

PGs directly attempt to differentiate RL objective, then perform gradient decent on policy parameters to make the policy better!

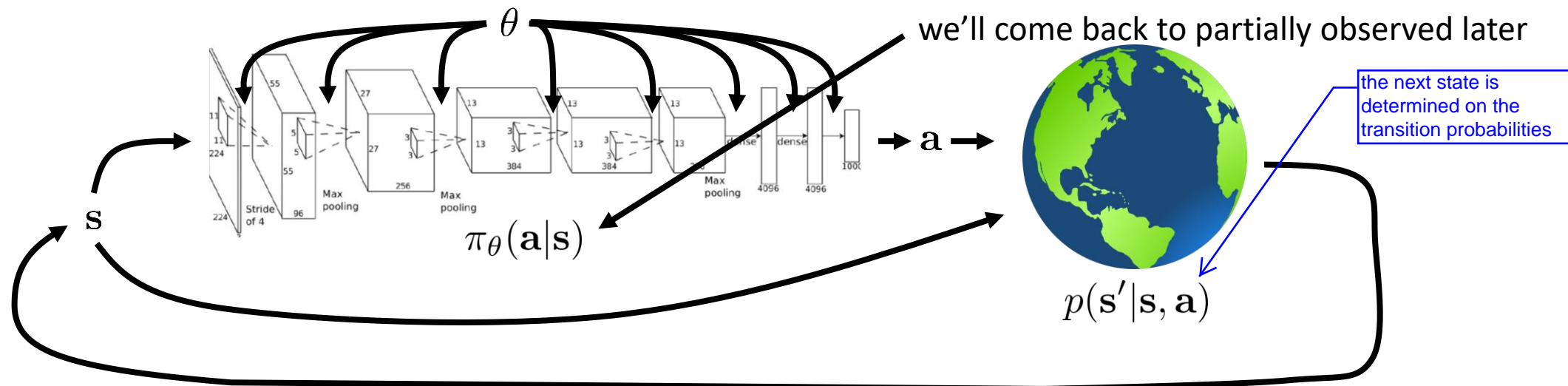
Policy Gradients

CS 285

Instructor: Sergey Levine
UC Berkeley



The goal of reinforcement learning



Input: State
Output: Action

Theta: the weights/parameters of the policy!

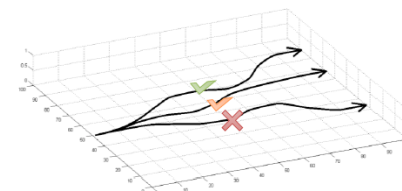
$$\underbrace{p_{\theta}(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T)}_{p_{\theta}(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^T \pi_{\theta}(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

Trajectory distribution is a probability distribution over a sequence of states and actions. It depends on the policy (and its parameters theta).

For model-free, we don't assume we know the transition probabilities or the initial state probability $p(\mathbf{s}_1)$, we just assume we can interact with the real world, which effectively samples from the distribution

goal: find parameters theta that maximize this expectation

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



The goal of reinforcement learning

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

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

infinite horizon case

$$\theta^* = \arg \max_{\theta} \sum_{t=1}^T E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_{\theta}(\mathbf{s}_t, \mathbf{a}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)]$$

finite horizon case

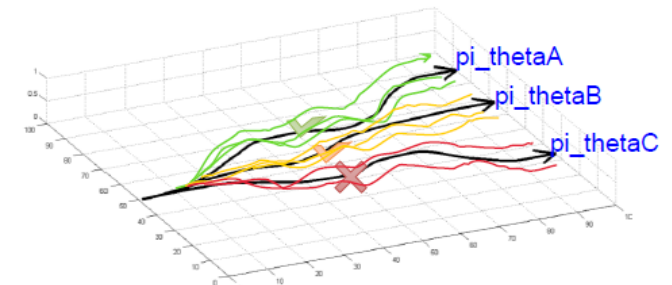
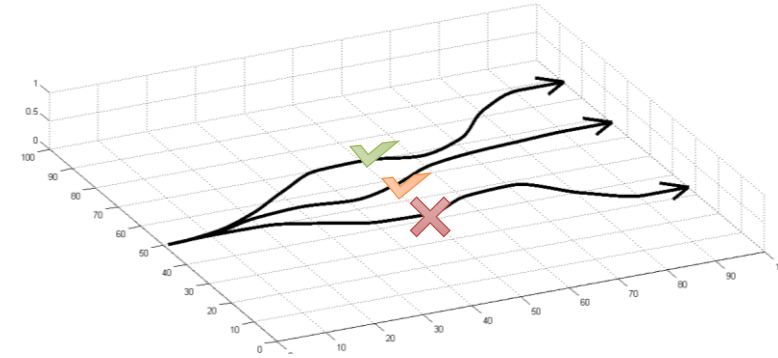
Evaluating the objective

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

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

How do we estimate J? Trial and error! We can approximate J(theta) by doing multiple rollouts of our policy! Then we can sum up the rewards for each rollout, then average over all the rollouts.

sum over samples from π_{θ}



Direct policy differentiation

But we don't just want to estimate the objective we want to improve it! How do we improve it? Well, first we need to figure out how to estimate its derivative! This will allow us to use gradient decent to update the weights and improve the policy!

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

a convenient identity

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

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

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

Direct policy differentiation

$$\begin{aligned}
 \theta^* &= \arg \max_{\theta} J(\theta) \\
 J(\theta) &= E_{\tau \sim p_{\theta}(\tau)}[r(\tau)] \\
 \nabla_{\theta} J(\theta) &= E_{\tau \sim p_{\theta}(\tau)}[\nabla_{\theta} \log p_{\theta}(\tau) r(\tau)]
 \end{aligned}$$

log of both sides

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

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

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

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

Evaluating the policy gradient

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

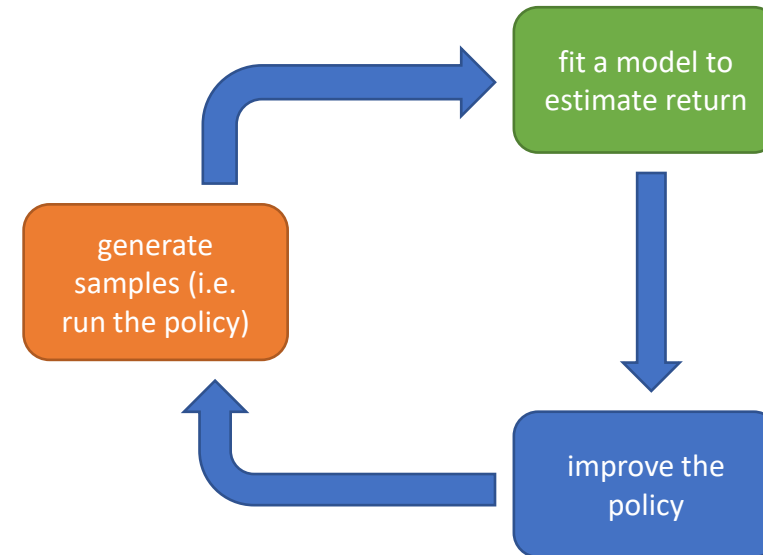
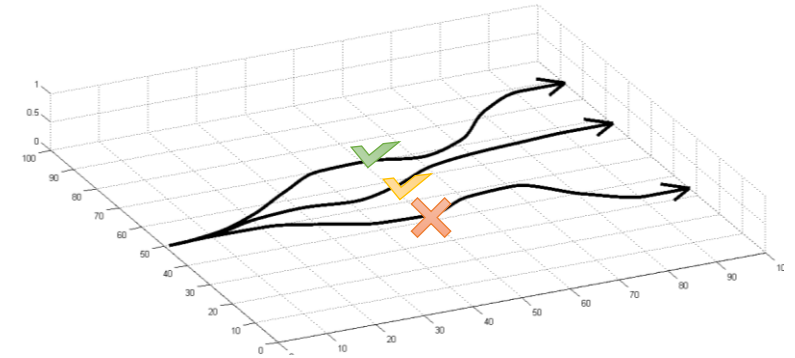
$$\nabla_{\theta} J(\theta) = E_{\tau \sim p_{\theta}(\tau)} \left[\left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

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

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

REINFORCE algorithm:

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



Understanding Policy Gradients

Evaluating the policy gradient

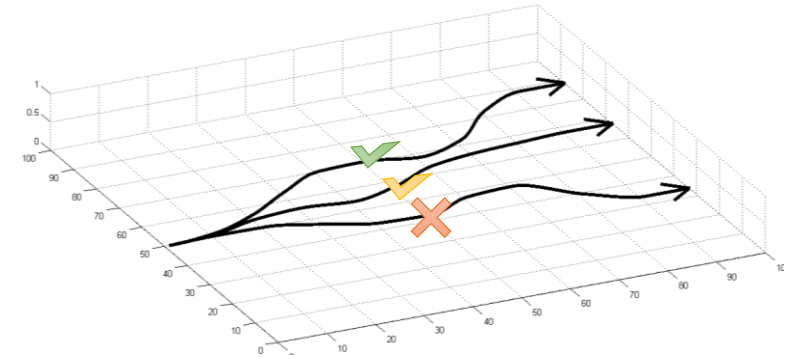
recall: $J(\theta) = E_{\tau \sim p_{\theta}(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$

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

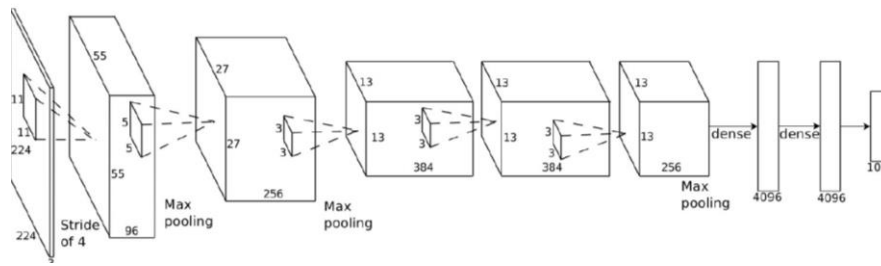
$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

what is this?

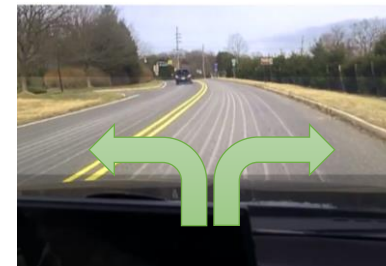
- the $\log(\pi)$ is the log probability that the policy assigns to one of those two actions
- $\text{grad } \log(\pi)$ is the derivative of that probability
- so a NN will output the probs, and you can take the log of those probs.
- when you do MLE in supervised learning, you want to maximize the log probs of observed labels



\mathbf{s}_t



$\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$



\mathbf{a}_t

Comparison to maximum likelihood

supervised learning.

we do this in imitation learning

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

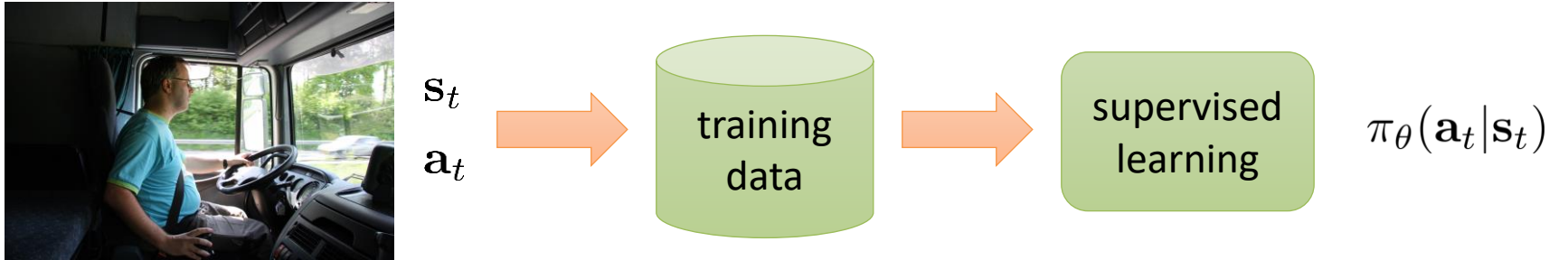
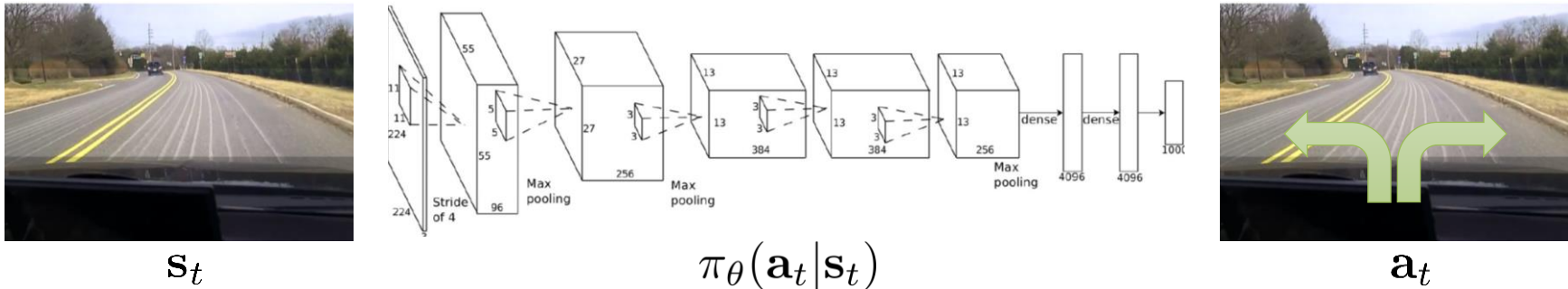
when we do PG, we can't assume the actions in our data are "good", because we generated those actions by running our previous policy, which may not have been any good.

- taking the PG gradient may increase OR decrease the logprob of our actions, depending on the value of their reward. Intuitively, higher reward trajs get their log probs increased, while lower reward trajs get their log probs decreased. It's like a weighted version for the gradient for MLE

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

- when we do MLE, we assume the actions in our data are "good" actions to take.

- thus, the MLE gradient increase the logprobs of all our actions.



Example: Gaussian policies

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

- we need a representation of π that can output distributions over continuous actions

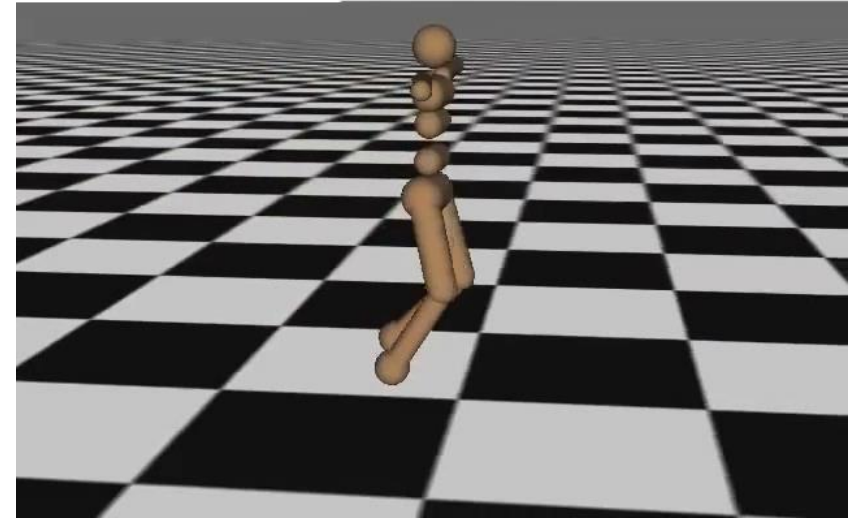
example: $\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) = \mathcal{N}(f_{\text{neural network}}(\mathbf{s}_t); \Sigma)$

$$\log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) = -\frac{1}{2} \|f(\mathbf{s}_t) - \mathbf{a}_t\|_{\Sigma}^2 + \text{const}$$

$$\nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) = -\frac{1}{2} \Sigma^{-1} (f(\mathbf{s}_t) - \mathbf{a}_t) \frac{df}{d\theta}$$

we could represent the policy as a multi-variate normal/gaussian distribution where the mean (and possibly the variance) is learned by the NN

Iteration 2000



REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$ (run it on the robot)
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. $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$

What did we just do?

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

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

good stuff is made more likely

bad stuff is made less likely

simply formalizes the notion of “trial and error”!

REINFORCE algorithm:

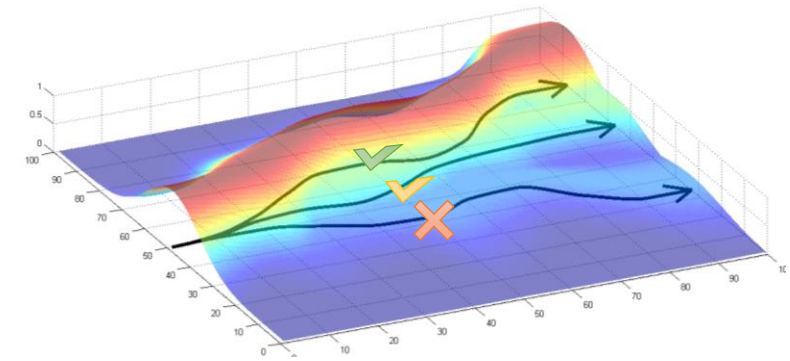
1. sample $\{\tau^i\}$ from $\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$ (run it on the robot)
2. $\nabla_{\theta} J(\theta) \approx \sum_i \left(\sum_t \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t^i | \mathbf{s}_t^i) \right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) \right)$
3. $\theta \leftarrow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

- what's the PG doing?

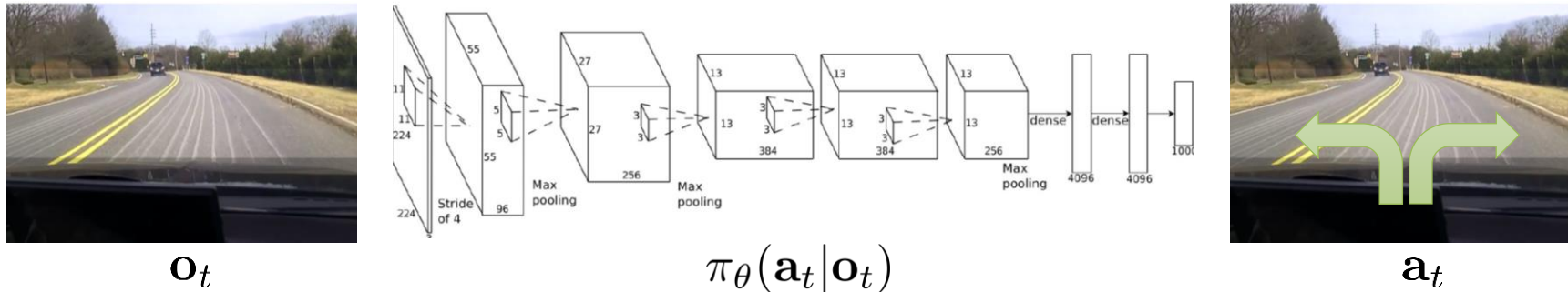
- we want to take the logprobs of the good trajectories and increase them (make it more likely to do good trajectories). we want to take the logprobs of bad trajectories and lower them (make it less likely to do bad trajectories)

similar to PG except we don't have the rewards. I think we can ignore the rewards because we can assume that they are all the same. There's no notion of "more correct". This makes sense to me in a discrete scenario, but not sure I understand in a continuous scenario

maximum likelihood: $\nabla_{\theta} J_{\text{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \log \pi_{\theta}(\tau_i)$



Partial observability



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

Markov property is not actually used!

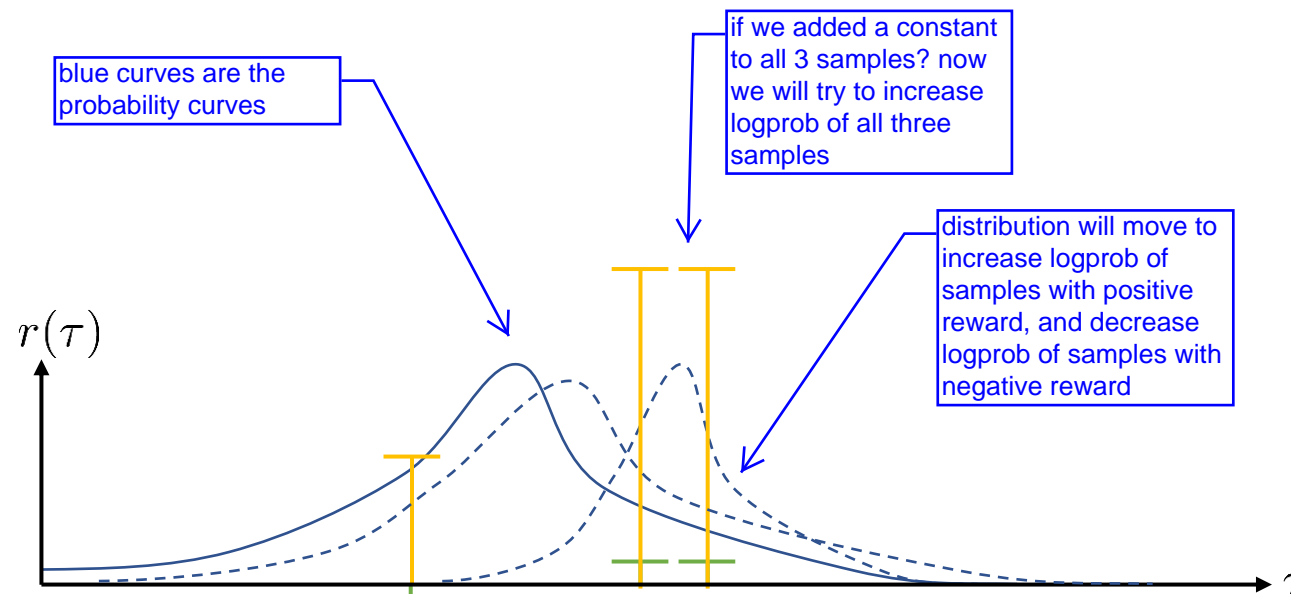
Markov property is not actually used when we derived the PG. This means that if we wanted to derive it for a partially observed system, we could do so and we'd get the same equation. Thus, we can use PG in partially observed MDP without modification!!

Can use policy gradient in partially observed MDPs without modification

What is wrong with the policy gradient?

PG so far won't work that well if you try to implement them.

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \log \pi_{\theta}(\tau) r(\tau)$$



even worse: what if the two “good” samples have $r(\tau) = 0$?

there's high variance! if we add a constant to each sample, the logprob distribution will change! This is bad. If we sampled infinitely, it wouldn't change anything, but we can't sample infinitely.

Thus, in practice, we need to lower the variance to get it to work. A lot of RL algorithms use different methods to decrease this variance.

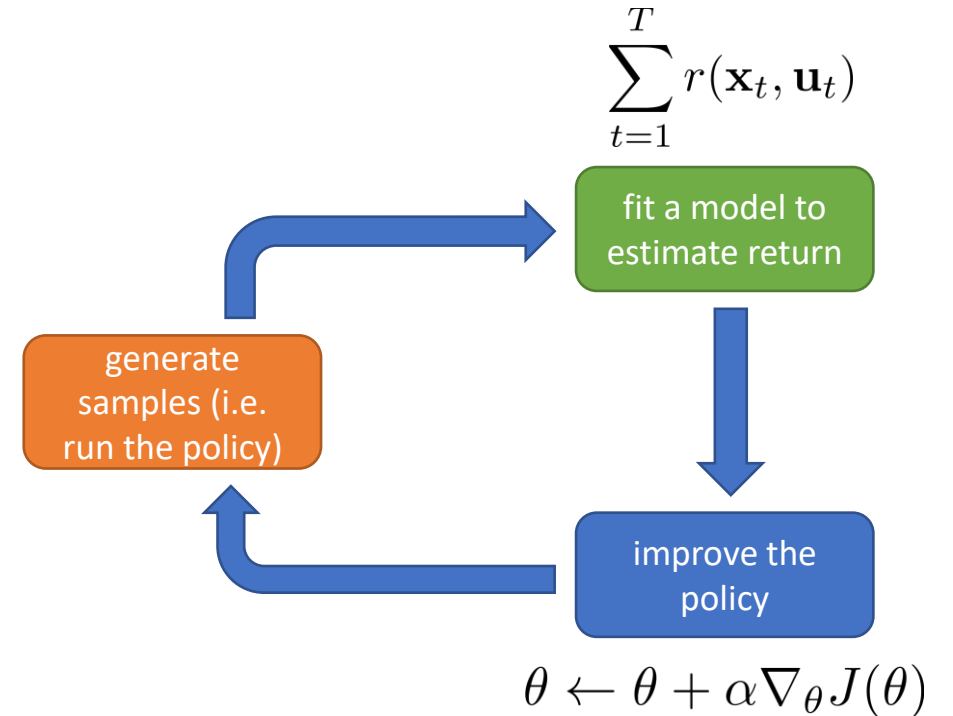
high variance

green bar is the reward

then their gradient wouldn't matter at all.

Review

- Evaluating the RL objective
 - Generate samples
- Evaluating the policy gradient
 - Log-gradient trick
 - Generate samples
- Understanding the policy gradient
 - Formalization of trial-and-error
- Partial observability
 - Works just fine
- What is wrong with policy gradient?



Reducing Variance

through reducing variance, we can
obtain a version of PG that can
actually be used as a practice RL
algorithm

Reducing variance

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

- "what you do now cannot affect the reward you got in the past"
- in other words, you can't do time travel

Causality: policy at time t' cannot affect reward at time t when $t < t'$

The PG algo we built so far does not actually make use of this assumption. We should modify it to make use of this assumption.

- note that this is NOT the same as the Markov property

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

let's change this to $t'=t$, so we know the current timestep can't affect rewards of the past!

by changing the sum of rewards from $t'=1$ to $t'=t$, we've reduce the variance because the sum is now smaller.

sum over all of our samples from $i=1$ to N , times the sum over time steps $t=1$ to T of the grad log pi, multiplied by the sum from timestep $t'=1$ to T of the rewards. That means that at every timestep, we multiply the gradlog pi by every timestep that's happened in the past, the current timestep and the future.

So we change the logprob of the action at every timestep based on whether that action corresponded to larger rewards in the present and future, and also the past...even though we know that it can't affect the rewards of the past!

In the expectation (i.e. if we took infinite samples), the rewards of the future would cancel out, but in practice, they don't!

"reward to go"

$$\hat{Q}_{i,t}$$

Baselines

Another trick to reduce variance and make PG more practical. Use a baseline.

before we said that PG will make the good trajectories more likely and the bad ones less likely. But is this actually what PG do? Intuitively, PG does this if the rewards are centered (meaning that the good traj have positive rewards and the bad traj have negative rewards)! But this might not necessarily be true! What if all your rewards are positive? Or negative? Thus, we want to center our rewards so the things better than average get increased, and the things worse than average get decreased.

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \log p_{\theta}(\tau) [r(\tau) - b]$$

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

but... are we *allowed* to do that??

so maybe we can center our rewards by setting b to be the "average reward"

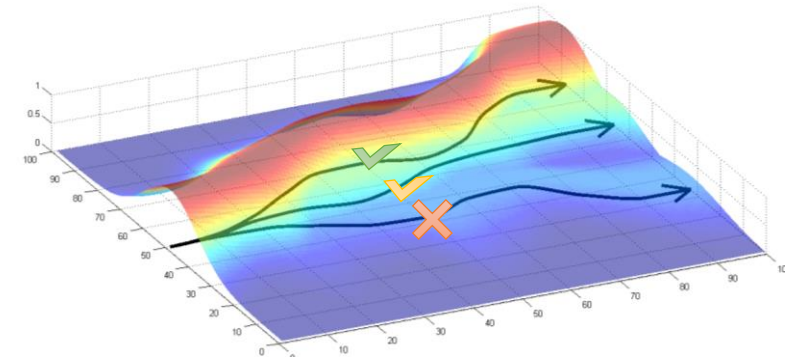
$$E[\nabla_{\theta} \log p_{\theta}(\tau) b] = \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) b d\tau = \int \nabla_{\theta} p_{\theta}(\tau) b d\tau = b \nabla_{\theta} \int p_{\theta}(\tau) d\tau = b \nabla_{\theta} 1 = 0$$

subtracting a baseline is **unbiased** in expectation!

average reward is *not* the best baseline, but it's pretty good!

a convenient identity

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



we can derive the optimal baseline, but in practice we don't really use it. Doing the average is good enough!

Analyzing variance

can we write down the variance?

$$\text{Var}[x] = E[x^2] - E[x]^2$$

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

$$\text{Var} = E_{\tau \sim p_{\theta}(\tau)} [(\nabla_{\theta} \log p_{\theta}(\tau) (r(\tau) - b))^2] - \underbrace{E_{\tau \sim p_{\theta}(\tau)} [\nabla_{\theta} \log p_{\theta}(\tau) (r(\tau) - b)]^2}_{\text{this bit is just } E_{\tau \sim p_{\theta}(\tau)} [\nabla_{\theta} \log p_{\theta}(\tau) r(\tau)]}$$

(baselines are unbiased in expectation)

$$\frac{d\text{Var}}{db} = \frac{d}{db} E[g(\tau)^2 (r(\tau) - b)^2] = \frac{d}{db} (E[\cancel{g(\tau)^2 r(\tau)^2}] - 2E[g(\tau)^2 r(\tau) b] + b^2 E[g(\tau)^2])$$

$$= -2E[g(\tau)^2 r(\tau)] + 2bE[g(\tau)^2] = 0$$

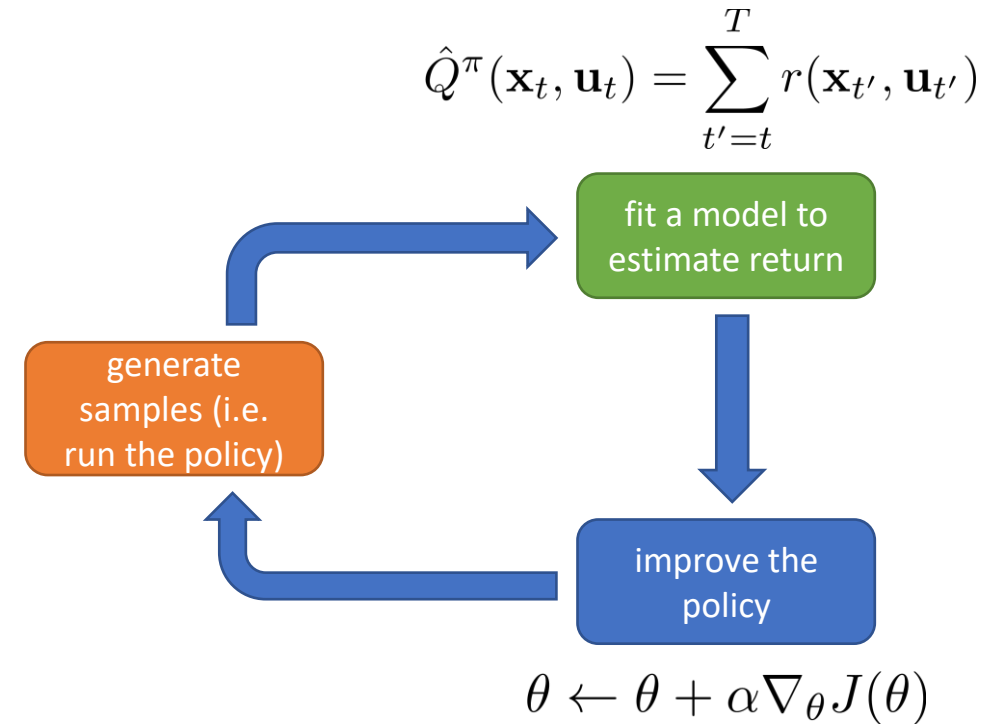
optimal value of b

$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]}$$

← This is just expected reward, but weighted by gradient magnitudes!

Review

- The high variance of policy gradient
- Exploiting causality
 - Future doesn't affect the past
- Baselines
 - Unbiased!
- Analyzing variance
 - Can derive optimal baselines



Off-Policy Policy Gradients

Policy gradient is on-policy

on-policy because we have to generate new samples every time we change the policy.

$$\theta^* = \arg \max_{\theta} J(\theta)$$

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

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

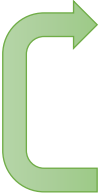
we must sample trajectories from the current policy

- Neural networks change only a little bit with each gradient step
- On-policy learning can be extremely inefficient!

so we have to make a large number of small gradient updates

can't just skip this!

REINFORCE algorithm:

- 
1. sample $\{\tau^i\}$ from $\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$ (run it on the robot)
 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. $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$

Off-policy learning & importance sampling

$$\theta^* = \arg \max_{\theta} J(\theta)$$

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

what if we don't have samples from $p_{\theta}(\tau)$?
(we have samples from some $\bar{p}(\tau)$ instead)

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

the only difference is the policy because they operate in the same MDP

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

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

for example, from a previous distribution

importance sampling

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

distribution $p(x)$ that we want

dist $q(x)$ is what we have

Deriving the policy gradient with IS

$$\theta^* = \arg \max_{\theta} J(\theta)$$

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

a convenient identity

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

can we estimate the value of some *new* parameters θ' ?

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

the only bit that depends on θ'

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim p_{\theta}(\tau)} \left[\frac{\nabla_{\theta'} p_{\theta'}(\tau)}{p_{\theta}(\tau)} r(\tau) \right] = E_{\tau \sim p_{\theta}(\tau)} \left[\frac{\cancel{p_{\theta'}(\tau)}}{\cancel{p_{\theta}(\tau)}} \nabla_{\theta'} \log p_{\theta'}(\tau) r(\tau) \right]$$

now estimate locally, at $\theta = \theta'$: $\nabla_{\theta} J(\theta) = E_{\tau \sim p_{\theta}(\tau)}[\nabla_{\theta} \log p_{\theta}(\tau) r(\tau)]$

The off-policy policy gradient

$$\theta^* = \arg \max_{\theta} J(\theta) \quad J(\theta) = E_{\tau \sim p_{\theta}(\tau)}[r(\tau)]$$

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim p_{\theta}(\tau)} \left[\frac{p_{\theta'}(\tau)}{p_{\theta}(\tau)} \nabla_{\theta'} \log \pi_{\theta'}(\tau) r(\tau) \right] \quad \text{when } \theta \neq \theta'$$

$$\frac{p_{\theta'}(\tau)}{p_{\theta}(\tau)} = \frac{\prod_{t=1}^T \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\prod_{t=1}^T \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)}$$

$$= E_{\tau \sim p_{\theta}(\tau)} \left[\left(\prod_{t=1}^T \frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)} \right) \left(\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right] \text{ what about causality?}$$

$$= E_{\tau \sim p_{\theta}(\tau)} \left[\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \underbrace{\left(\prod_{t'=1}^t \frac{\pi_{\theta'}(\mathbf{a}_{t'} | \mathbf{s}_{t'})}{\pi_{\theta}(\mathbf{a}_{t'} | \mathbf{s}_{t'})} \right)}_{\text{future actions don't affect current weight}} \left(\sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \left(\prod_{t''=t}^{t'} \frac{\pi_{\theta'}(\mathbf{a}_{t''} | \mathbf{s}_{t''})}{\pi_{\theta}(\mathbf{a}_{t''} | \mathbf{s}_{t''})} \right) \right) \right]$$

future actions don't affect current weight

if we ignore this, we get
a policy iteration algorithm
(more on this in a later lecture)

A first-order approximation for IS (preview)

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim p_{\theta}(\tau)} \left[\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \underbrace{\left(\prod_{t'=1}^t \frac{\pi_{\theta'}(\mathbf{a}_{t'} | \mathbf{s}_{t'})}{\pi_{\theta}(\mathbf{a}_{t'} | \mathbf{s}_{t'})} \right)}_{\text{exponential in } T} \left(\sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \right]$$

let's write the objective a bit differently...

on-policy policy gradient: $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$

"reward to go"

$(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \sim \pi_{\theta}(\mathbf{s}_t, \mathbf{a}_t)$

off-policy policy gradient: $\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \frac{\pi_{\theta'}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}{\pi_{\theta}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$

We'll see why this is reasonable later in the course!

$$= \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \frac{\cancel{\pi_{\theta'}(\mathbf{s}_{i,t})}}{\cancel{\pi_{\theta}(\mathbf{s}_{i,t})}} \frac{\pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

ignore this part

Implementing Policy Gradients

Policy gradient with automatic differentiation

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

It's hard to compute the policy gradient directly

If we wanted to calculate the PG naively, we could calculate grad log pi for every (s, a) tuple that we sampled. This is very inefficient, b/c NN can have millions of params!

pretty inefficient to compute these explicitly!

How can we compute policy gradients with automatic differentiation?

We need a graph such that its gradient is the policy gradient!

We need to set up a NN such that the derivative of that graph is the policy gradient!

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

if we wanted to compute MLE gradients, we would implement the MLE objective (using the cross entropy loss), then call `.backward()` or `.gradients()` to calculate the gradients

Just implement “pseudo-loss” as a weighted maximum likelihood:

$$\tilde{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

Note that this equation is NOT the RL objective! In fact this equation is not anything. It's just a quantity chosen such that its derivatives come out to be the policy gradient. Thus, we "trick" the autodiff package into giving us the gradient that we want

cross entropy (discrete) or squared error (Gaussian)

J is our objective function (i.e. our Loss function) !!

Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

N: number of samples
T: number of timesteps
Da: number of action dimensions
Ds: number of state dimensions

Maximum likelihood:

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
loss = tf.reduce_mean(negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

This is $J(\theta)$ from
the previous slide

Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

Policy gradient:

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# q_values - (N*T) x 1 tensor of estimated state-action values
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, q_values)
loss = tf.reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

So here, we've tricked our autodiff package into calculating the policy gradient

this is the Policy gradient

This is $J(\theta)$ from the previous slide

$$\tilde{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

q_values

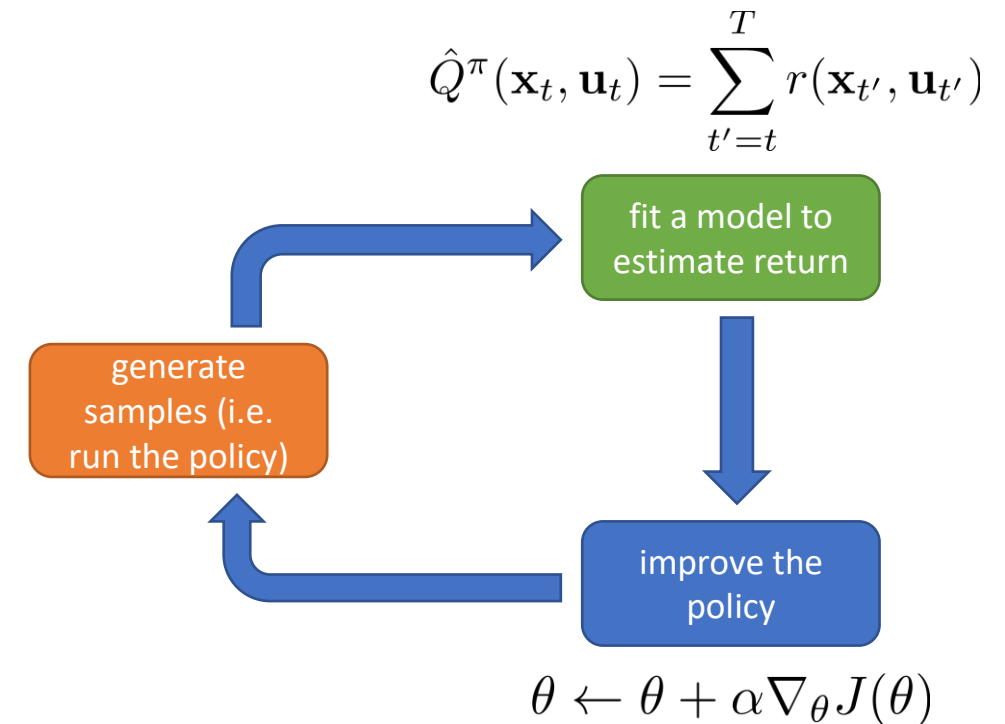
Policy gradient in practice

- Remember that the gradient has high variance
 - This isn't the same as supervised learning!
 - Gradients will be really noisy!
- Consider using much larger batches batch sizes in the thousands or tens of thousands is typical
- Tweaking learning rates is very hard
 - Adaptive step size rules like ADAM can be OK-ish typical SGD with momentum can be hard to use. Using ADAM is a good starting point
 - We'll learn about policy gradient-specific learning rate adjustment methods later!

In general, we will have to do more hyperparameter tuning than we do for supervised learning

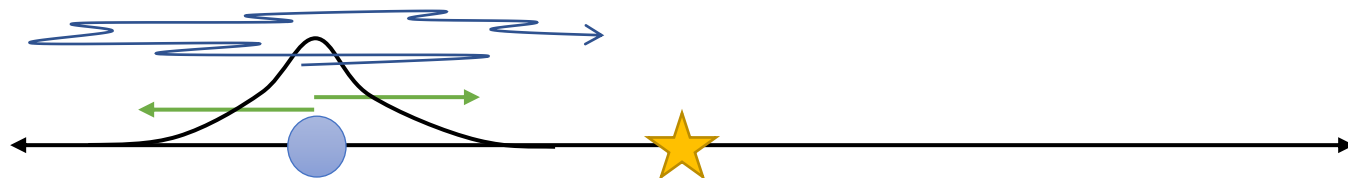
Review

- Policy gradient is on-policy
- Can derive off-policy variant
 - Use importance sampling
 - Exponential scaling in T
 - Can ignore state portion (approximation)
- Can implement with automatic differentiation – need to know what to backpropagate
- Practical considerations: batch size, learning rates, optimizers



Advanced Policy Gradients

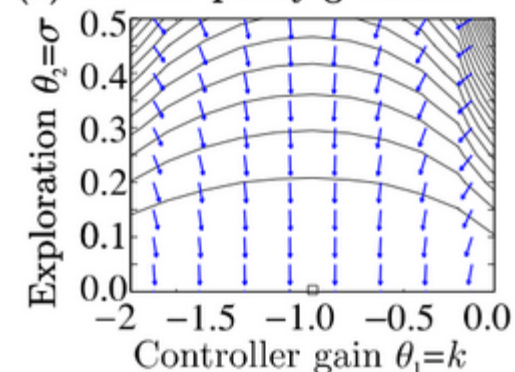
What *else* is wrong with the policy gradient?



$$r(s_t, a_t) = -s_t^2 - a_t^2$$

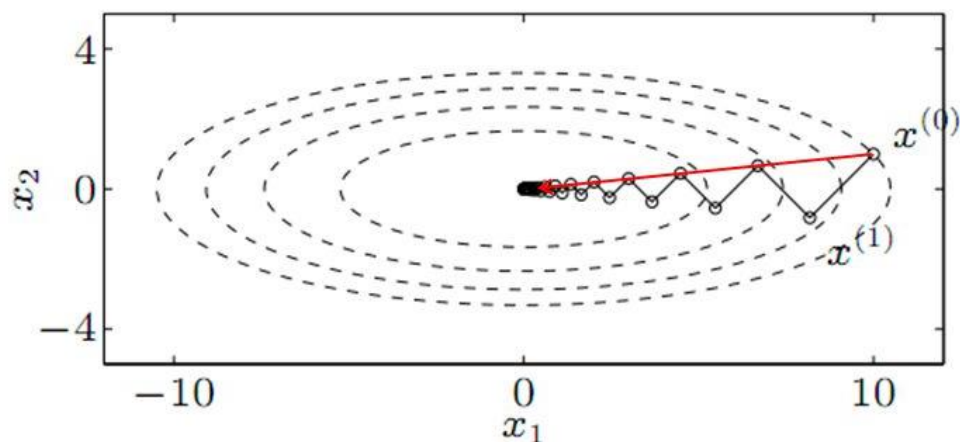
$$\log \pi_\theta(a_t|s_t) = -\frac{1}{2\sigma^2}(ks_t - a_t)^2 + \text{const} \quad \theta = (k, \sigma)$$

(a) 'Vanilla' policy gradients



(image from Peters & Schaal 2008)

Essentially the same problem as this:



Covariant/natural policy gradient

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta) \quad \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$$

some parameters change probabilities a lot more than others!

$$\theta' \leftarrow \arg \max_{\theta'} (\theta' - \theta)^T \nabla_{\theta} J(\theta) \text{ s.t. } \|\theta' - \theta\|^2 \leq \epsilon$$

controls how far we go

can we *rescale* the gradient so this doesn't happen?

$$\theta' \leftarrow \arg \max_{\theta'} (\theta' - \theta)^T \nabla_{\theta} J(\theta) \text{ s.t. } D(\pi_{\theta'}, \pi_{\theta}) \leq \epsilon$$

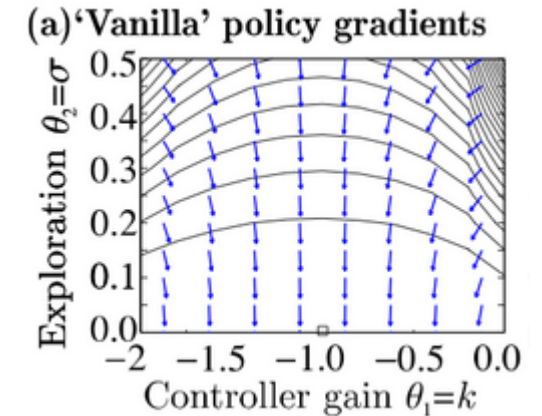
parameterization-independent divergence measure

usually KL-divergence: $D_{\text{KL}}(\pi_{\theta'} || \pi_{\theta}) = E_{\pi_{\theta}} [\log \pi_{\theta} - \log \pi_{\theta'}]$

$$D_{\text{KL}}(\pi_{\theta'} || \pi_{\theta}) \approx (\theta' - \theta)^T \mathbf{F} (\theta' - \theta) \quad \mathbf{F} = E_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(\mathbf{a} | \mathbf{s}) \nabla_{\theta} \log \pi_{\theta}(\mathbf{a} | \mathbf{s})^T]$$

Fisher-information matrix

can estimate with samples



Covariant/natural policy gradient

$$D_{\text{KL}}(\pi_{\theta'} \parallel \pi_{\theta}) \approx (\theta' - \theta)^T \mathbf{F} (\theta' - \theta) \quad \mathbf{F} = E_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(\mathbf{a}|\mathbf{s}) \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}|\mathbf{s})^T]$$

$$\theta' \leftarrow \arg \max_{\theta'} (\theta' - \theta)^T \nabla_{\theta} J(\theta) \text{ s.t. } \|\theta' - \theta\|_{\mathbf{F}}^2 \leq \epsilon$$

$$\theta \leftarrow \theta + \alpha \mathbf{F}^{-1} \nabla_{\theta} J(\theta)$$

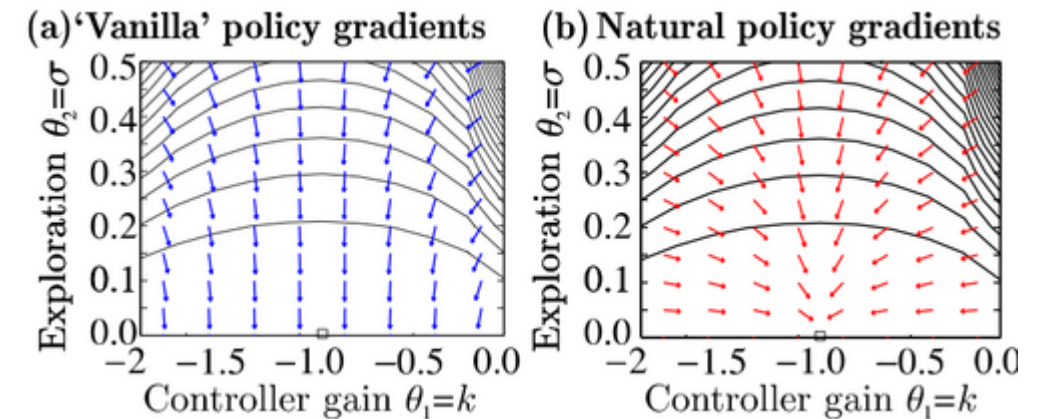
natural gradient: pick α

trust region policy optimization: pick ϵ

can solve for optimal α while solving $\mathbf{F}^{-1} \nabla_{\theta} J(\theta)$

conjugate gradient works well for this

see Schulman, L., Moritz, Jordan, Abbeel (2015) Trust region policy optimization



(figure from Peters & Schaal 2008)

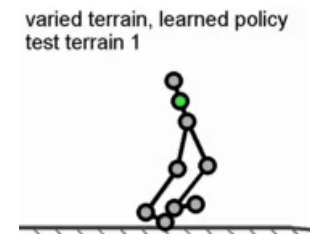
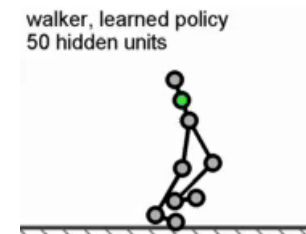
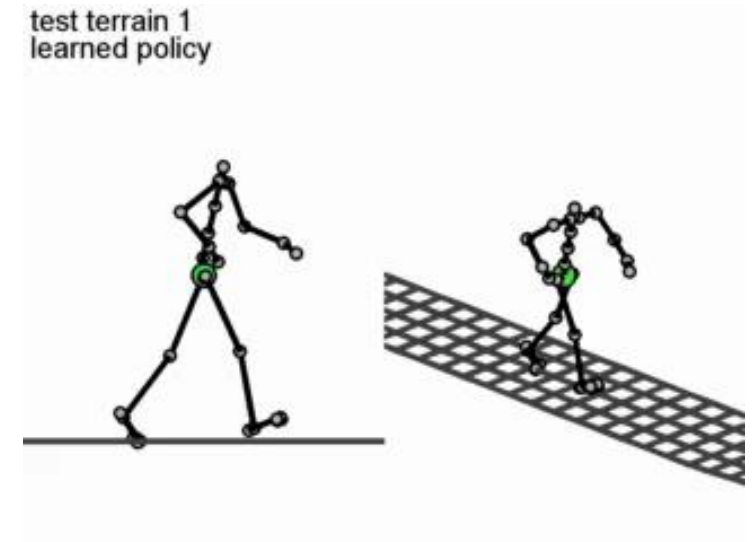
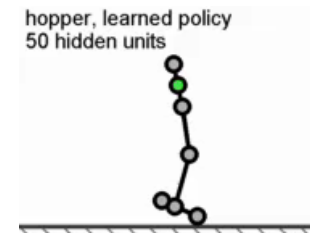
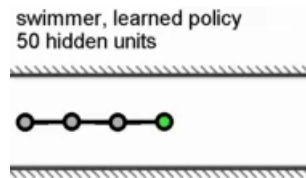
Advanced policy gradient topics

- What more is there?
- Next time: introduce value functions and Q-functions
- Later in the class: more on natural gradient and automatic step size adjustment

Example: policy gradient with importance sampling

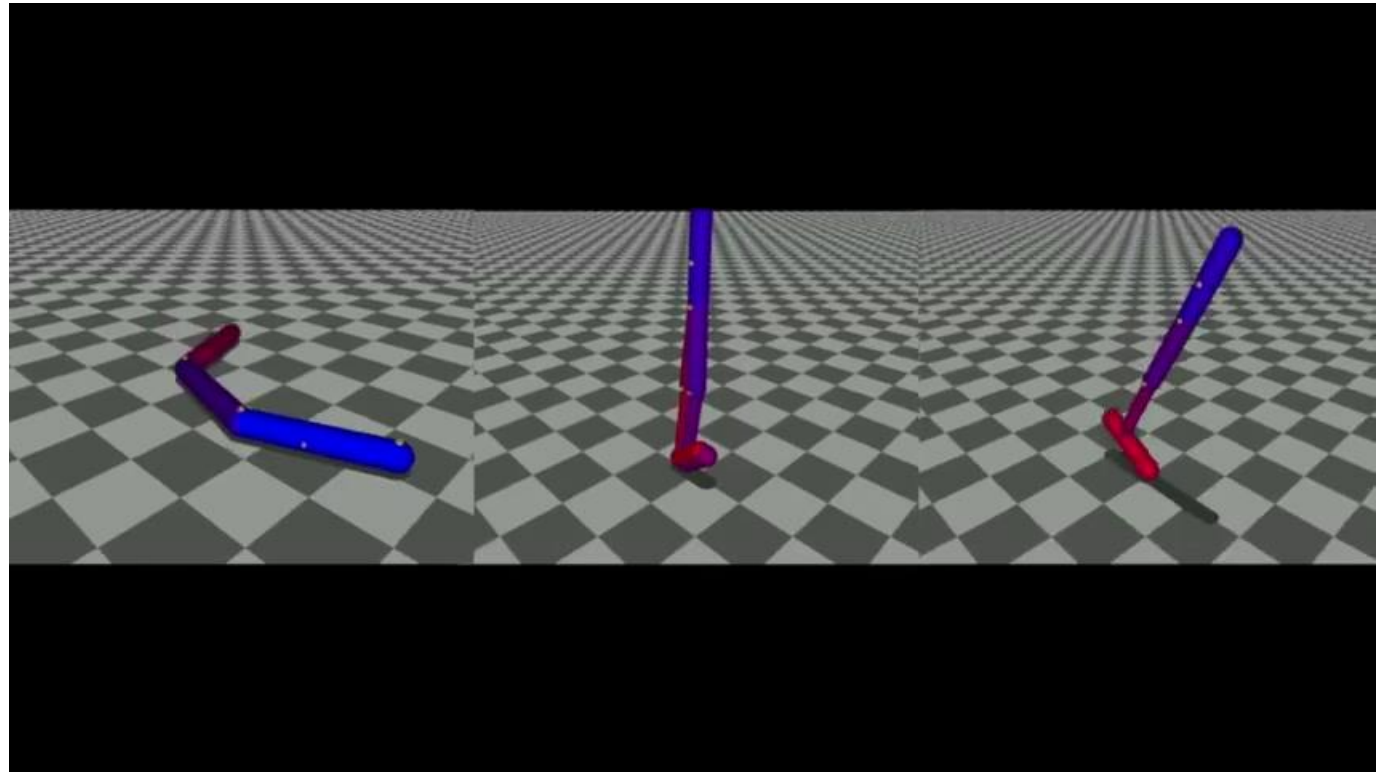
$$\nabla_{\theta'} J(\theta') = E_{\tau \sim \pi_{\theta}(\tau)} \left[\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \left(\prod_{t'=1}^t \frac{\pi_{\theta'}(\mathbf{a}_{t'} | \mathbf{s}_{t'})}{\pi_{\theta}(\mathbf{a}_{t'} | \mathbf{s}_{t'})} \right) \left(\sum_{t'=t}^T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

- Incorporate example demonstrations using importance sampling
- Neural network policies



Example: trust region policy optimization

- Natural gradient with automatic step adjustment
- Discrete and continuous actions
- Code available (see Duan et al. '16)



Policy gradients suggested readings

- Classic papers
 - Williams (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning: introduces REINFORCE algorithm
 - Baxter & Bartlett (2001). Infinite-horizon policy-gradient estimation: temporally decomposed policy gradient (not the first paper on this! see actor-critic section later)
 - Peters & Schaal (2008). Reinforcement learning of motor skills with policy gradients: very accessible overview of optimal baselines and natural gradient
- Deep reinforcement learning policy gradient papers
 - Levine & Koltun (2013). Guided policy search: deep RL with importance sampled policy gradient (unrelated to later discussion of guided policy search)
 - Schulman, L., Moritz, Jordan, Abbeel (2015). Trust region policy optimization: deep RL with natural policy gradient and adaptive step size
 - Schulman, Wolski, Dhariwal, Radford, Klimov (2017). Proximal policy optimization algorithms: deep RL with importance sampled policy gradient