samples from  $\bar{p}(\tau)$ ?  
(e.g. previous policy)

$$J(\theta) = E_{\tau \sim \bar{p}(\tau)} \left[ \frac{p_\theta(\tau)}{\bar{p}(\tau)} r(\tau) \right]$$

$$\frac{p_\theta(\tau)}{\bar{p}(\tau)} = \frac{\cancel{p(s_1)} \prod_{t=1}^T \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \cancel{p(s_{t+1} | s_t, \mathbf{a}_t)}}{\cancel{p(s_1)} \prod_{t=1}^T \bar{\pi}(\mathbf{a}_t | \mathbf{s}_t) \cancel{p(s_{t+1} | s_t, \mathbf{a}_t)}} = \frac{\prod_{t=1}^T \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)}{\prod_{t=1}^T \bar{\pi}(\mathbf{a}_t | \mathbf{s}_t)}$$

## Importance sampling

Using proposal distribution  $q$

$$\begin{aligned} E_{x \sim p(x)}[f(x)] &= \int p(x) f(x) dx \\ &= \int \frac{q(x)}{q(x)} p(x) f(x) dx \\ &= \int q(x) \frac{p(x)}{q(x)} f(x) dx \\ &= E_{x \sim q(x)} \left[ \frac{p(x)}{q(x)} f(x) \right] \end{aligned}$$

Note: Important for  $q$  to have non-zero support for high probability  $p(x)$

# Off-policy version of policy gradient?

Importance sampling

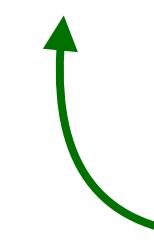
$$\frac{p_\theta(\tau)}{\bar{p}(\tau)} = \frac{\prod_{t=1}^T \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)}{\prod_{t=1}^T \bar{\pi}(\mathbf{a}_t | \mathbf{s}_t)}$$

Say we want to update our latest policy  $\pi_{\theta'}$  but we want to use samples from  $\pi_\theta$

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[ \left( \sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \right) \left( \left( \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) - b \right) \right]$$

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \frac{p_{\theta'}(\tau)}{p_\theta(\tau)} \left( \sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \right) \left( \left( \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) - b \right) \right]$$

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \prod_{t=1}^T \frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left( \sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \right) \left( \left( \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) - b \right) \right]$$



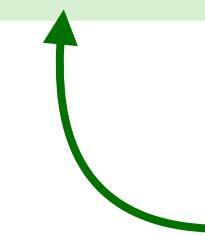
This can become very small or very large, for larger  $T$

# Off-policy version of policy gradient?

Importance sampling

Say we want to update our latest policy  $\pi_{\theta'}$  but we want to use samples from  $\pi_{\theta}$

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[ \prod_{t=1}^T \frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)} \left( \sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \right) \left( \left( \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) - b \right) \right]$$



This can become very small or very large, for larger  $T$

What if we consider the expectation over timesteps instead of trajectories?

$$\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \frac{\pi_{\theta'}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}{\pi_{\theta}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left( \left( \sum_{t'=t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$



Much less likely to explode/vanish ...but, hard to measure

$$\frac{\pi_{\theta'}(\mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{s}_{i,t})}$$

Common final form

$$\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \frac{\pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left( \left( \sum_{t'=t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

often approximated as 1

# Off-policy policy gradient

Common final form

$$\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \frac{\pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left( \left( \sum_{t'=t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

Full algorithm:

1. sample  $\{\tau^i\}$  from  $\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$
2. compute  $\nabla_{\theta} J(\theta)$  using  $\{\tau^i\}$  Can take multiple gradient steps on the same batch
3.  $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$

Run policy to collect batch of data

Improve policy using batch of data

What if our policy changes **a lot** before sampling new data?

Data no longer reflects states that policy will visit. Gradient estimate less accurate.

# Off-policy policy gradient

Common final form

$$\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \frac{\pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left( \left( \sum_{t'=t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

What if our policy changes **a lot** before sampling new data?

Data no longer reflects states that policy will visit. Gradient estimate less accurate.

Can we constrain the policy to not stray too far during gradient updates?

One common choice:  $\mathbb{E}_{\mathbf{s} \sim \pi_{\theta}} [D_{KL}(\pi_{\theta'}(\cdot | \mathbf{s}) \| \pi_{\theta}(\cdot | \mathbf{s}))] \leq \delta$

# Review

## Online RL via policy gradients

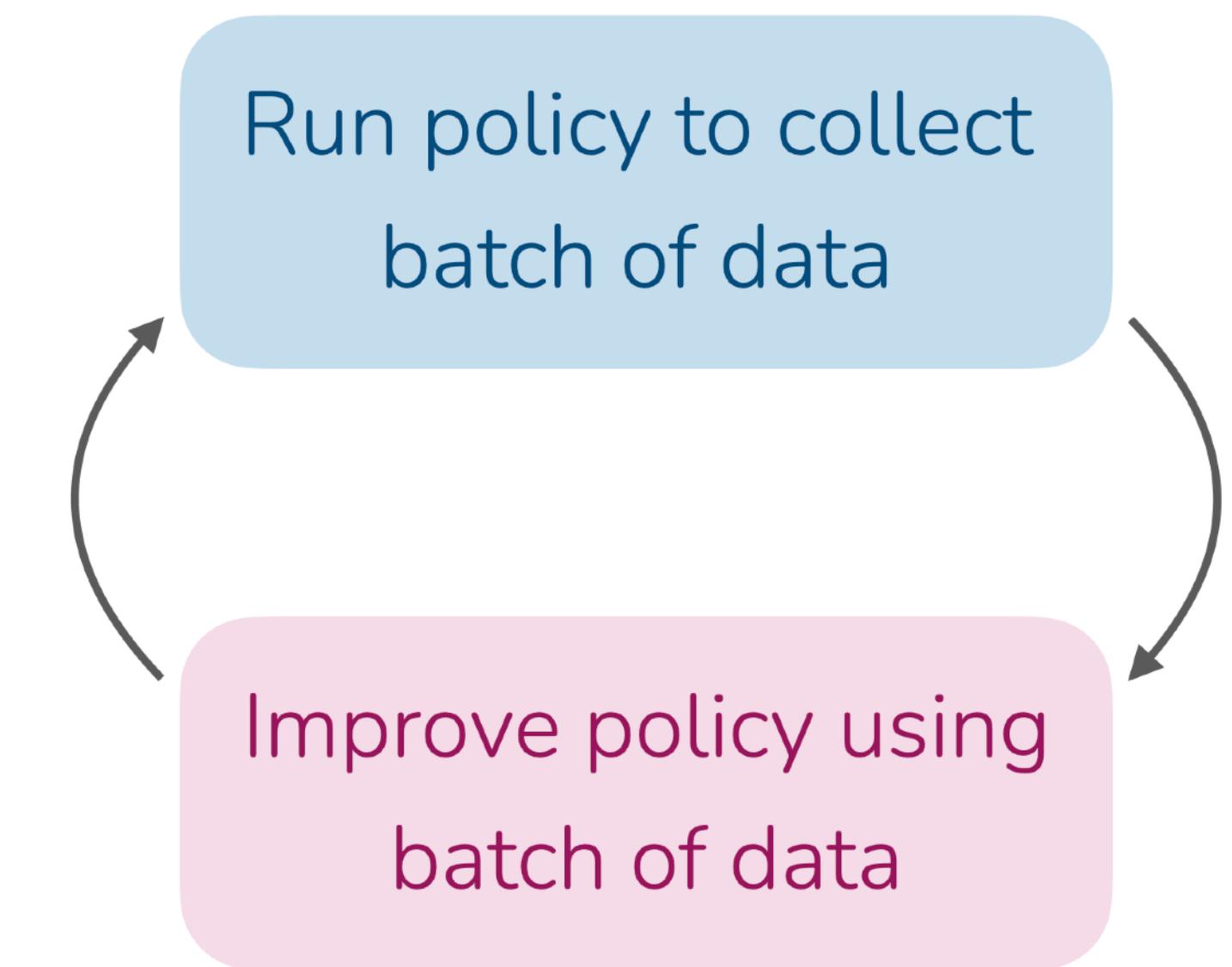
- On-policy algorithm, differentiating the RL objective
- Baselines, causality for reducing gradient variance
- collect batch of data, improve policy by applying gradient

## Derived off-policy policy gradient

- importance sampling
- KL constraint on policy
- collect batch of data, apply multiple gradient updates

## Intuition

- Do more high reward stuff, less low reward stuff
- Gradient still very noisy, best with large batch sizes and dense rewards

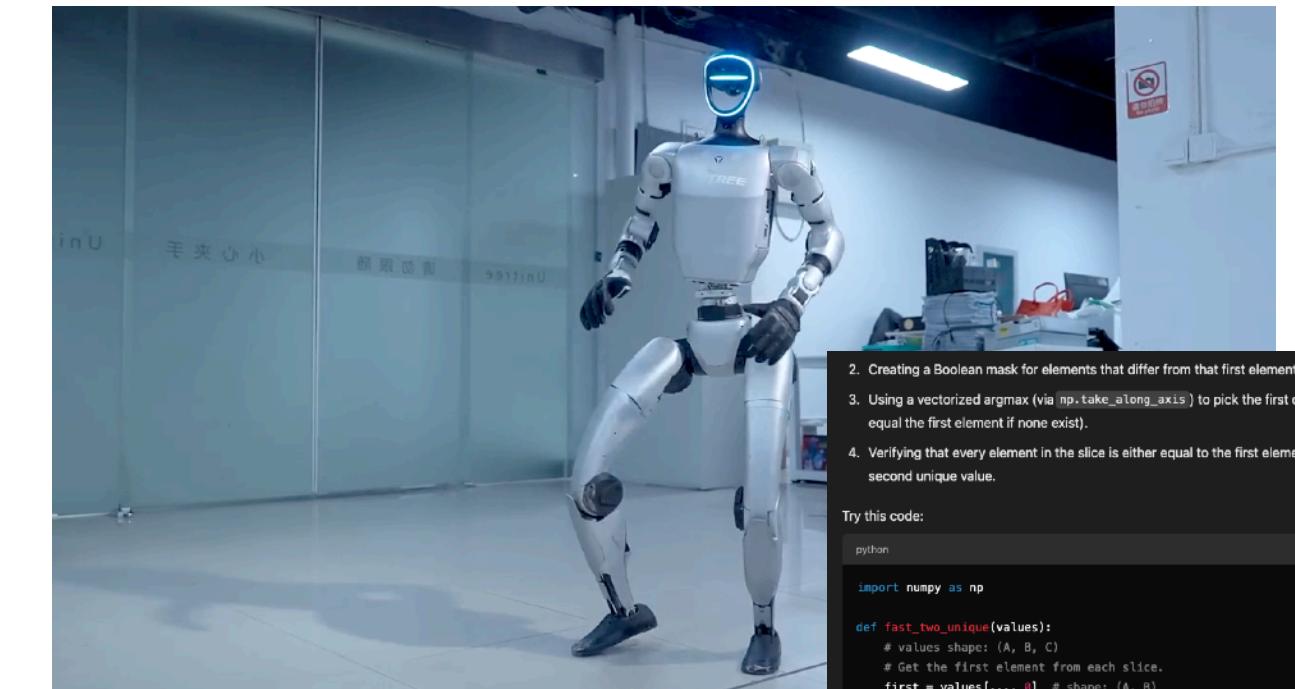


# The plan for today

the basis for:

## Policy gradients: our first online RL algorithm

1. On-policy policy gradient
  - a. Derivation and intuition of policy gradients
  - b. Full algorithm
  - c. How to make it better - causality and baselines
2. Off-policy policy gradients
  - a. Importance sampling
  - b. KL constraints



Source: Unitree

+ part of  
default project

## Key learning goals:

- Key intuition behind policy gradients
- How to implement, when to use policy gradients

```
2. Creating a Boolean mask for elements that differ from that first element.
3. Using a vectorized argmax (via np.take_along_axis) to pick the first different value (which will
equal the first element if none exist).
4. Verifying that every element in the slice is either equal to the first element or that candidate
second unique value.

Try this code:
python ⌂ Copy
import numpy as np

def fast_two_unique(values):
    # values shape: (A, B, C)
    # Get the first element from each slice.
    first = values[...,:,0] # shape: (A, B)

    # Create a mask that is True where elements differ from the first element.
    mask = (values != first[...,:,None])

    # Find, for each slice, the index of the first differing element.
    # np.argmax returns 0 if no True exists; that's fine because then candidate2 equals
    # first.
    idx = np.argmax(mask, axis=2)

    # Extract the candidate second unique value.
    candidate2 = np.take_along_axis(values, idx[...,:,None], axis=2).squeeze(axis=2)

    # Check that every element is either equal to 'first' or 'candidate2'.
    return np.all((values == first[...,:,None]) | (values == candidate2[...,:,None]))

# values is a boolean array indicating, for each (i, j) slice, if the
# first element: values[...,:,0] so each (i, j) slice's first value becomes our reference.
# mask[...,:,0] identifies elements that differ from the first value.
# idx[...,:,0] identifies the first index along axis 2 where the element is different. If all elements
# are equal, np.argmax returns 0 and candidate2 will equal first.
# That's fine because then candidate2 equals first.
# That's fine because then candidate2 equals first.

# This vectorized solution should be considerably faster than using apply_along_axis with a Python
# loop.

That works great! Can you generalize the function to work with
any axis, not just axis 2?

Reasoned about generalized vectorized solution for 5 seconds.
Below is a generalized version that lets you choose the axis along which to check for at most two
unique values. It works by moving the target axis to the end, applying the logic, and then reducing
that axis.

python ⌂ Copy
def fast_n_unique(values, n=2, axis=-1):
    # values shape: (A, B, C, D, ..., N)
    # Get the first element from each slice.
    first = values[...,:,:,0] # shape: (A, B, C, ..., N)

    # Create a mask that is True where elements differ from the first element.
    mask = (values != first[...,:,:,None])

    # Find, for each slice, the index of the first differing element.
    # np.argmax returns 0 if no True exists; that's fine because then candidate2 equals
    # first.
    idx = np.argmax(mask, axis=axis)

    # Extract the candidate second unique value.
    candidate2 = np.take_along_axis(values, idx[...,:,:,None], axis=axis).squeeze(axis=axis)

    # Check that every element in each slice is either equal to 'first' or 'candidate2'.
    return np.all((values == first[...,:,:,None]) | (values == candidate2[...,:,:,None]))
```

# Next time

## *Actor critic methods*

- > build closely on policy gradients!
- > basis for popular algorithms like PPO

## Course reminders

- Start forming final project groups (survey due next Wednesday)
- Homework 1 out, due next Friday