Robotics
Spring 2023

Abhishek Gupta

TAs: Yi Li, Srivatsa GS

Recap: Course Overview

- Filtering/Smoothing
- Localization
- Mapping
- SLAM

- Search
- Motion Planning
- TrajOpt
- Stability/Certification

- MDPs and RL
- Imitation Learning
- Off-Policy/MBRL

Lecture Outline

Recap + Policy Gradient

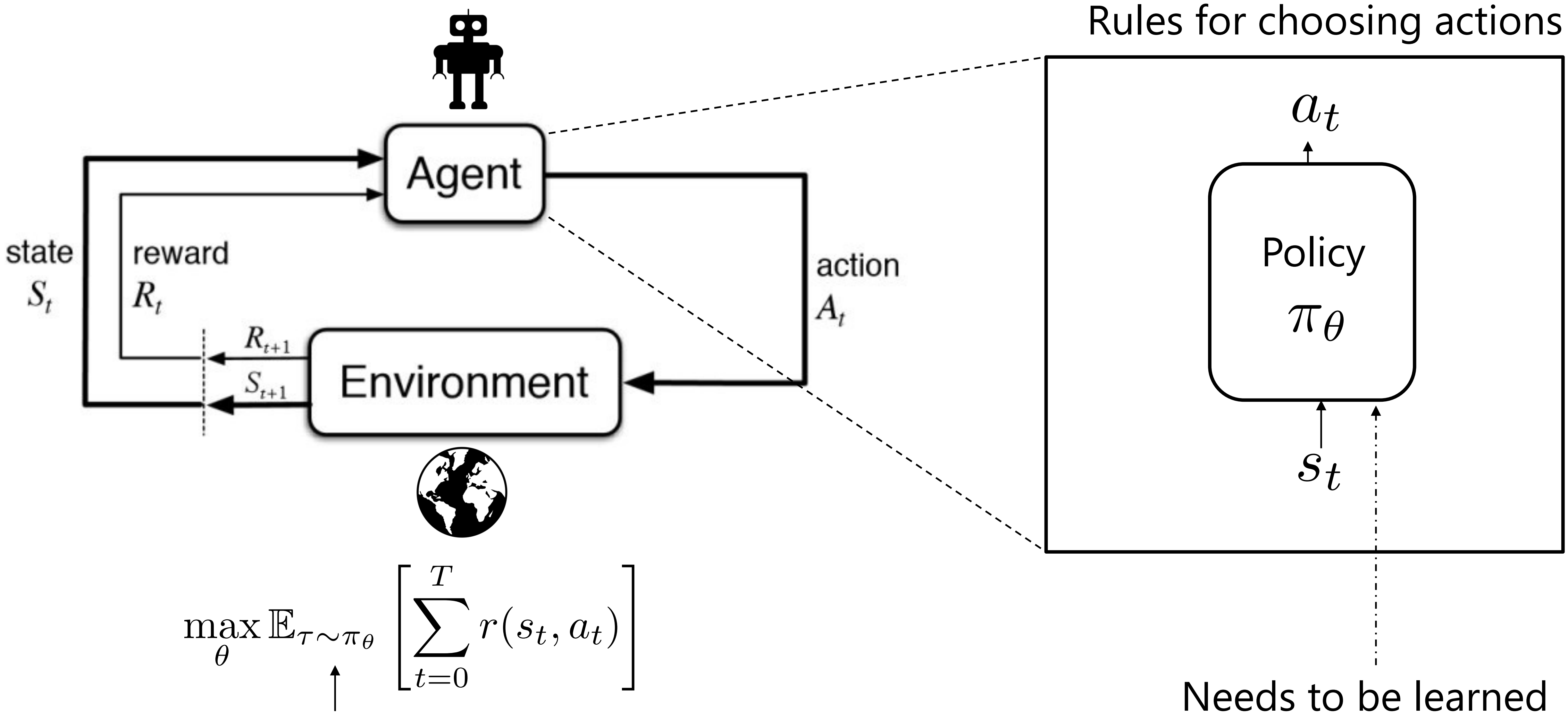
↓

Basic Actor Critic Algorithms

↓

Making Actor-Critic Practical

Reinforcement Learning Formalism

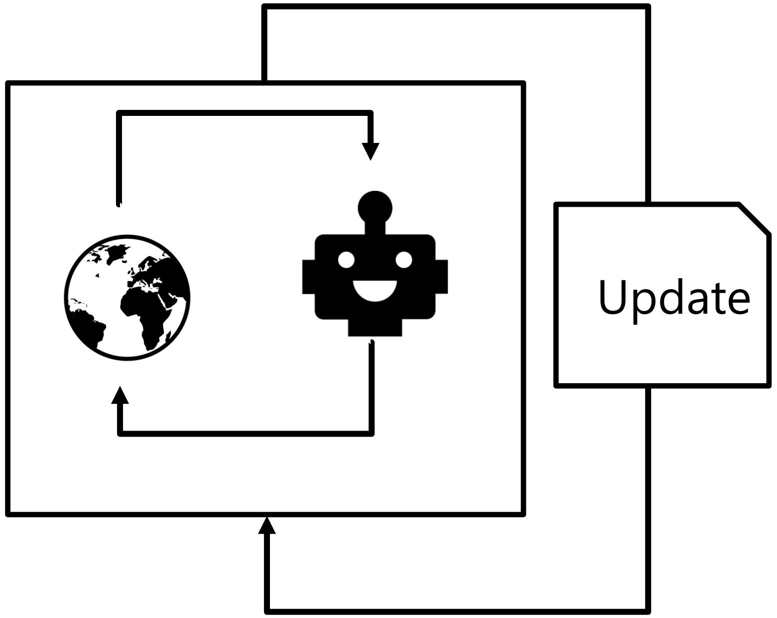


$$\max_\theta \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} r(s_t, a_t) \right]$$

Trajectory sampled using policy

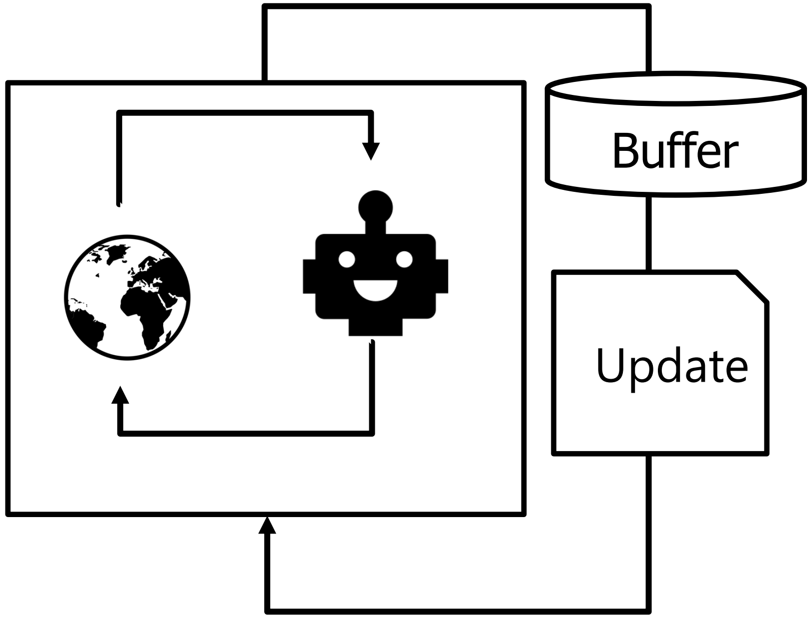
Learning Algorithms for Robotics

On-Policy Algorithms



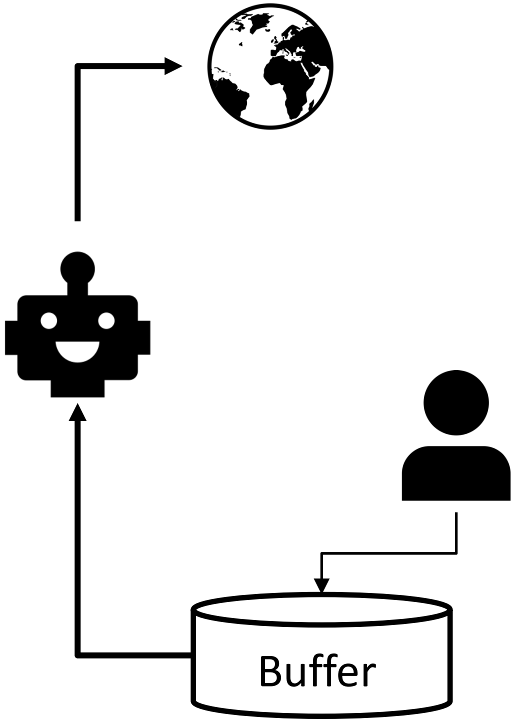
Simple, performant,
Data inefficient

Off-Policy Algorithms



Data-efficient,
sometimes unstable

Imitation Learning



Performant, efficient, but
compounding error and
expensive data collection

What if we just performed gradient ascent?

$$\max_\theta \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} r(s_t, a_t)\right]$$

$$= \int p_\theta(\tau) R(\tau) d\tau$$

Standard gradient descent (supervised learning)

$$\nabla_\theta \mathbb{E}_{x \sim g(x)}\left[f_\theta(x)\right]$$

REINFORCE gradient descent (RL)

$$\nabla_\theta \mathbb{E}_{x \sim p_\theta(x)}\left[f(x)\right]$$

Gradient wrt expectation variable, not of integrand!

Taking the gradient of sum of rewards

$$J(\theta) = \int p_\theta(\tau) R(\tau) d(\tau)$$

$$\nabla_\theta J(\theta) = \nabla_\theta \int p_\theta(\tau) R(\tau) d(\tau)$$

$$= \int \nabla_\theta p_\theta(\tau) R(\tau) d(\tau) \quad = \int \frac{p_\theta(\tau)}{p_\theta(\tau)} \nabla_\theta p_\theta(\tau) R(\tau) d(\tau)$$

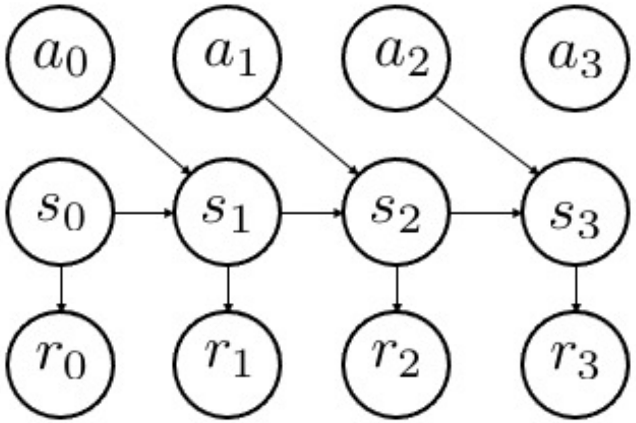
$$= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) R(\tau) d(\tau) \ = \mathbb{E}_{p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) R(\tau)\right]$$

REINFORCE trick

Taking the gradient of return

Initial State   Dynamics   Policy

$$p_\theta(\tau) = p(s_0)\Pi_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$$



$$\log p_\theta(\tau) = \log p(s_0) + \sum_{t=0}^{T-1} \log p(s_{t+1}|s_t, a_t) + \log \pi(a_t|s_t)$$

$$\nabla_\theta \log p_\theta(\tau) = \nabla_\theta \log p(s_0) + \sum_{t=0}^{T-1} \nabla_\theta \log p(s_{t+1}|s_t, a_t) + \nabla_\theta \log \pi(a_t|s_t)$$

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t)$$

Model Free!!

Taking the gradient of return

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) \sum_{t=0}^{T} r(s_t, a_t) \right]$$

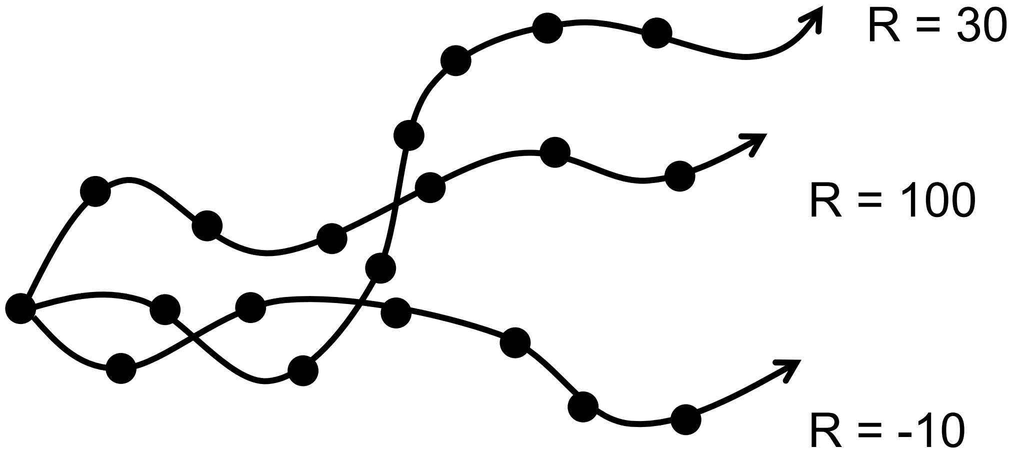
$$\nabla_\theta J(\theta) = \mathbb{E}_{\substack{s_0 \sim p(s_0) \\ s_{t+1} \sim p(s_{t+1}|s_t, a_t) \\ a_t \sim \pi(a_t|s_t)}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=0}^{T} r(s_t, a_t) \right]$$

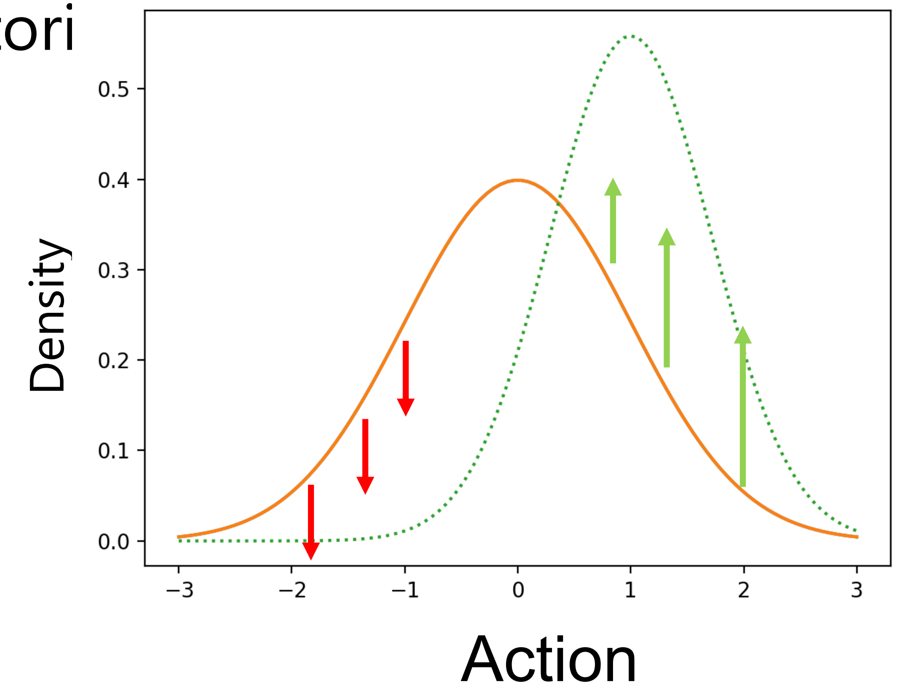
$$\approx \frac{1}{N} \sum_{i=0}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \sum_{t'=0}^{T} r(s_{t'}^i, a_{t'}^i)$$ (approximating using samples)

What does this mean?

$$\nabla_\theta J(\theta) = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) d\tau \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \sum_{t'=0}^{T} r(s_{t'}^i, a_{t'}^i)$$

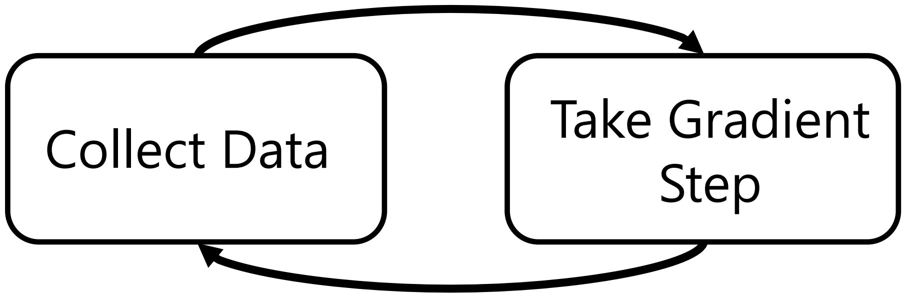
Increase the likelihood of actions in high return trajectori





Resulting Algorithm (REINFORCE)

$$\nabla_\theta J(\theta) = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) d\tau$$



REINFORCE algorithm:

On-policy →

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

REINFORCE Pseudocode

Algorithm 1 Vanilla Policy Gradient Algorithm

```
Input: initial policy parameters θ_0, initial value function parameters φ_0
for k = 0, 1, 2, ... do
    Collect set of trajectories D_k = {τ_i} by running policy π_k = π(θ_k) in the environment.
    Compute rewards-to-go R̂_t.
    Compute advantage estimates, Â_t (using any method of advantage estimation) based
    on the current value function V_{φ_k}.
    Estimate policy gradient as
```

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

```
    Compute policy update, either using standard gradient ascent,
```

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

```
    or via another gradient ascent algorithm like Adam.
    Fit value function by regression on mean-squared error:
```

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

```
    typically via some gradient descent algorithm.
end for
```

→ Sum up reward to go

→ Simply subtract baseline

→ Regression to learn baseline

How is this related to supervised learning?

<u>Reinforcement Learning</u>

$$\nabla_\theta J(\theta) = \int p_\theta(\tau)\nabla_\theta \log p_\theta(\tau)d\tau$$

$$\approx \frac{1}{N}\sum_{i=0}^{N}\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\sum_{t'=0}^{T} r(s_{t'}^i, a_{t'}^i)$$

<u>Supervised Learning</u>

$$\max_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\log p_\theta(y|x)\right]$$

$$\approx \frac{1}{N}\sum_i \nabla_\theta \log p_\theta(y^i|x^i)$$

PG = select good data + increase likelihood of selected data

What makes policy gradient challenging?

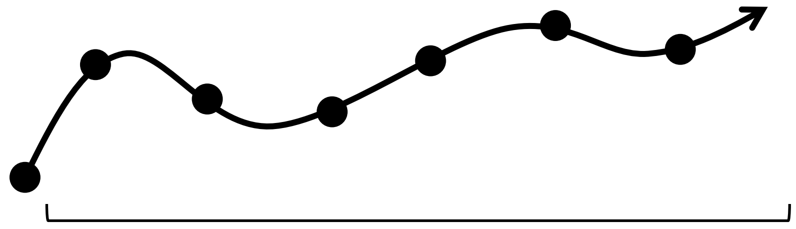
$$\nabla_\theta J(\theta) = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) d\tau$$

$$\approx \frac{1}{N} \sum_{i=0}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \sum_{t'=0}^{T} r(s_{t'}^i, a_{t'}^i)$$

High variance estimator!!

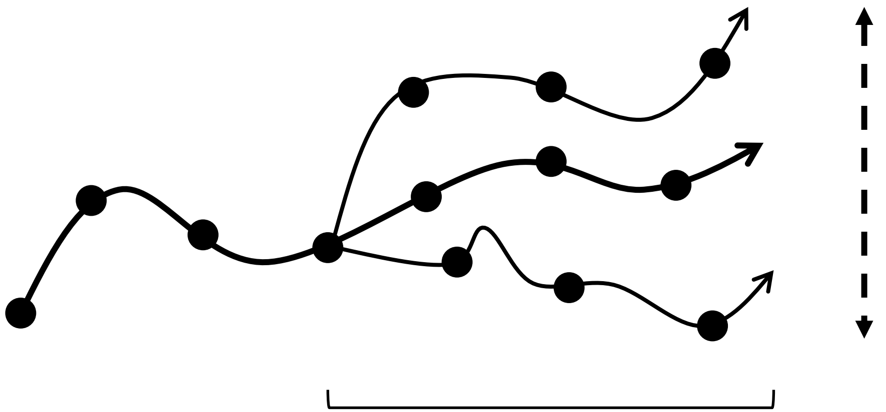
Hard to tell what matters without many samples

What we do



Single sample estimate

What we actually want

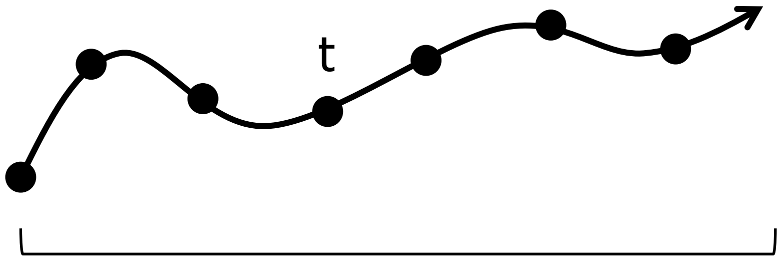


Averaged return estimate

Variance Reduction with Causality

Idea: Trajectory returns depend on past and future, but we only care about the future, since actions cannot affect the past. Instead, consider **"return-to-go"**

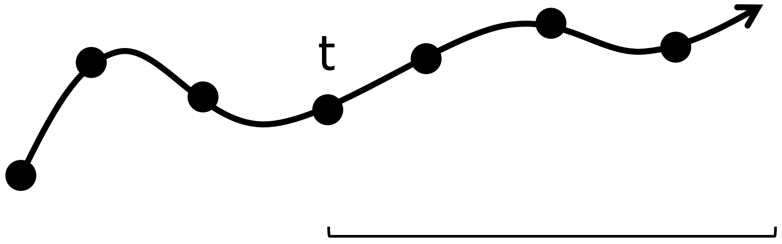
$$\approx \frac{1}{N}\sum_{i=0}^{N}\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\underbrace{\sum_{t'=0}^{T} r(s_{t'}^i, a_{t'}^i)}_{\text{Includes } t' < t}$$



Full trajectory return

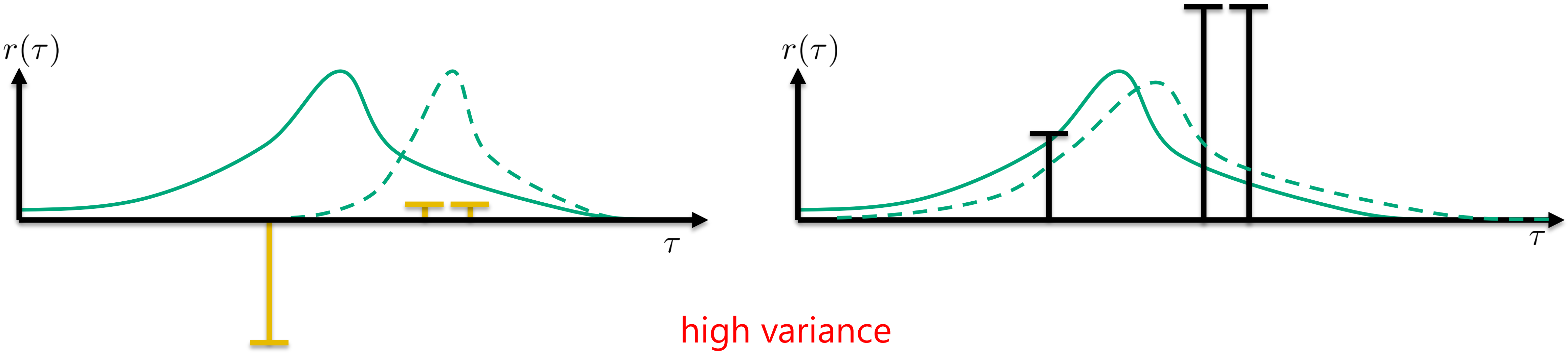
Ignore past terms

$$\frac{1}{N}\sum_{i=0}^{N}\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\sum_{t'=t}^{T} r(s_t^i, a_t^i)$$



Return to go

Can we reduce variance further?



high variance

Arbitrarily uncentered, scaled returns can lead to huge variance:

→ Imagine all rewards were positive, every action would be pushed up, some more than others

→ What if instead, we pushed down some actions and pushed up some others (even if rewards are posi

Credit: Sergey Levine

Variance Reduction with a Baseline

Idea: We can reduce variance by subtracting a current state dependent function from the policy gradient return

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\left[\sum_{t'=t}^{T} r(s_{t'}^i, a_{t'}^i) - b(s_t)\right]$$

Baseline: Centers the returns, reduces variance

But does this increase bias??

Variance Reduction with a Baseline

$$\int_{\mathcal{S}} \int_{\mathcal{A}} p(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t) \left[\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) - b(s_t) \right] \, ds_t \, da_t$$

$$\int_{\mathcal{S}} \int_{\mathcal{A}} p(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t) \left[\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \right] \, ds_t \, da_t - \int_{\mathcal{S}} \int_{\mathcal{A}} p(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t) \, ds_t \, da_t$$

Let us show this is 0!

Variance Reduction with a Baseline

$$\int\int p(s_t, a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)\left[b(s_t)\right] ds_t da_t = \int\int p(s_t)\pi_\theta(a_t|s_t)\nabla_\theta \log \pi_\theta(a_t|s_t)\left[b(s_t)\right] ds_t da_t$$

$$= \int p(s_t)b(s_t) \int \pi_\theta(a_t|s_t)\nabla_\theta \log \pi_\theta(a_t|s_t) da_t ds_t$$

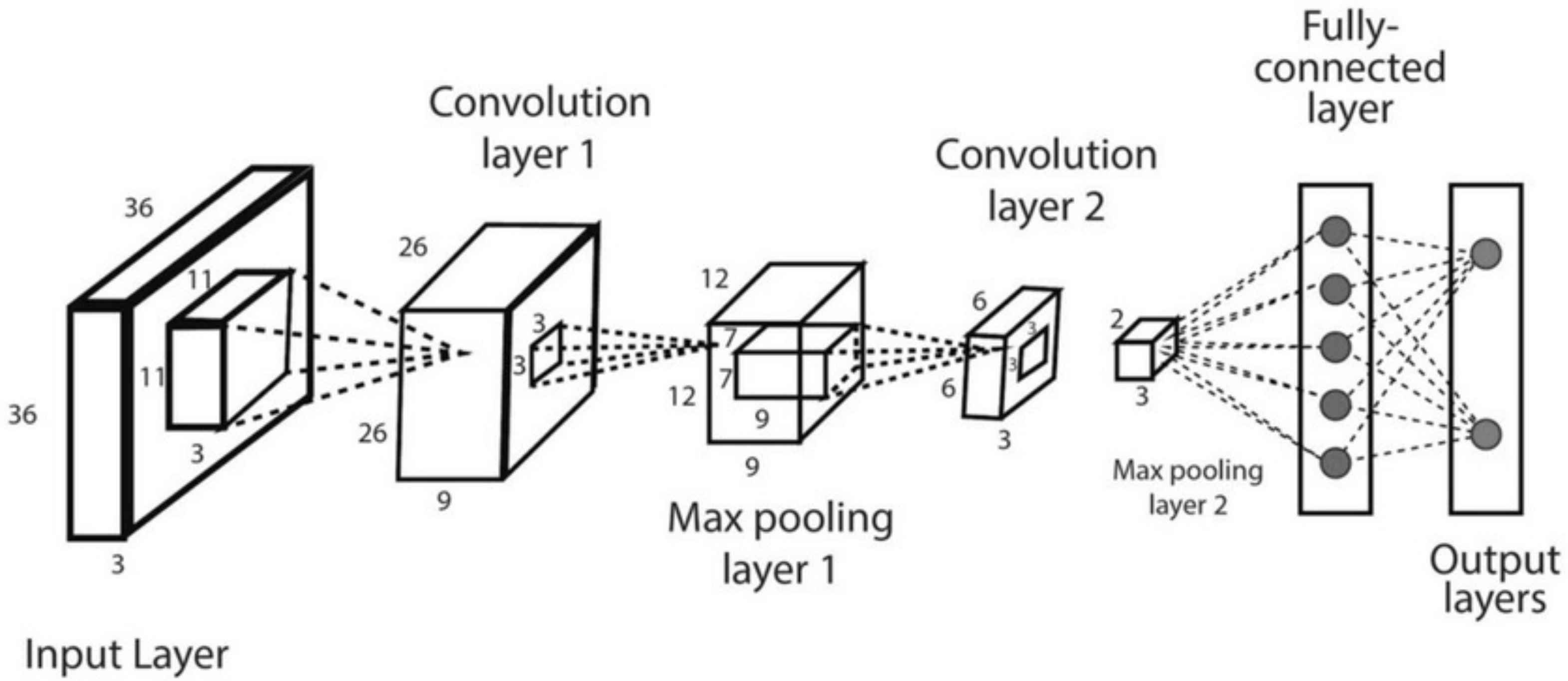
$$= \int p(s_t)b(s_t) \int \nabla_\theta \pi_\theta(a_t|s_t) da_t ds_t$$

$$= \int p(s_t)b(s_t)\nabla_\theta \int \pi_\theta(a_t|s_t) da_t ds_t = \int p(s_t)b(s_t)\nabla_\theta(1) ds_t = 0$$

Unbiased!

Learning Baselines

Baselines are typically learned as deep neural nets from $R^s \rightarrow R^1$



$$\frac{1}{N}\sum_{j=1}^{N}\|\hat{V}(s_t^j, a_t^j) - \sum_{t=1}^{H} r(s_t^j, a_t^j)\|$$

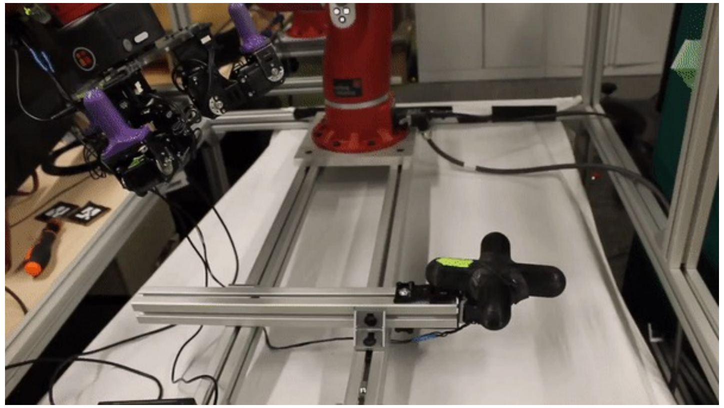
Minimize with Monte-carlo regression at every iteration, club with policy loss

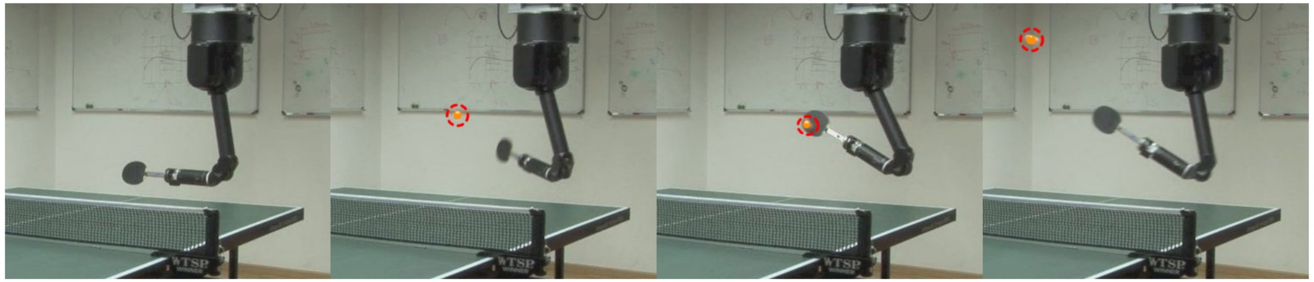
$$A(s_t, a_t) = \sum_{t'=t}^{T} r(s'_t, a'_t) - V(s_t)$$

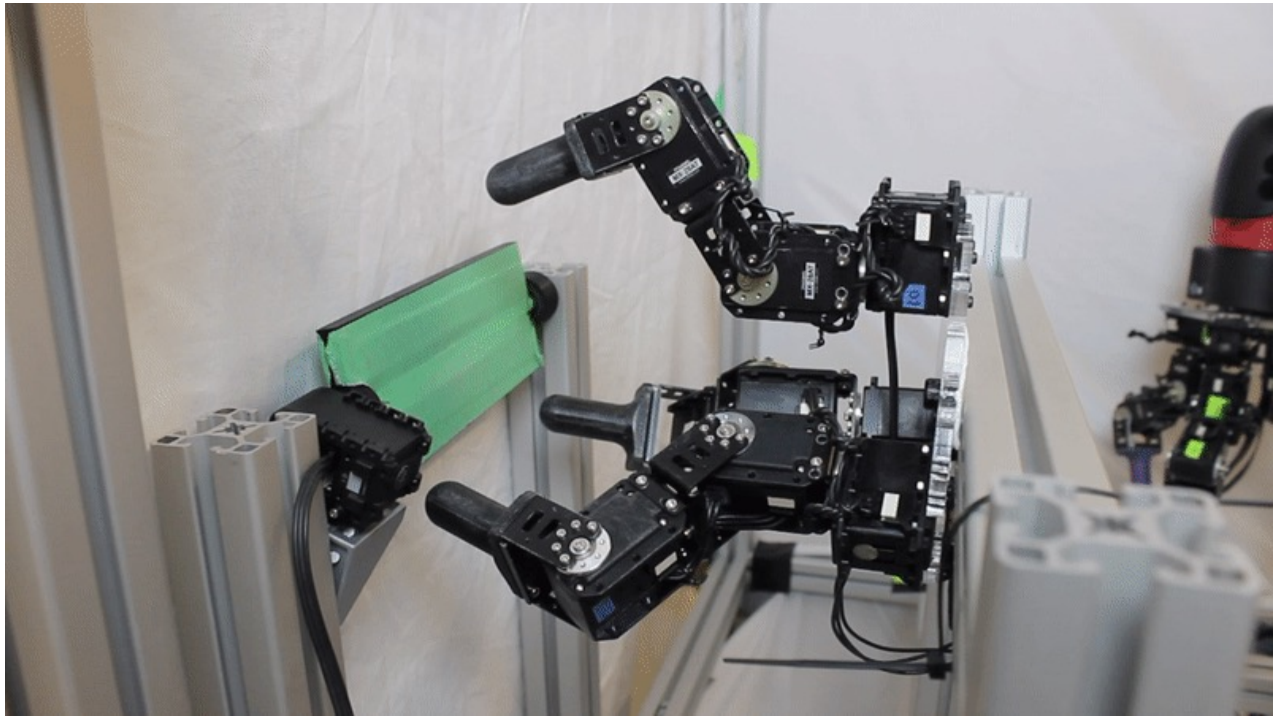
Allows us to define advantages

How is this useful for robotics?

Can be used to train robots in the real world but only in limited settings

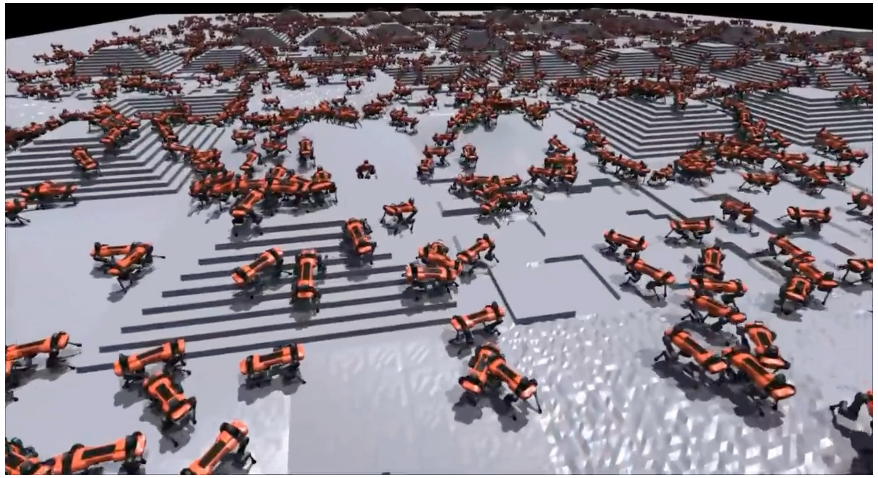


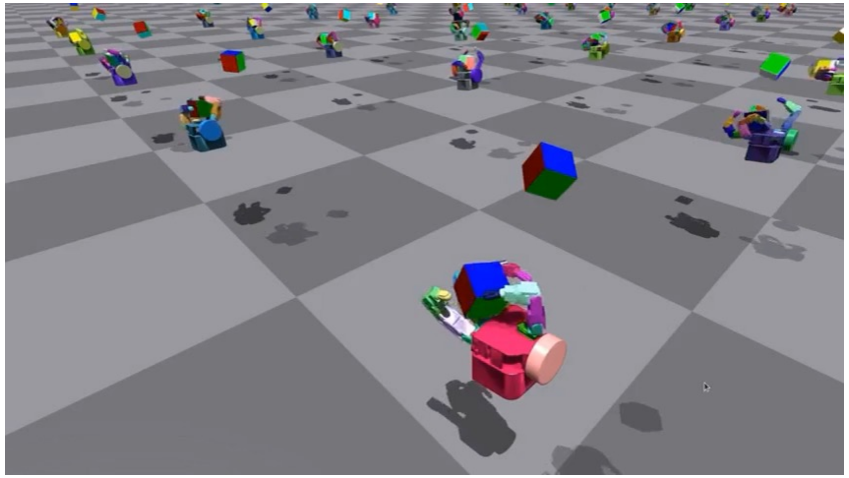


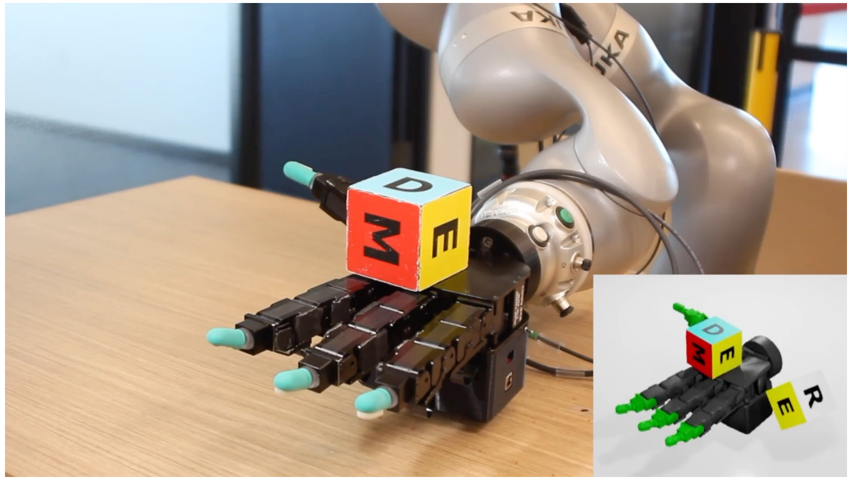


How is this useful for robotics?

Largely useful for pretraining in simulation







More in the sim2real lecture!

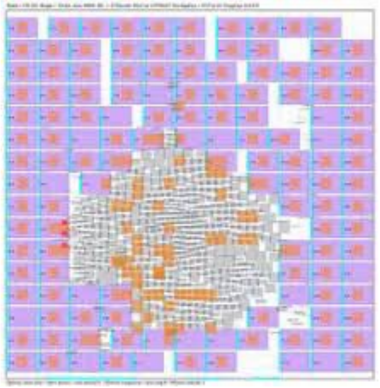
Pros/Cons of Policy Gradient Methods

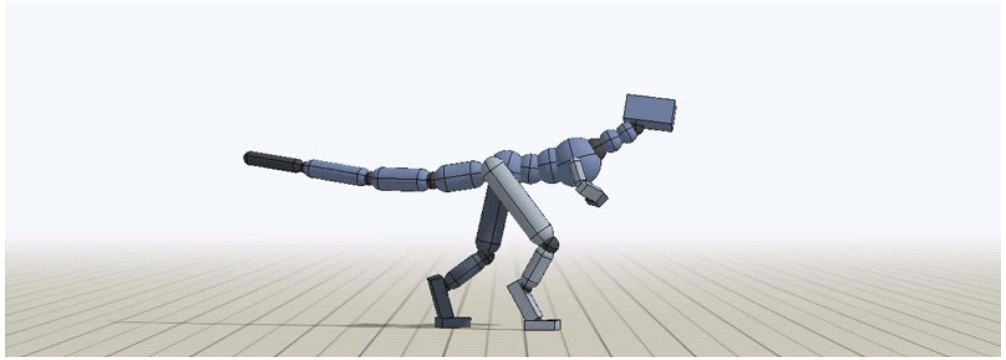
<u>Pros</u>

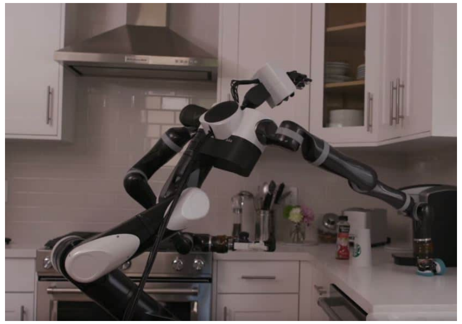
- Conceptually simple, easy to implement
- Stable, good asymptotic performance
- Compatible with deep models
- Require minimal modeling

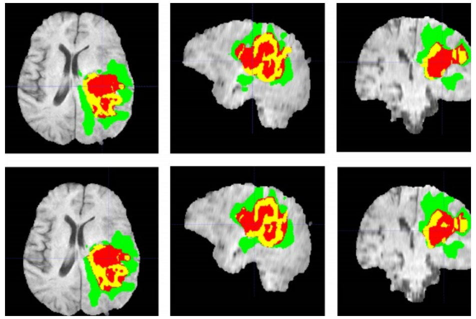
<u>Cons</u>

- Sample inefficient
- Unable to reuse prior data effectively → on-policy
- Blackbox, can be hard to debug









Lecture Outline

Recap + Policy Gradient

↓

Basic Actor Critic Algorithms

↓

Making Actor-Critic Practical

Why is Policy Gradient sample inefficient?

$$\nabla_\theta J(\theta) = \int p_\theta(\tau)\nabla_\theta \log p_\theta(\tau)d\tau$$

$$\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\sum_{t'=t}^{T} r(s_t^i, a_t^i)$$

On-policy, unable to effectively use past data

High Variance Estimator

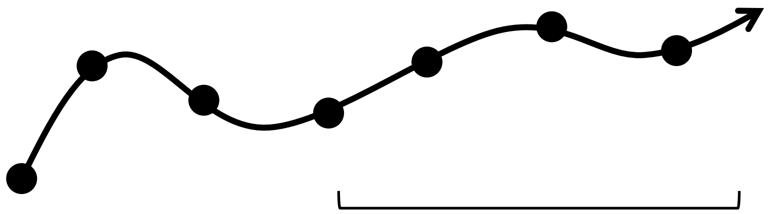
Can we develop a **low variance** **off-policy** RL algorithm that can bootstrap from prior data?

What can we do to lower variance?

$$\nabla_\theta J(\theta) = \int p_\theta(\tau)\nabla_\theta \log p_\theta(\tau)d\tau$$

$$\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\underbrace{\sum_{t'=t}^{T} r(s_t^i, a_t^i)}$$

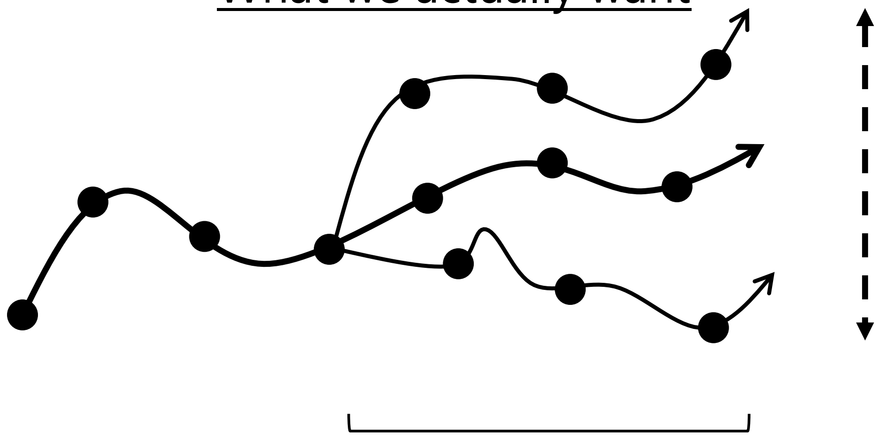
<ins>What we do</ins>



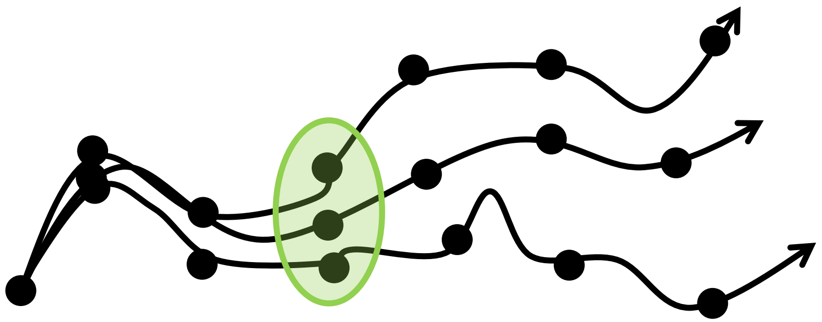
Single sample estimate

Idea: bundle this across many (s, a) with a function approximator

<ins>What we actually want</ins>



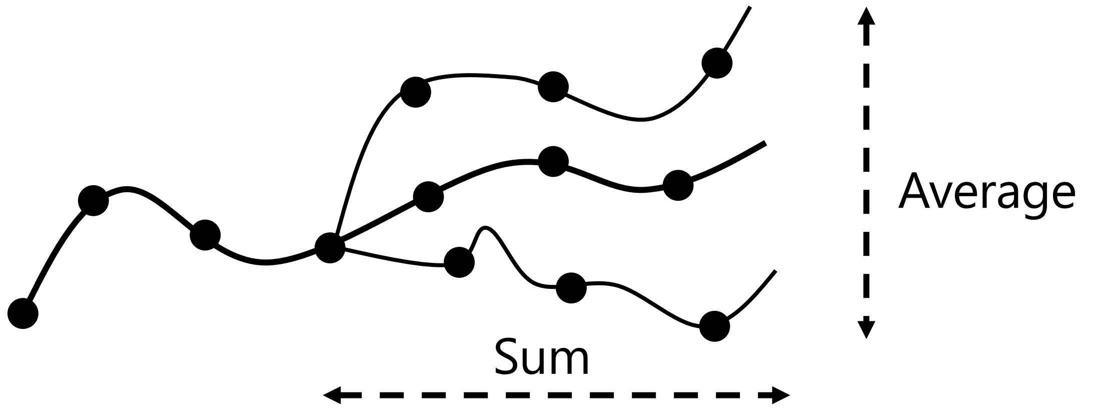
Averaged return estimate



Function approximator bundles return estimates across states

Notation: Q functions

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{t}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\sum_{t'=t}^{T} r(s_t^i, a_t^i)$$



Expected sum of rewards in the future, starting from (s, a) on first step, then π

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\pi_\theta}\left[\sum_{t'=t}^{T} r(s'_t, a'_t)|s_t, a_t\right]$$

Bundles estimates across (s, a)

Use the magic of (deep) function approximation

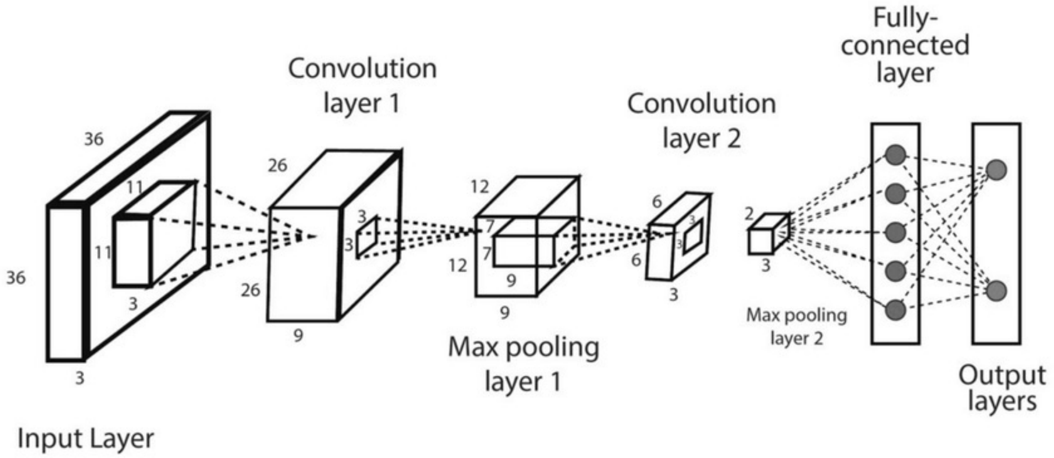
Estimation of Q-Functions

$$\frac{1}{N}\sum_{i=0}^{N}\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)Q^\pi(s_{t'}^i, a_{t'}^i)$$

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\pi_\theta}\left[\sum_{t'=t}^{T} r(s_t', a_t')|s_t, a_t\right]$$ ← Monte-carlo approximation

Idea: Regression from (s, a) to Monte-Carlo estimate

State
Action



Return to Go

Unbiased, but high variance!

Can we do better?

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{t}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\sum_{t'=t}^{T} r(s_t^i, a_t^i)$$

$$\Downarrow$$

Much lower variance if estimated well

Can be learned off-policy!

$$\frac{1}{N}\sum_{i=0}^{N}\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)Q^\pi(s_{t'}^i, a_{t'}^i)$$

Has special structure we can exploit!!

Recursive structure in Q functions

Q functions have special recursive structure!

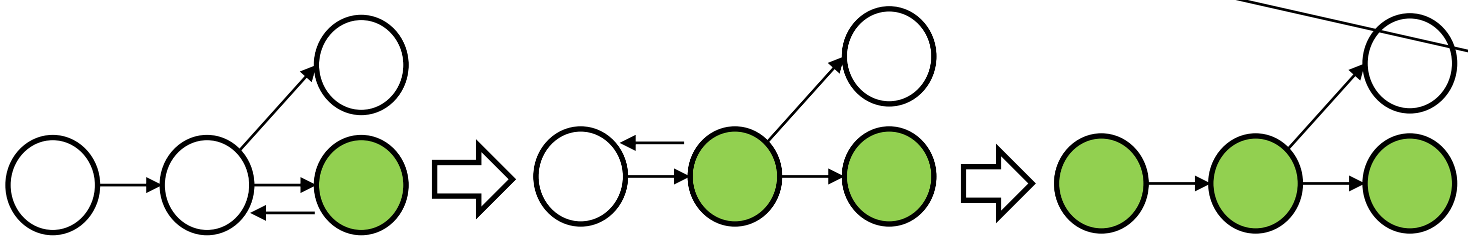
$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi_\theta}\left[\sum_{t'=t}^{T} r(s'_t, a'_t) | s_t, a_t\right]$$

$$= r(s_t, a_t) + \mathbb{E}_{\pi}\left[\sum_{t'=t+1} r(s_{t'}, a_{t'}) | s_{t+1}, a_{t+1} \sim \pi(.|s_{t+1})\right]$$

Bellman equation

$$Q^{\pi}(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{\substack{s_{t+1} \sim p(.|s_t, a_t) \\ a_{t+1} \sim \pi_\theta(.|s_{t+1})}} [Q^{\pi}(s_{t+1}, a_{t+1})]$$

Can be from different policies



Decompose temporally via dynamic programming

Learning Q-functions via Dynamic Programming

<u>Policy Evaluation:</u> Try to minimize Bellman Error (almost)

Bellman equation

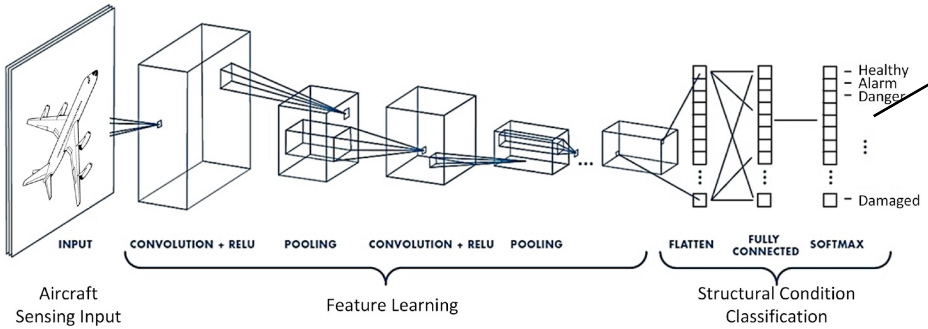
$$Q^\pi(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{\substack{s_{t+1} \sim p(.|s_t, a_t) \\ a_{t+1} \sim \pi_\theta(.|s_{t+1})}} [Q^\pi(s_{t+1}, a_{t+1}]$$

Same function approximator

How can we convert this recursion into a learning objective?

$$\min_\phi \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim \mathcal{D}} \Big(Q^\pi_\phi(s_t, a_t) - \big(r(s_t, a_t) + \mathbb{E}_{a_{t+1} \sim \pi_\theta(a_{t+1}|s_{t+1})} \left[Q^\pi_{\hat{\phi}}(s_{t+1}, a_{t+1}) \right]\big)\Big)^2$$

Off-policy



Can train via GD!

Note: this may look like gradient descent on Bellman error, it is not!

Improving Policies with Learned Q-functions

Policy Improvement: Improve policy with **policy gradient**

$$\max_{\theta} \mathbb{E}_{s \sim \mathcal{D}, a \sim \pi_\theta(a|s)} \left[Q^{\pi_\theta}(s, a) \right]$$

Replace Monte-Carlo sum of rewards with learned Q function

Lowers variance compared to policy gradient!

Policy Updates – REINFORCE or Reparameterization

Let's look a little deeper into the policy update

$$\max_\theta J(\theta) = \max_\theta \mathbb{E}_{s\sim\mathcal{D}}\mathbb{E}_{a\sim\pi_\theta(.|s)}\left[Q^\pi(s,a)\right]$$

<u>Likelihood Ratio/Score Function</u>

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\mathcal{D}}\mathbb{E}_{a\sim\pi_\theta(.|s)}\left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)\right]$$

Easier to Apply to Broad Policy Class

<u>Pathwise derivative/Reparameterization</u>

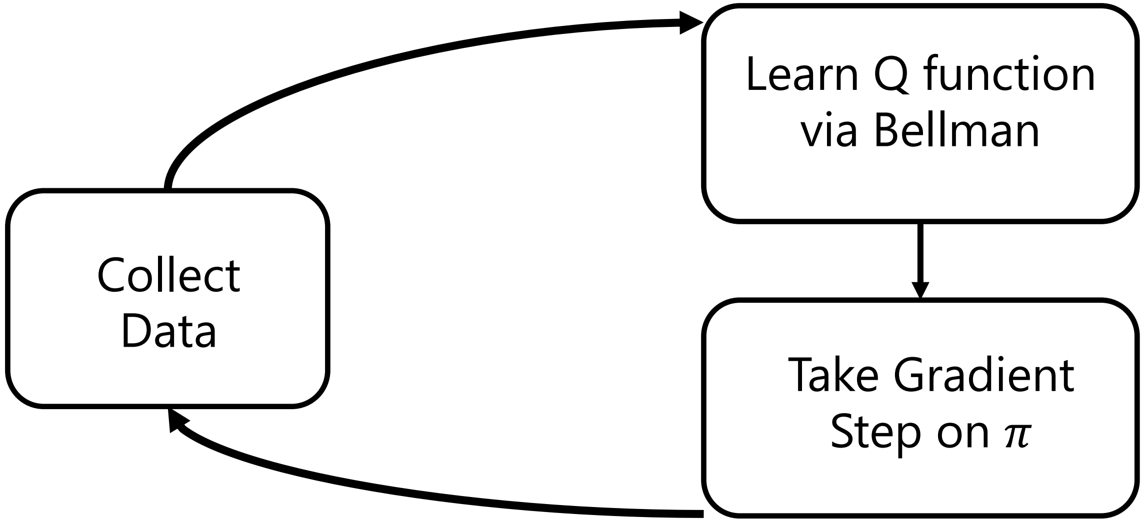
$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\mathcal{D}}\mathbb{E}_{z\sim p(z)}\left[\nabla_a Q^\pi(s,a)|_{a=\mu_\theta+z\sigma_\theta}\nabla_\theta(\mu_\theta + z\sigma_\theta)\right]$$

Lower variance

Actor-Critic: Policy Gradient in terms of Q functions

Critic: learned via the Bellman update (Policy Evaluation)

$$\min_{\phi} \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim \mathcal{D}} \left[\left(Q^{\pi}_{\phi}(s_t, a_t) - \left(r(s_t, a_t) + \mathbb{E}_{a_{t+1} \sim \pi(.|s_{t+1})} \left[Q_{\bar{\phi}}(s_{t+1}, a_{t+1}) \right] \right) \right)^2 \right]$$

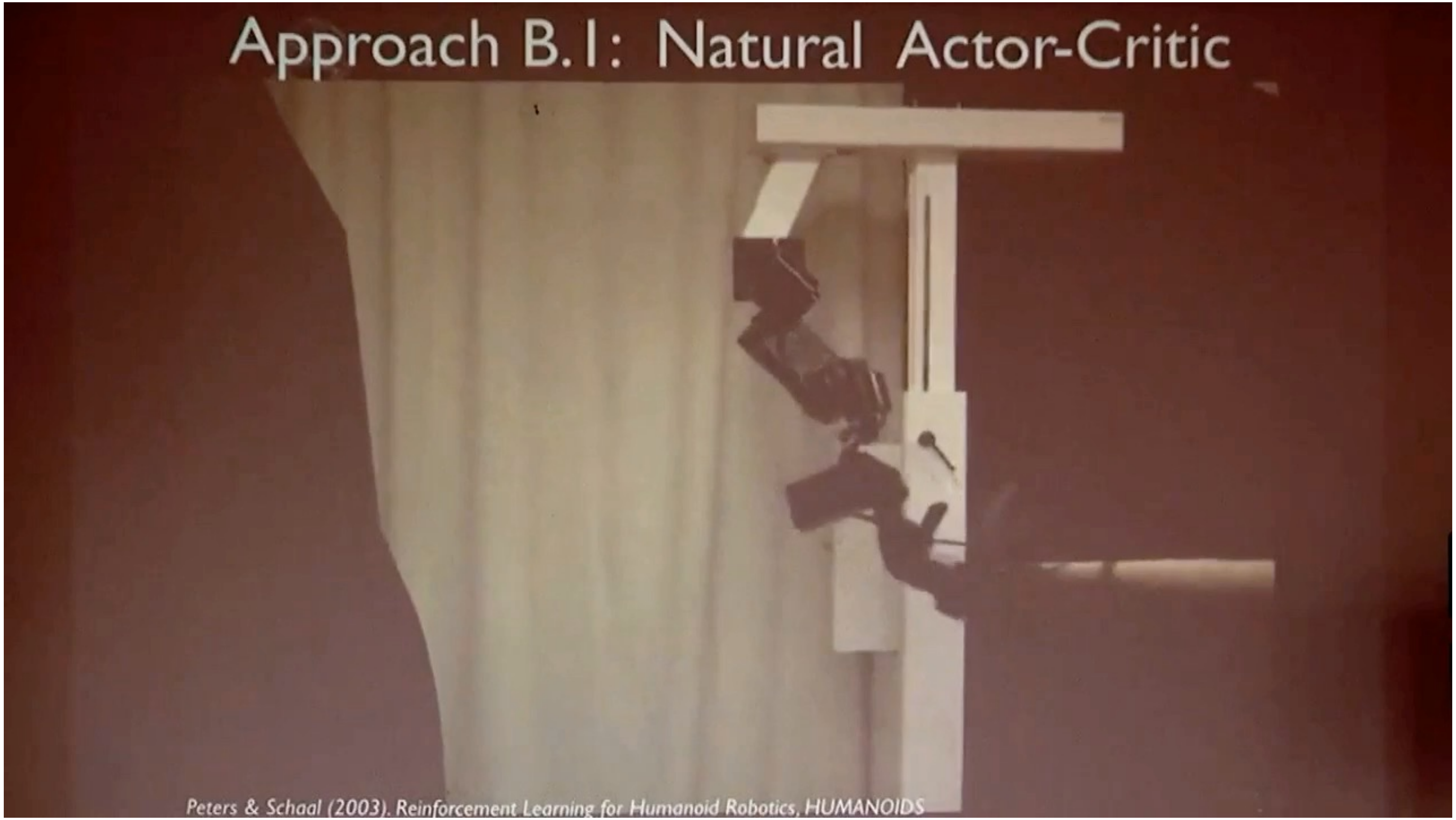


Lowers variance and is off-policy!

Actor: updated using learned critic (Policy Improvement)

$$\max_{\pi} \mathbb{E}_{s \sim \mathcal{D}} \mathbb{E}_{a \sim \pi(.|s)} \left[Q^{\pi}(s, a) \right]$$

Actor-Critic in Action



Lecture Outline

Recap + Policy Gradient

↓

Basic Actor Critic Algorithms

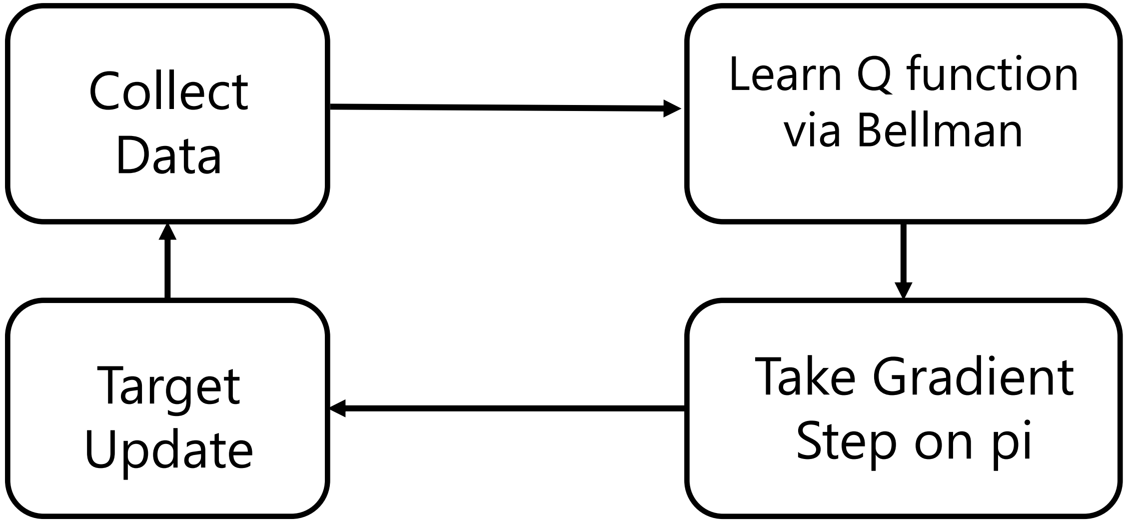
↓

Making Actor-Critic Practical

Going from Batch Updates to Online Updates

This algorithm can go from full batch mode to fully online updates

1 sample



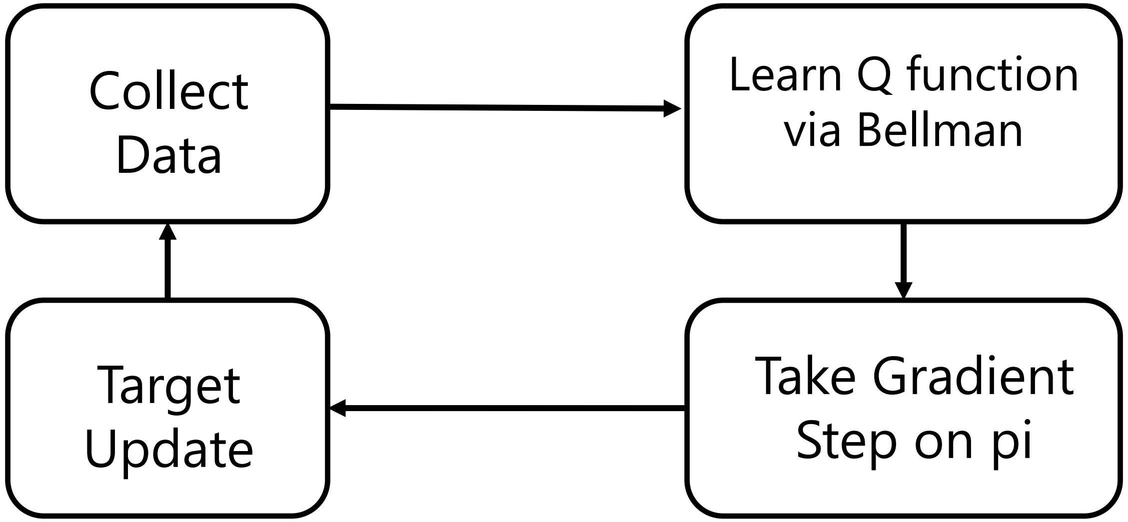
Critic: 1 gradient step on Bellman error

Actor: 1 gradient step on policy gradient

Allows for much more immediate updates

Challenges of doing online updates

1 sample



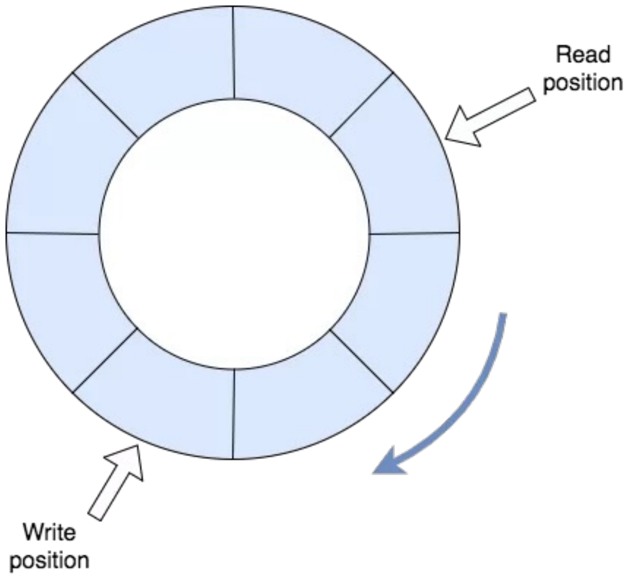
Critic: 1 gradient step on Bellman error

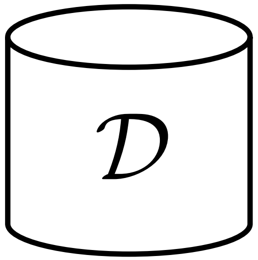
Actor: 1 gradient step on policy gradient

When updates are performed online, two issues persist:
1. Correlated updates since samples are correlated
2. Optimization objective changes constantly, unstable

Decorrelating updates with replay buffers

Updates can be decorrelated by storing and shuffling data in a replay buffer





Instead of doing updates in order, sample batches from replay buffer

↓

How?

Sampled from replay buffer

$$\min_{\phi} \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \left[Q^{\pi}_{\phi}(s_t, a_t) - \left(r(s_t, a_t) + \mathbb{E}_{a_{t+1}\sim\pi(.|s_{t+1})} \left[Q_{\bar{\phi}}(s_{t+1}, a_{t+1}) \right] \right] \right]^2$$

$$\max_{\pi} \mathbb{E}_{s\sim\mathcal{D}} \mathbb{E}_{a\sim\pi(.|s)} \left[Q^{\pi}(s, a) \right]$$

1. Sample uniformly
2. Prioritize by TD-error
3. Prioritize by target error
4. ... open area of research!

Slowing moving targets with target networks

Continuous updates can be unstable since there is a churn of projection and backup

$$\min_{\phi} \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \left[Q^{\pi}_{\phi}(s_t, a_t) - \left(r(s_t, a_t) + \mathbb{E}_{a_{t+1}\sim\pi(.|s_{t+1})} \left[Q_{\bar{\phi}}(s_{t+1}, a_{t+1}) \right]\right]^2$$

If we set $\bar{\phi}$ to ϕ every update, the update becomes very unstable

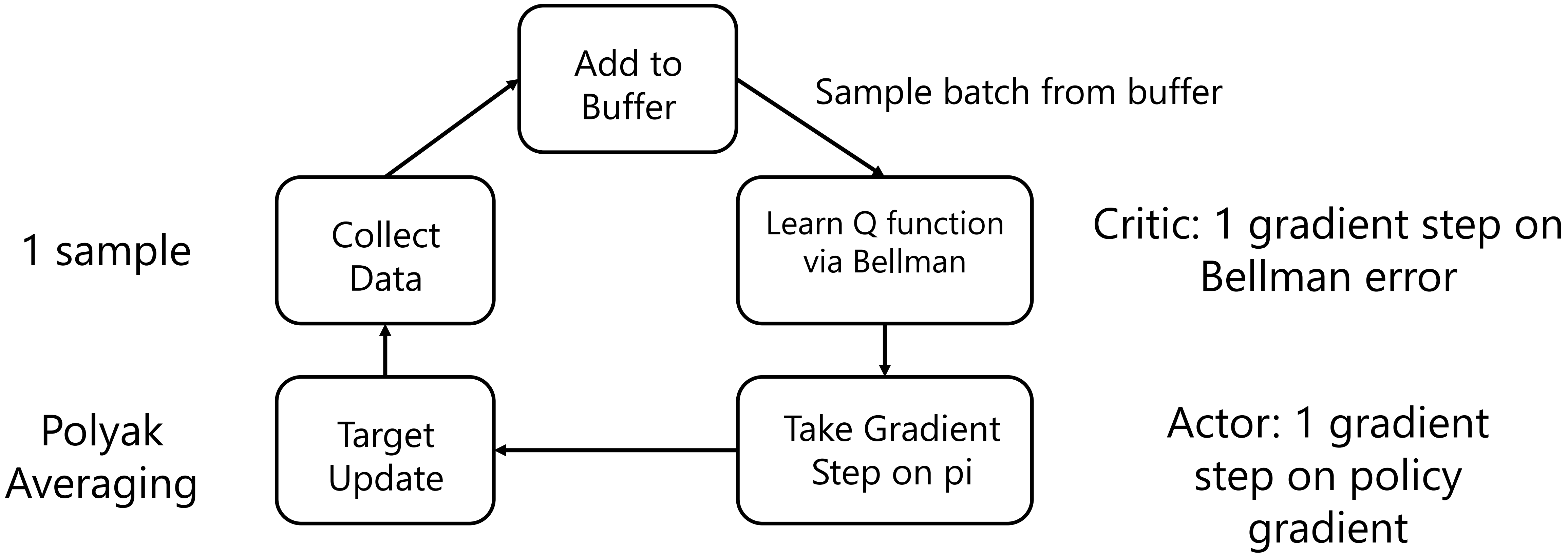
⇓

Move $\bar{\phi}$ to ϕ slowly!

$$\bar{\phi} = (1 - \tau)\phi + \tau\bar{\phi}$$

Polyak averaging

A Practical Off-Policy RL Algorithm



Simplify -- Can we get rid of a parametric actor?

Critic Update

$$\min_{\phi} \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \left[Q^{\pi}_{\phi}(s_t, a_t) - (r(s_t, a_t) + \mathbb{E}_{a_{t+1}\sim\pi(.|s_{t+1})} \left[Q_{\bar{\phi}}(s_{t+1}, a_{t+1})\right]\right]^2$$

Actor Update

$$\max_{\pi} \mathbb{E}_{s\sim\mathcal{D}}\mathbb{E}_{a\sim\pi(.|s)} \left[Q^{\pi}(s, a)\right]$$

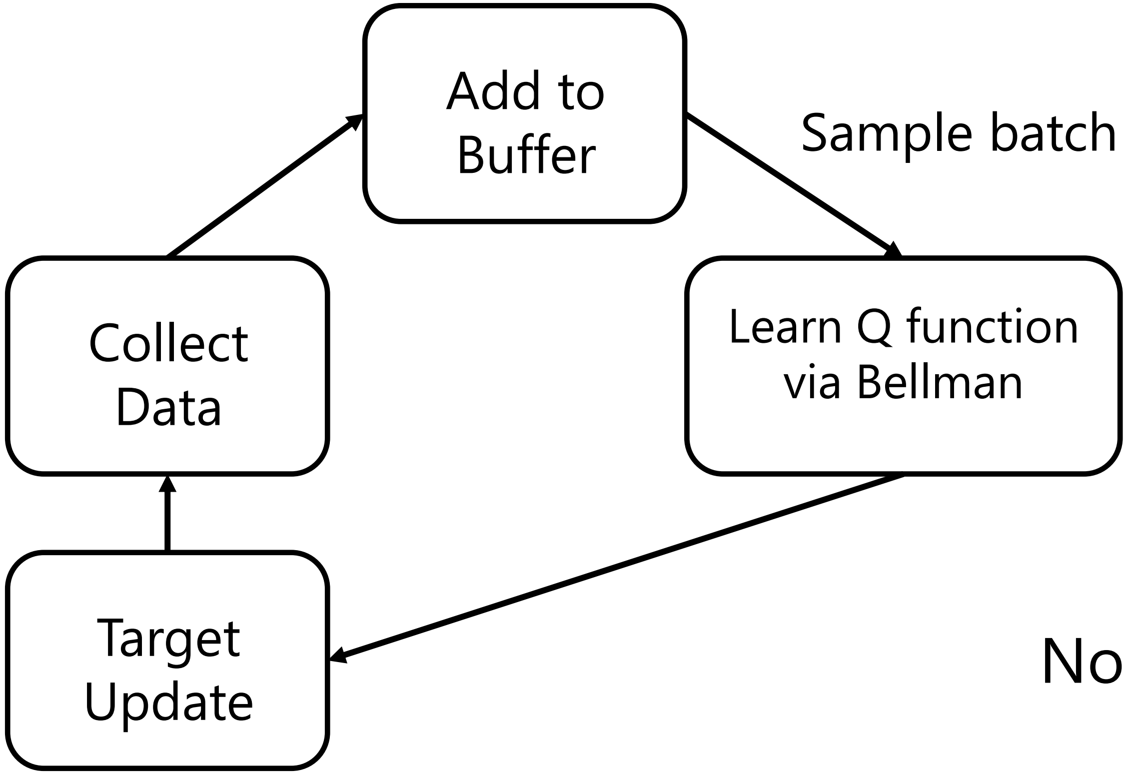
What if we removed this explicit actor completely?

Directly Learning Q*

$$\min_{\phi} \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \left[\left[Q^{\pi}_{\phi}(s_t, a_t) - \left(r(s_t, a_t) + \max_{a_{t+1}} \left[Q_{\bar{\phi}}(s_{t+1}, a_{t+1}) \right] \right) \right]^2 \right]$$

$$\pi(a|s) = \max_{a} Q(s, a)$$

Directly do max in the Bellman update



Sample batch from buffer

1 sample

Critic: 1 gradient step on Bellman error

Polyak Averaging

No actor updates, just learn Q!

How can we maximize w.r.t a?

$$\pi(a|s) = \max_a Q(s,a)$$

↑

Analytic maximization can be very difficult to perform in continuous action spaces
Much easier in discrete spaces! → just do categorical max!

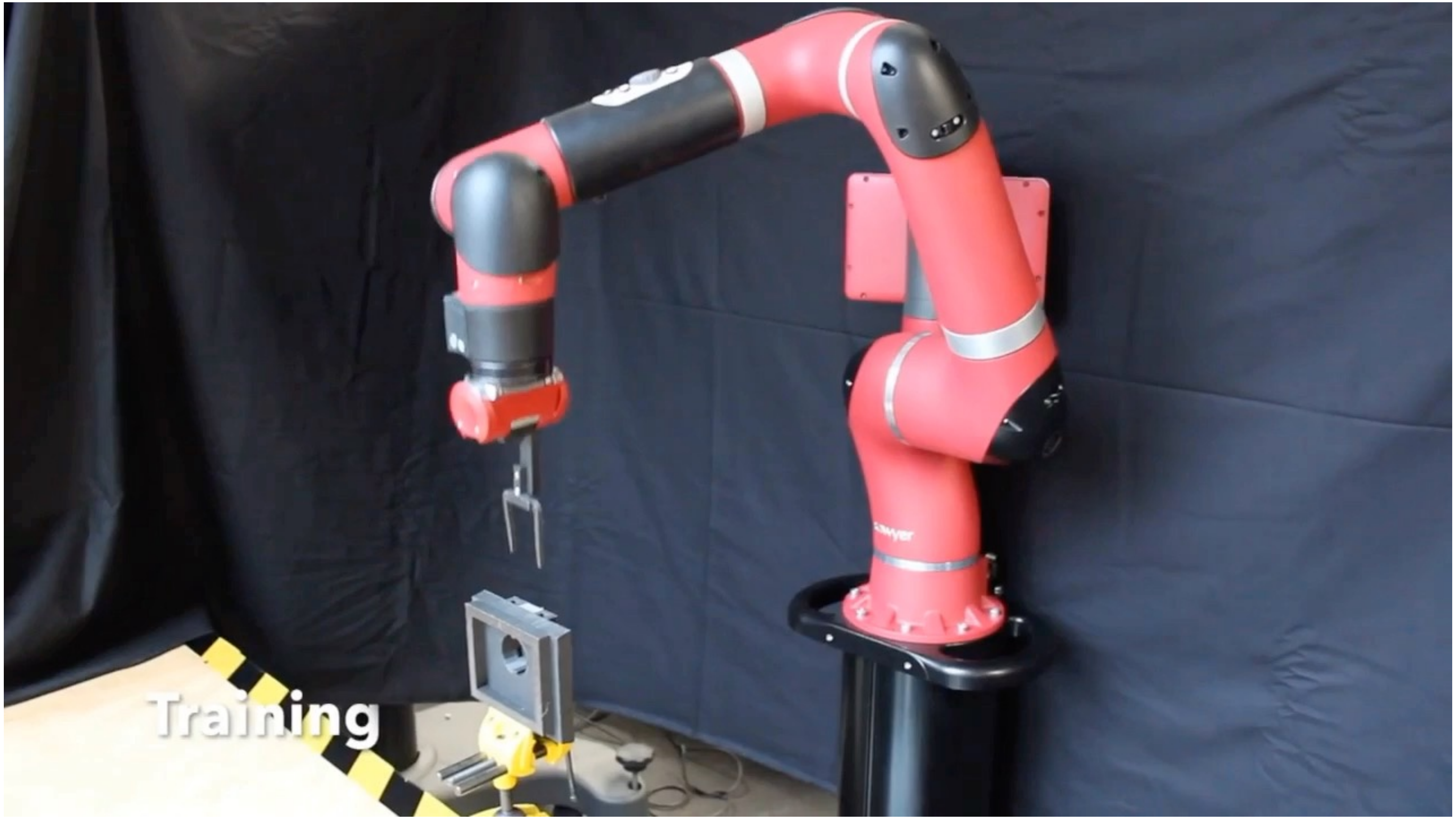
Some ideas to do maximization:

1. Sampling based (QT-opt (Kalashnikov et al))
2. Optimization based (NAF, Gu et al)

Practical Actor-Critic in Action

Trained using QT-Opt

Practical Actor-Critic in Action



Trained using DDPG

What makes off-policy RL hard?

Deadly triad:

1. Function Approximation
2. Bootstrapping
3. Off-policy learning

$$\min_{\phi} \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \left[\left[Q^{\pi}_{\phi}(s_t, a_t) - \left(r(s_t, a_t) + \max_{a_{t+1}} \left[Q_{\bar{\phi}}(s_{t+1}, a_{t+1}) \right] \right) \right]^2 \right]$$

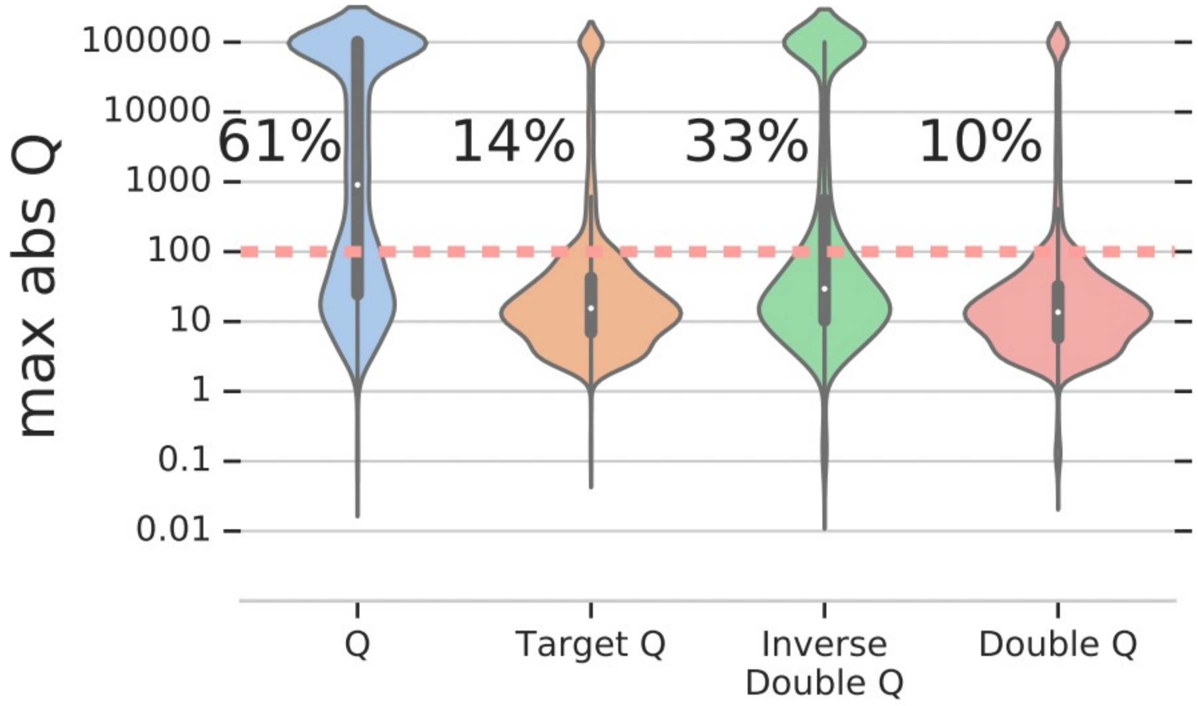
These in combination lead to many of the difficulties in stabilizing off-policy RL with function approximation

Zooming out – what makes off-policy RL hard?

Deadly triad:

1. Function Approximation
2. Bootstrapping
3. Off-policy learning

61% of runs show divergence of Q-values



Diverges even with linear function approximation, when off-policy + bootstrapping

Lecture Outline

Recap + Policy Gradient

↓

Basic Actor Critic Algorithms

↓

Making Actor-Critic Practical

Recap: Course Overview

Filtering/Smoothing
Localization
Mapping
SLAM

Search
Motion Planning
TrajOpt
Stability/Certification

MDPs and RL
Imitation Learning
Off-Policy/MBRL