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

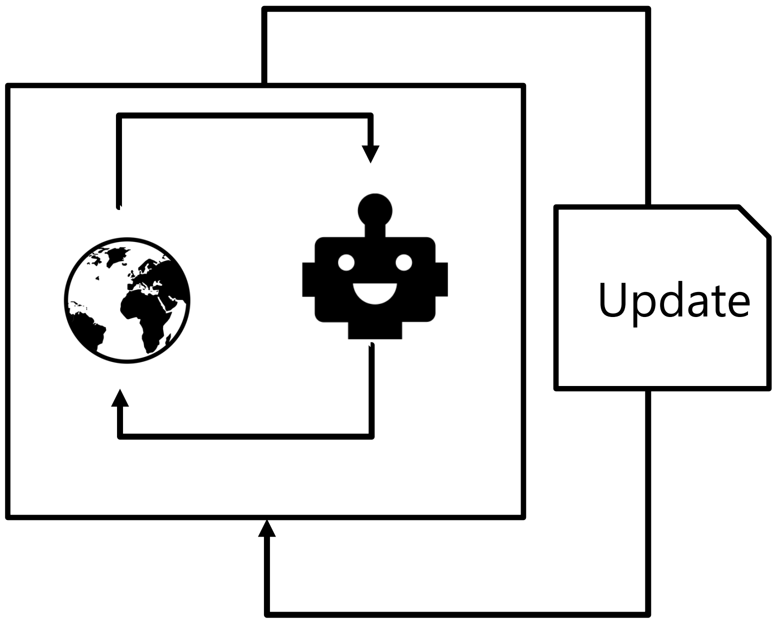
Recap + Actor Critic

Making Actor-Critic Practical

Model-Based RL

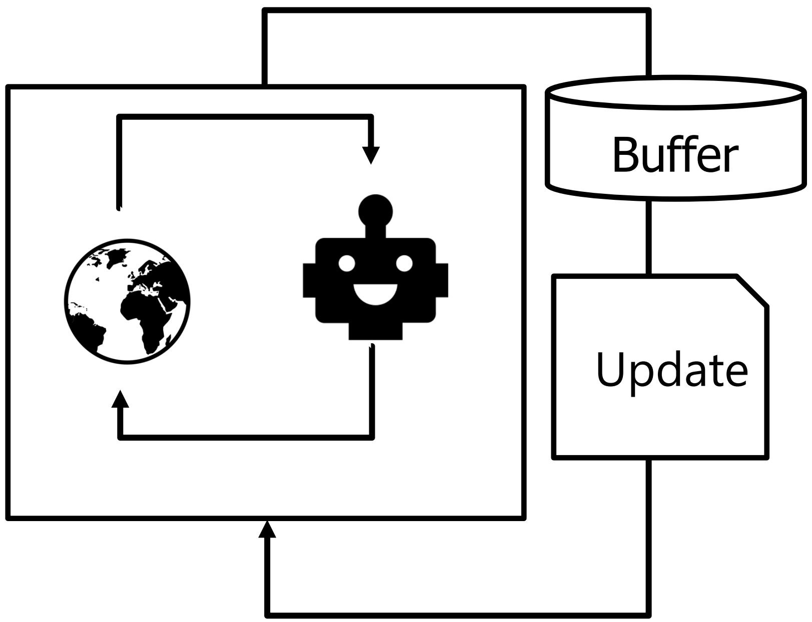
Learning Algorithms for Robotics

On-Policy Algorithms



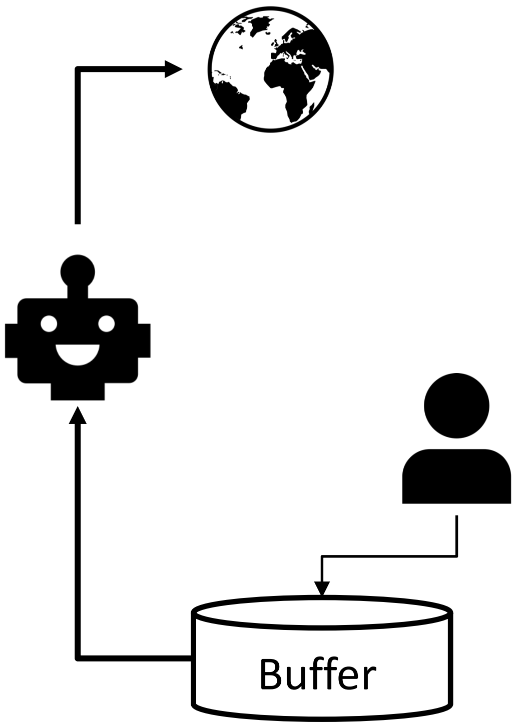
Simple, performant,
Data inefficient

Off-Policy Algorithms



Data-efficient,
sometimes unstable

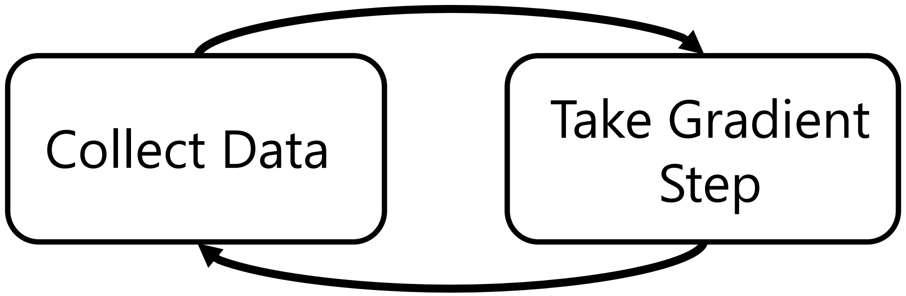
Imitation Learning



Performant, efficient, but
compounding error and
expensive data collection

Policy Gradient - REINFORCE

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



REINFORCE algorithm:

On-policy

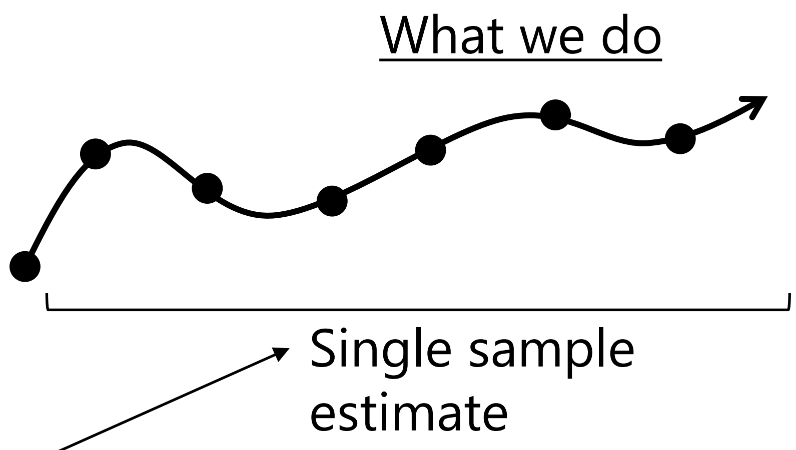
1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha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

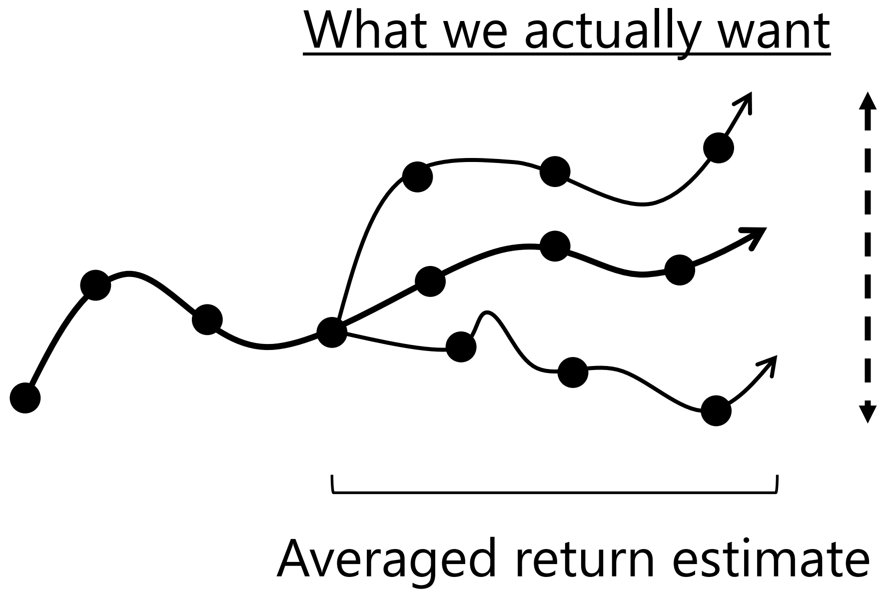
What makes policy gradient challenging?

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

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

High variance estimator!!





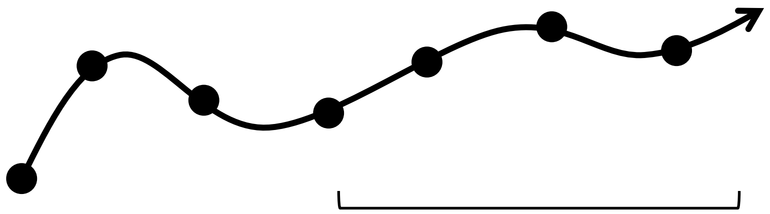
Hard to tell what matters without many samples

What can we do to lower variance?

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

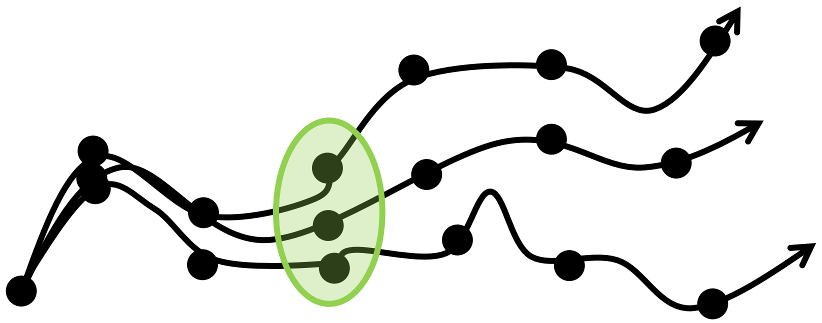
$$\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\underbrace{\sum_{t'=t}^{T} r(s_t^i, a_t^i)}_{}$$

<u>What we do</u>



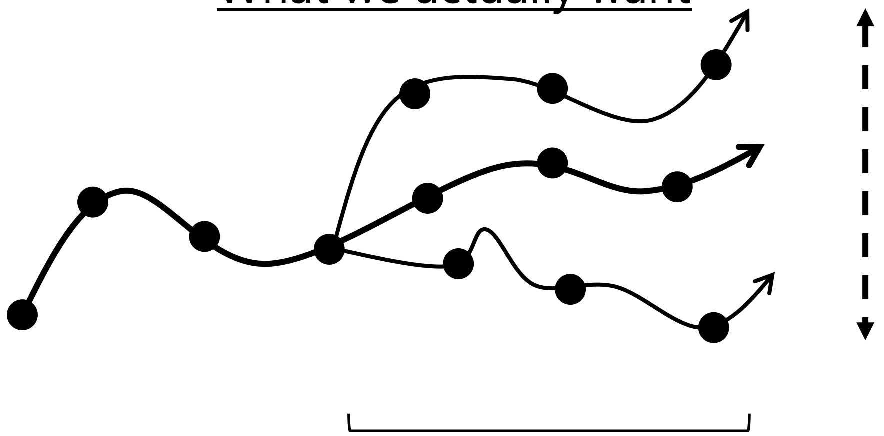
Single sample estimate

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



Function approximator bundles return estimates across states

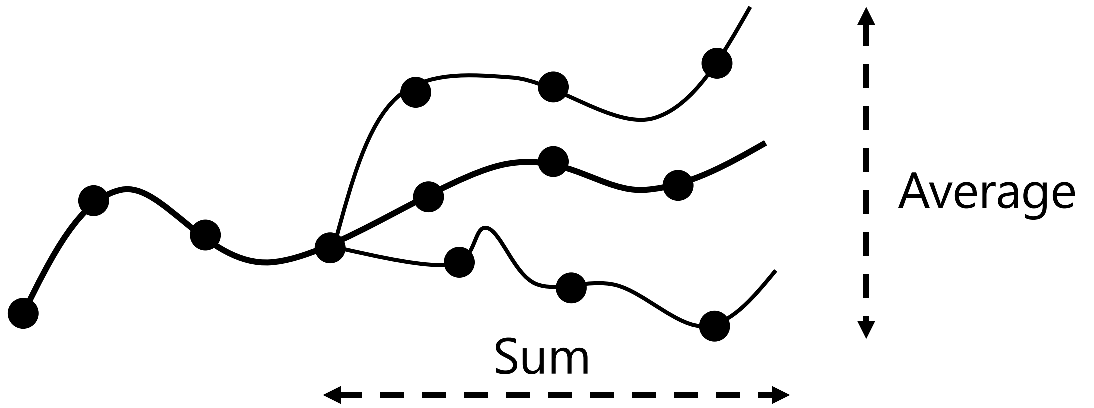
<u>What we actually want</u>



Averaged return estimate

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

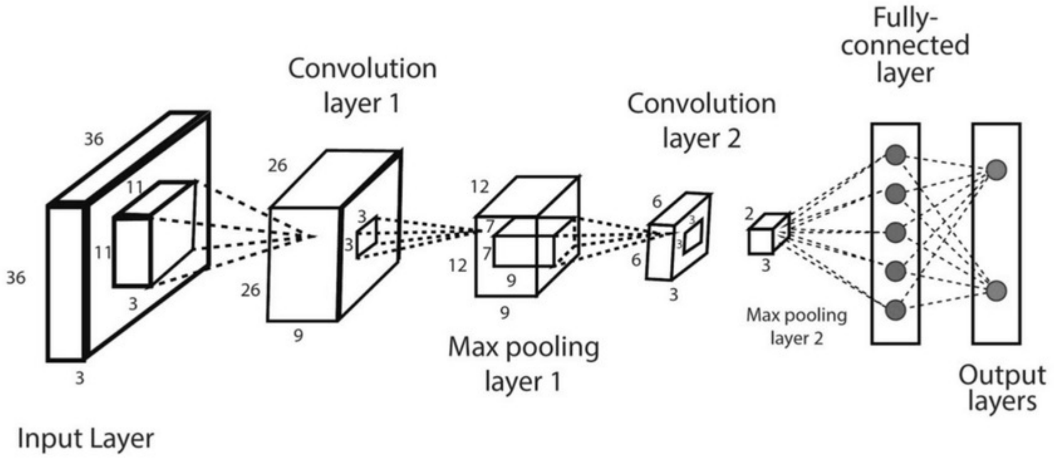
Estimation of Q-Functions

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

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

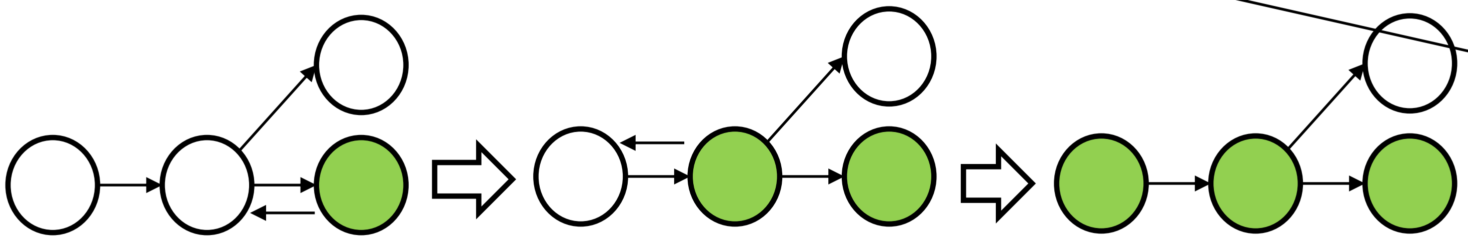
$$Q^\pi(s_t, a_t) = \mathbb{E}_{\pi_\theta}\left[\sum_{t'=t}^{T} r(s'_t, a'_t) | s_t, a_t\right]$$

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

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

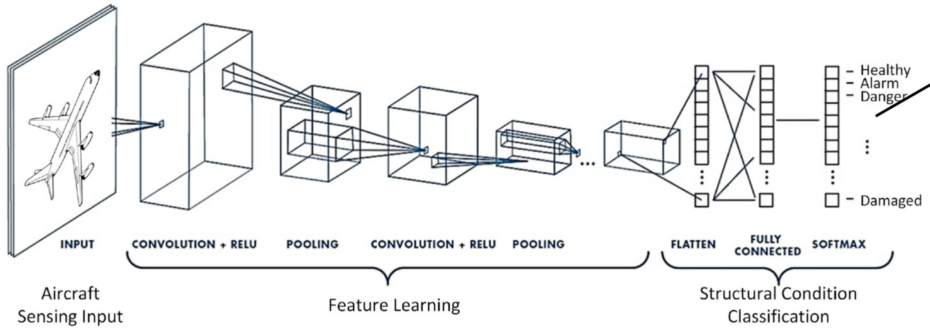
$$Q^\pi(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{\substack{s_{t+1} \sim p(.|s_t, a_t) \\ a_{t+1} \sim \pi_\theta(.|s_{t+1})}} \left[Q^\pi(s_{t+1}, a_{t+1}\right]$$

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>

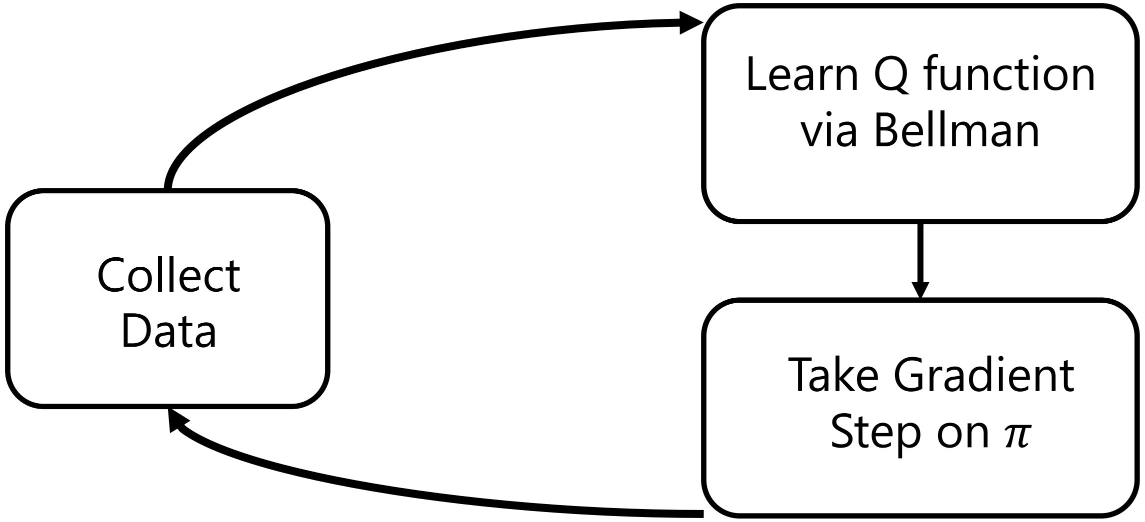
$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\mathcal{D}}\mathbb{E}_{z\sim p(z)}\left[\nabla_a Q^\pi(s,a)|_{a=\mu_\theta+z\sigma_\theta}\nabla_\theta(\mu_\theta + z\sigma_\theta)\right]$$

Lower variance

Actor-Critic: Policy Gradient in terms of Q functions

Critic: learned via the Bellman update (Policy Evaluation)

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

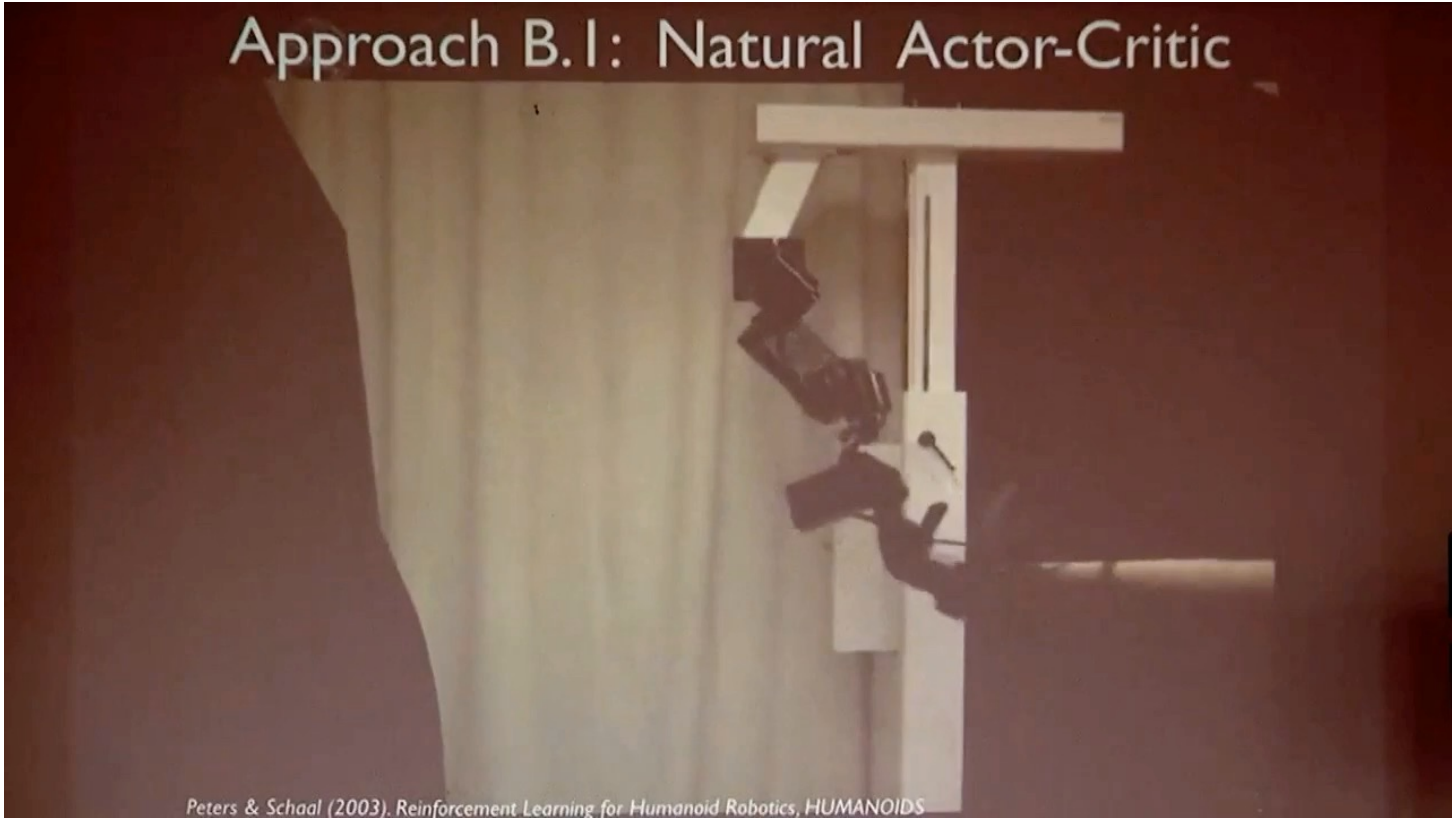


Lowers variance and is off-policy!

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 + Actor Critic

↓

Making Actor-Critic Practical

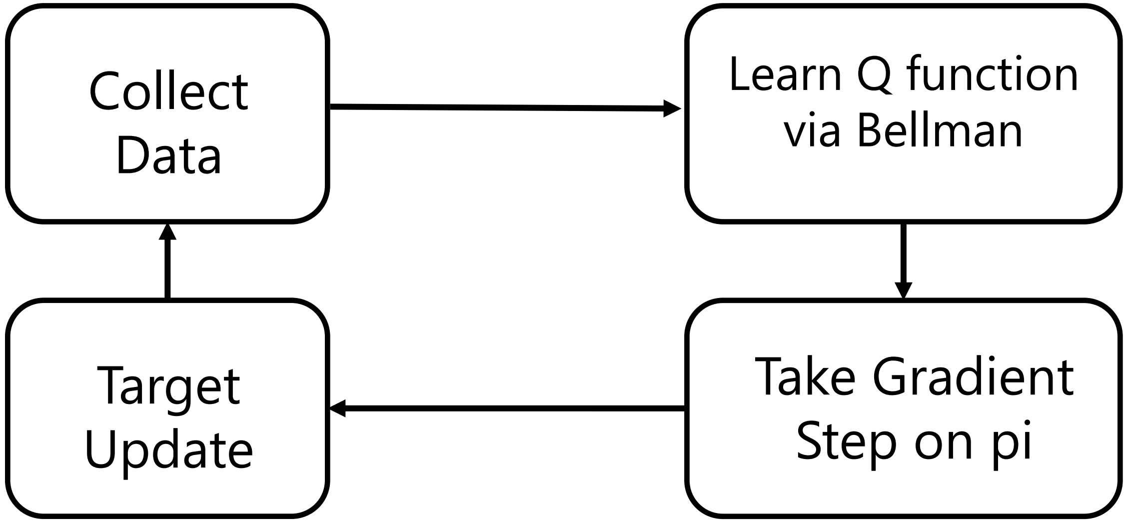
↓

Model-Based RL

Going from Batch Updates to Online Updates

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

1 sample



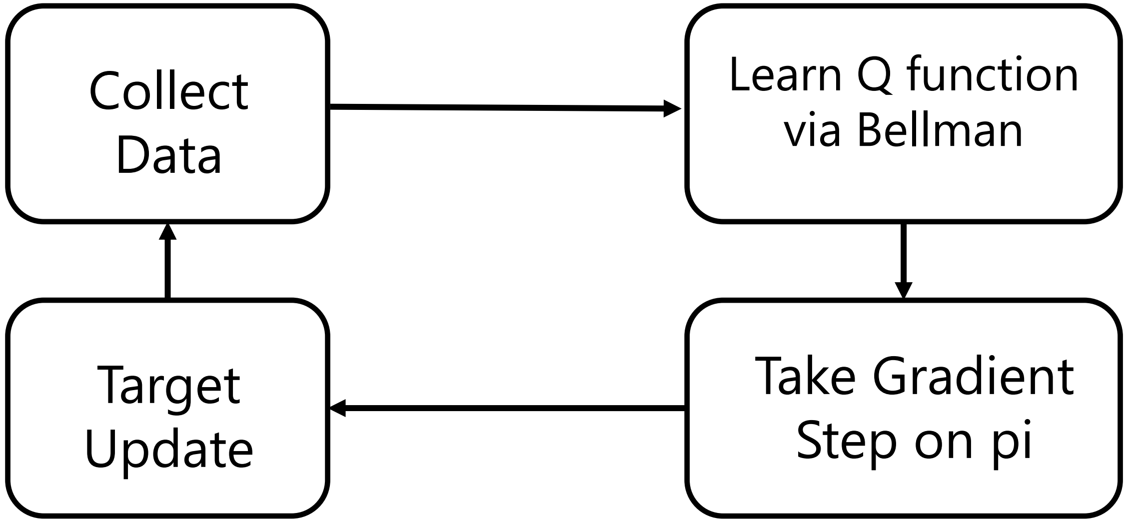
Critic: 1 gradient step on Bellman error

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

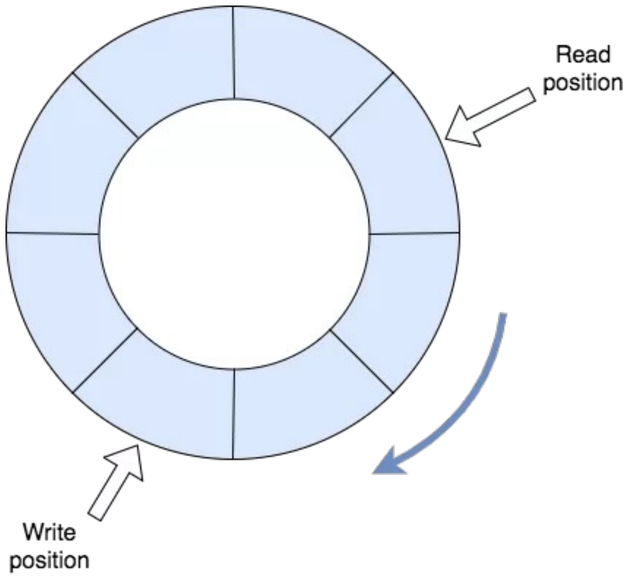
Actor: 1 gradient step on policy gradient

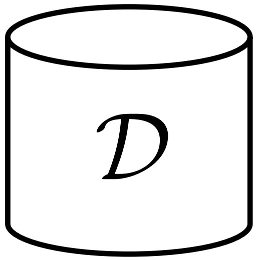
When updates are performed online, two issues persist:

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

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

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

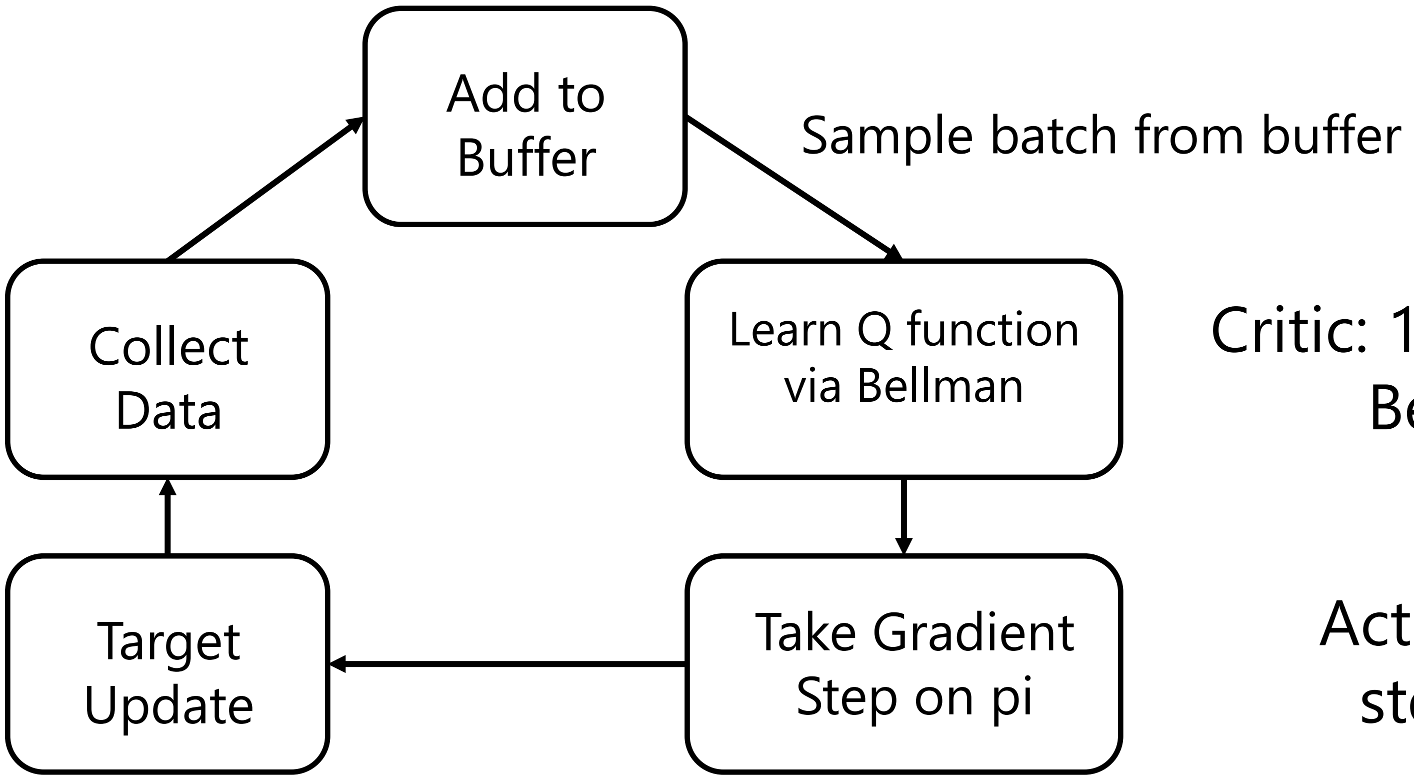
⇩

Move $\bar{\phi}$ to ϕ slowly!

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

Polyak averaging

A Practical Off-Policy RL Algorithm



1 sample

Polyak Averaging

Critic: 1 gradient step on Bellman error

Actor: 1 gradient step on policy gradient

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

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

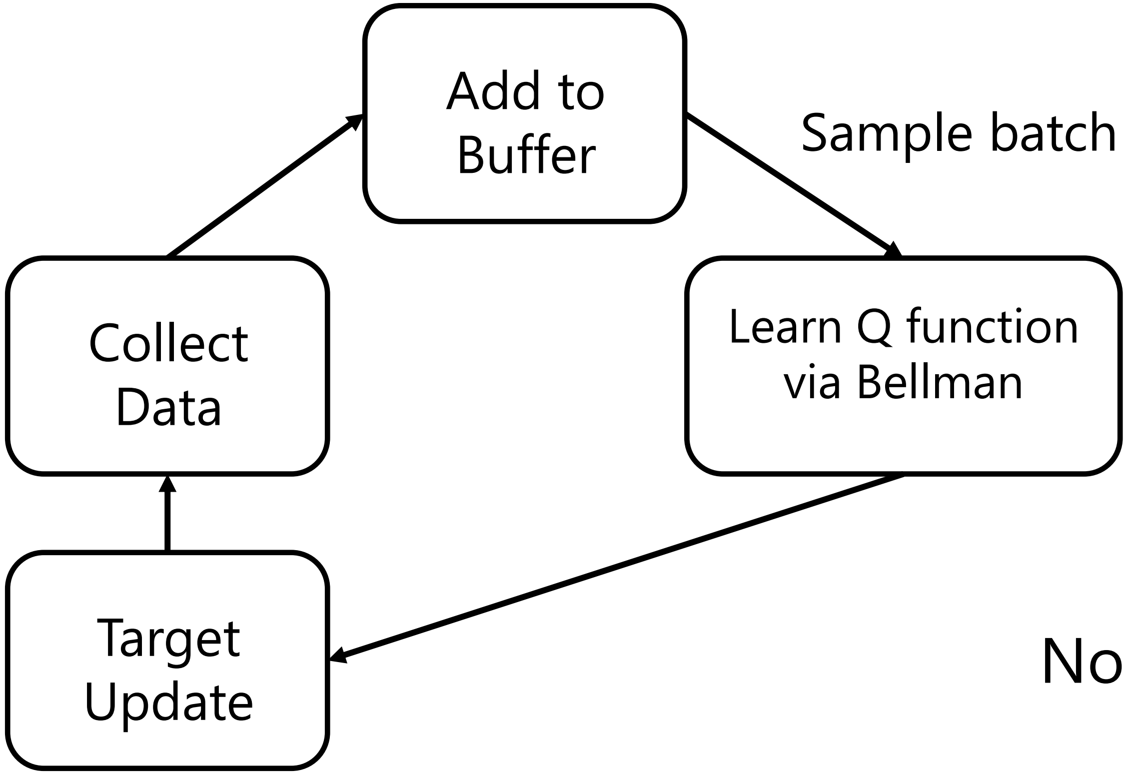
What if we removed this explicit actor completely?

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



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!

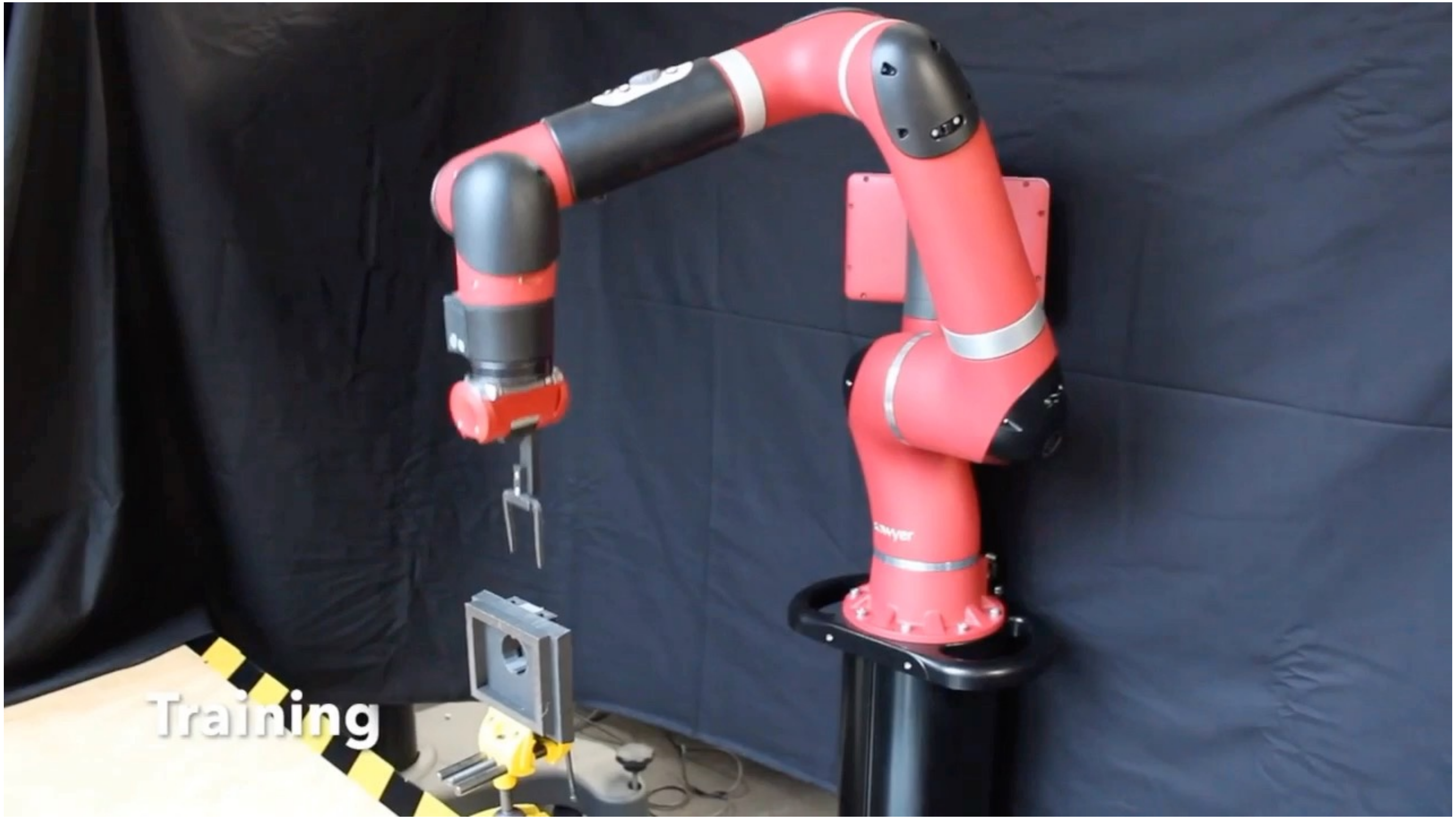
Some ideas to do maximization:

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

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

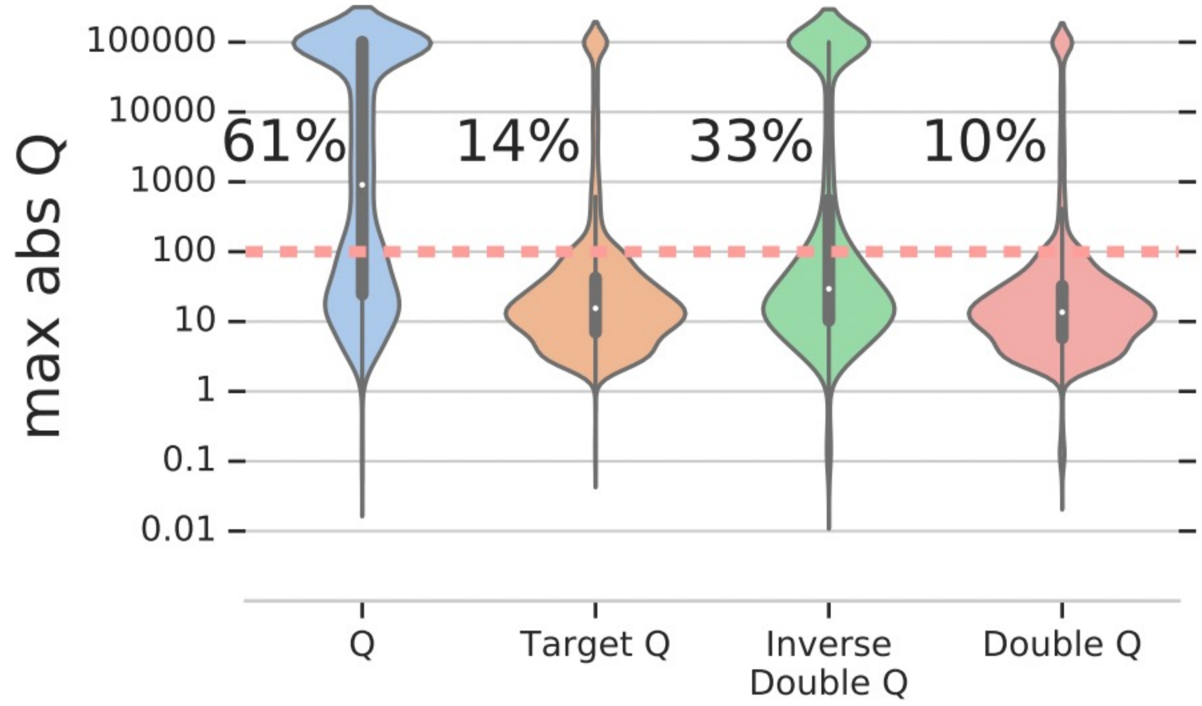
These in combination lead to many of the difficulties in stabilizing off-policy RL with function approximation

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

Deadly Triad, Van Hasselt et al

Lecture Outline

Recap + Actor Critic

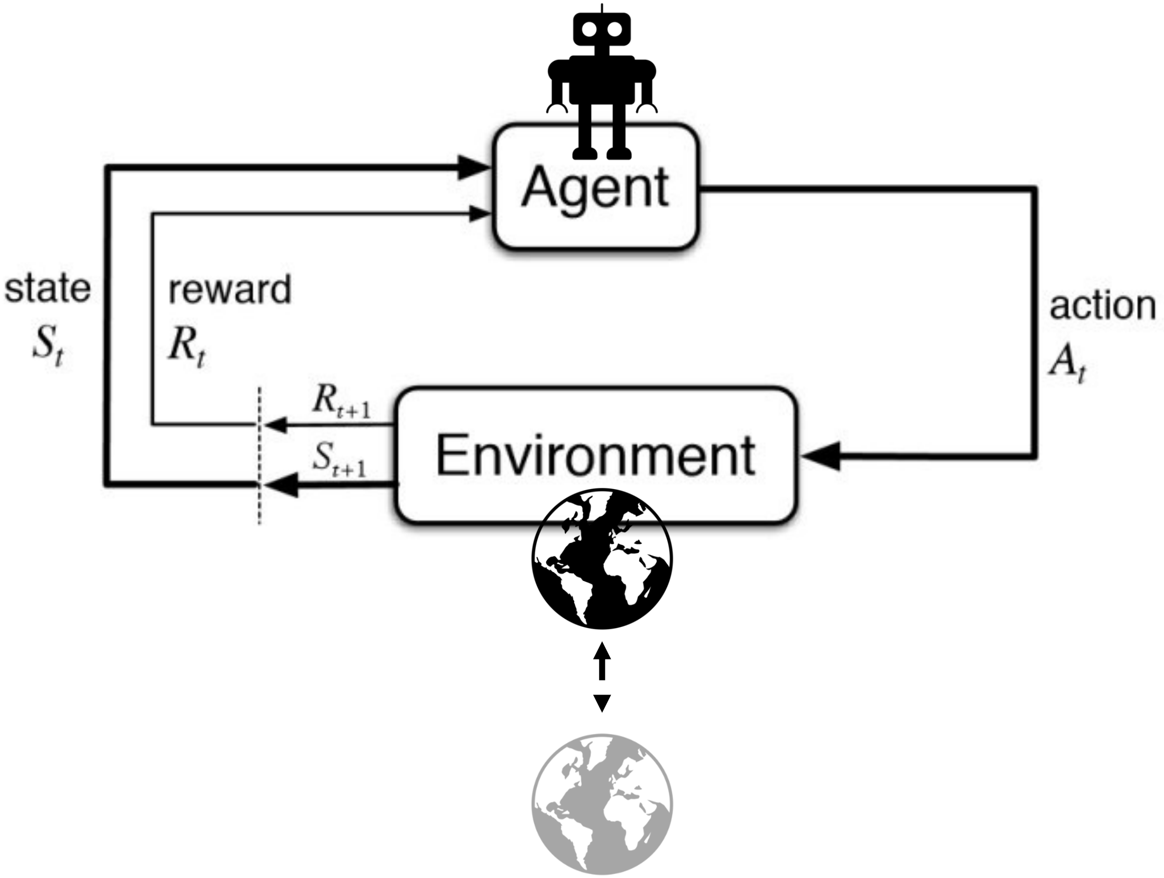
↓

Making Actor-Critic Practical

↓

Model-Based RL

What if we just learned how the world worked?



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

1. Learn a surrogate model of the transition dynamics from arbitrary off-policy data
2. Do reward maximization against this model

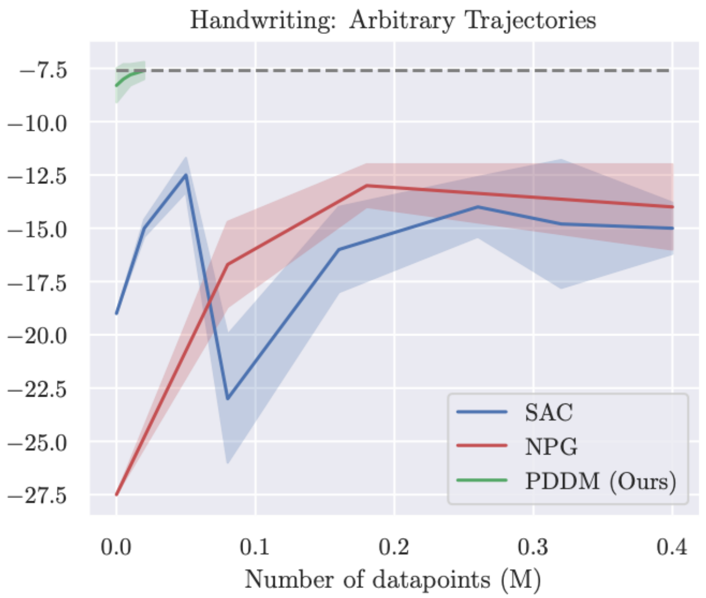
Intuitive: learn how the world works first and then plan in that model

Why do model-based RL?

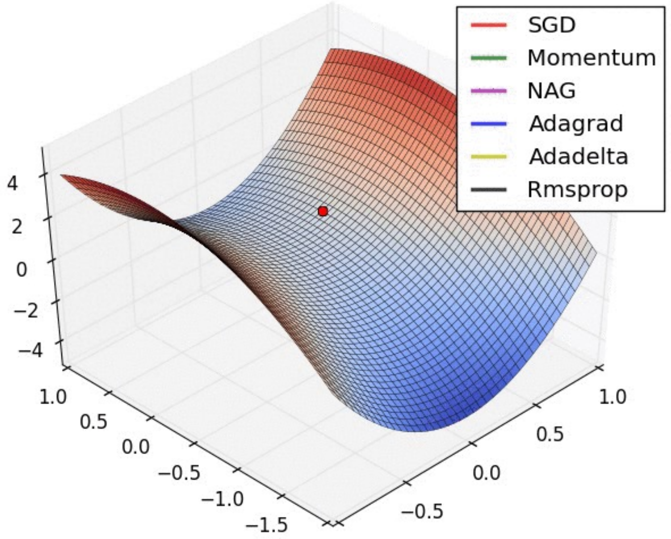
Why would we do this?

Transfer/Adaptive

Efficiency



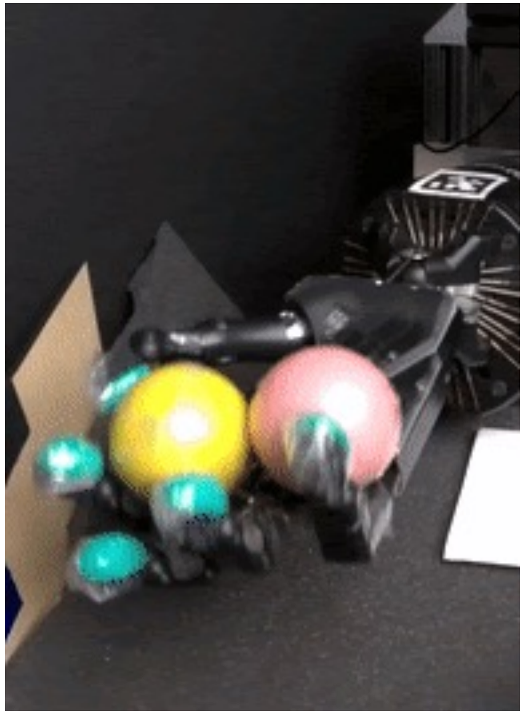
Simplicity



Naturally off-policy!

Why do model-based RL?

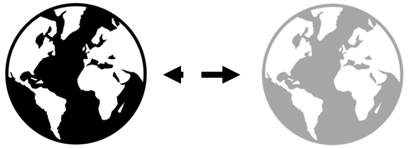


Just 2 hours of real robot training

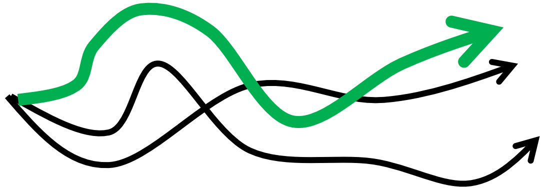
Model Based RL – Problem Statement

Model Learning



$$\hat{p}_\theta \leftarrow \arg\min_{\hat{p}_\theta} \mathcal{L}(\mathcal{D}, \hat{p}_\theta)$$

Planning

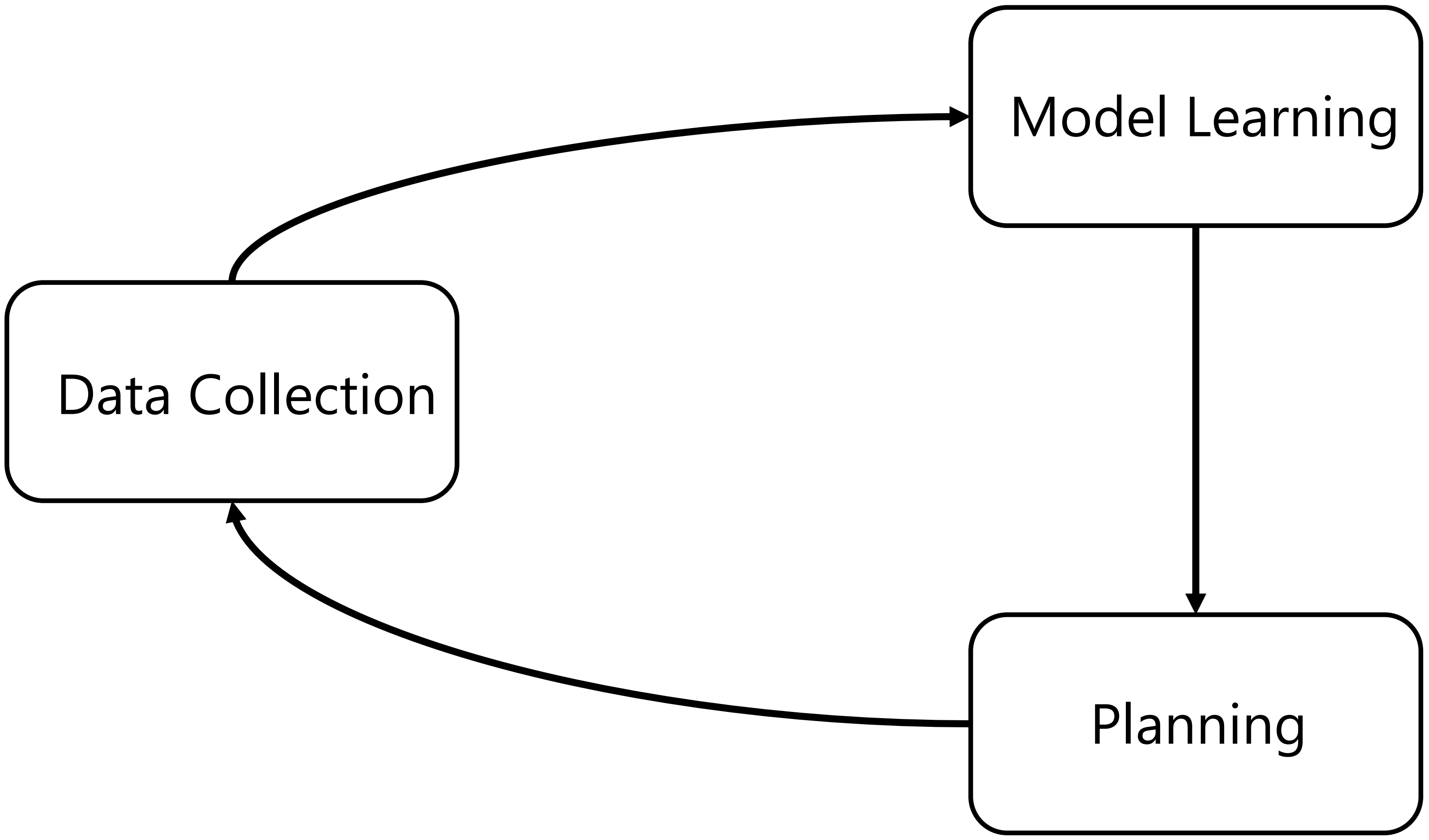


$$\arg\max_{\pi} \mathbb{E}_{\hat{p},\pi} \left[\sum_t r(s_t, a_t) \right]$$

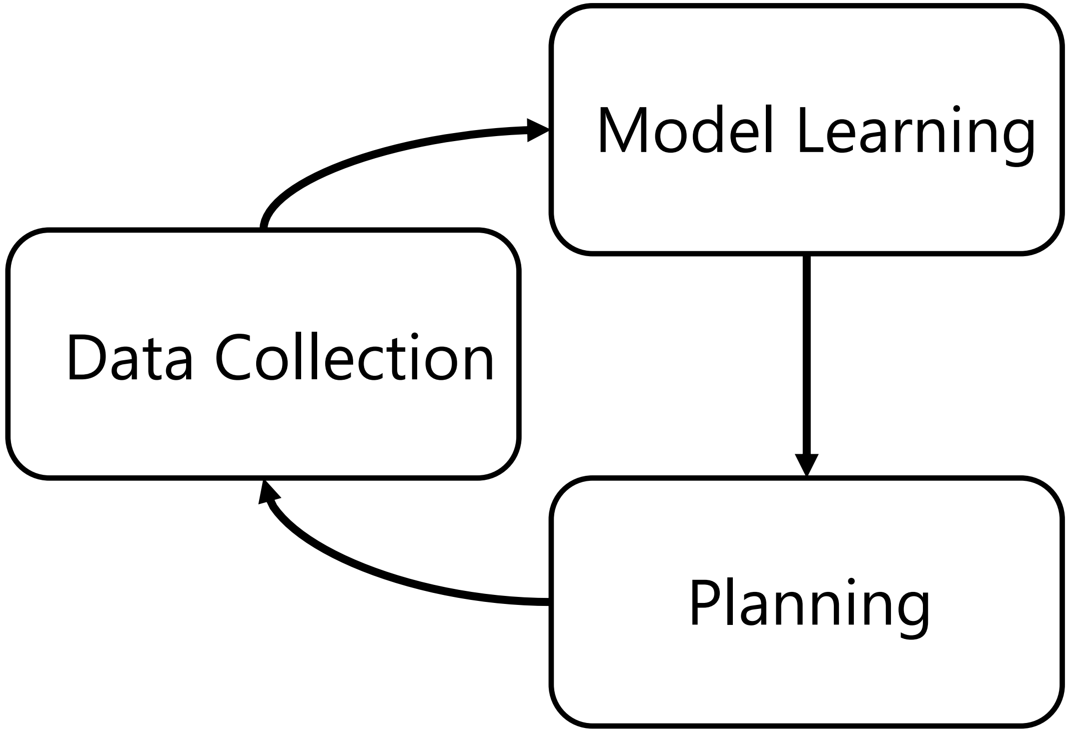
Can also just be a single trajectory

How should we instantiate these?

Model Based RL – **A template**



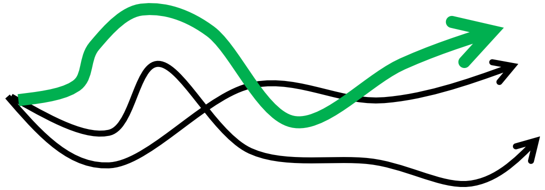
Model Based RL – **Naïve Algorithm (v0)**



Maximum likelihood supervised Learning

$$\max_{\theta} \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \left[\log \hat{p}_{\theta}(s'|s,a)\right]$$

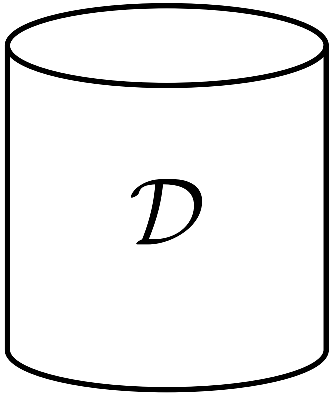
Random Search

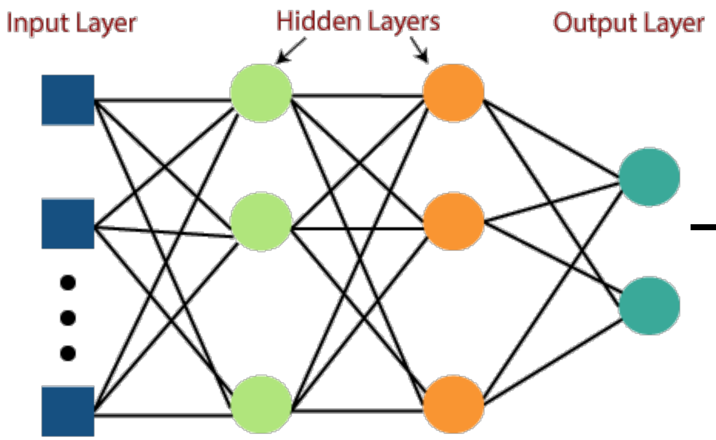


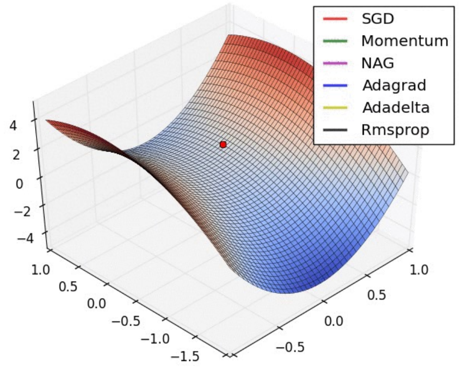
Model Based RL – **Naïve Algorithm (Model Learning) (v0)**

$$\max_{\theta} \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \left[\log \hat{p}_\theta(s'|s,a)\right]$$

Fit 1-step models







More expressive may be better, at the risk of overfitting

Choice of $\hat{p}_\theta$ distribution determines the loss function:

1. Gaussian → L_2
2. Energy Based Model → Contrastive Divergence
3. Diffusion Model → Score Matching

Trick: Model Residual's (s' – s)

Model Based RL – **Naïve Algorithm (Planning)**

Planning

$$\max_{a_0, a_1, \ldots, a_T} \sum_{t=0}^{T} r(\hat{s}_t, a_t)$$

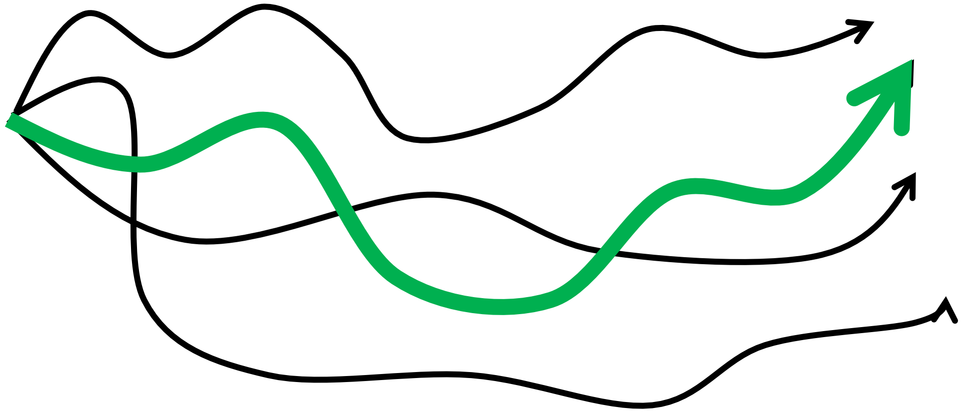
$$\hat{s}_{t+1} \sim \hat{p}_\theta(s_{t+1} | \hat{s}_t, a_t)$$

$$\hat{s}_1 \sim \hat{p}_\theta(s_{t+1} | s_0, a_0)$$

Just do random search!

$$\arg \max_{a_0^j, a_1^j, \ldots, a_T^j} \sum_{t=0}^{T} r(\hat{s}_t^j, a_t^j)$$

$$\hat{s}_{t+1}^j \sim \hat{p}_\theta(. | \hat{s}_t^j, a_t^j)$$

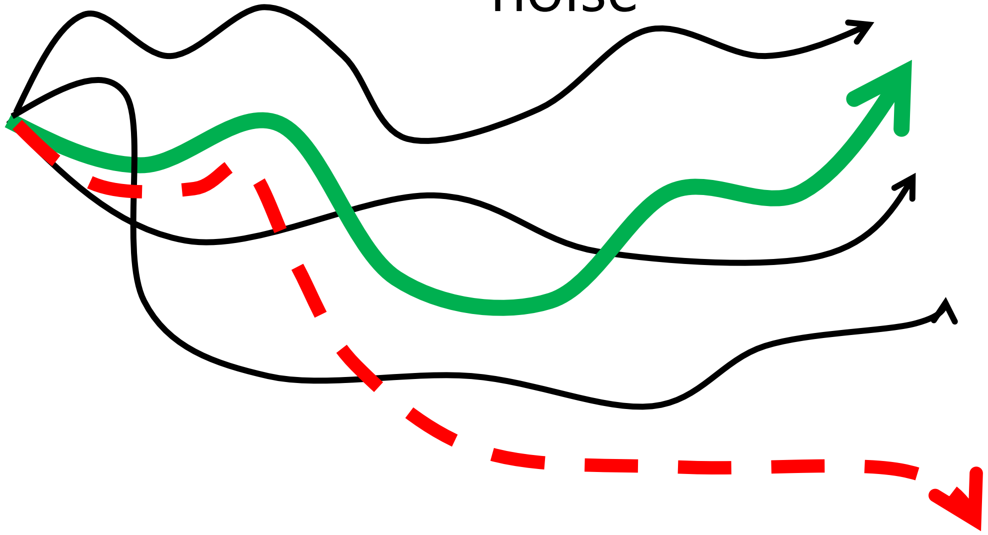


Just execute actions open loop!

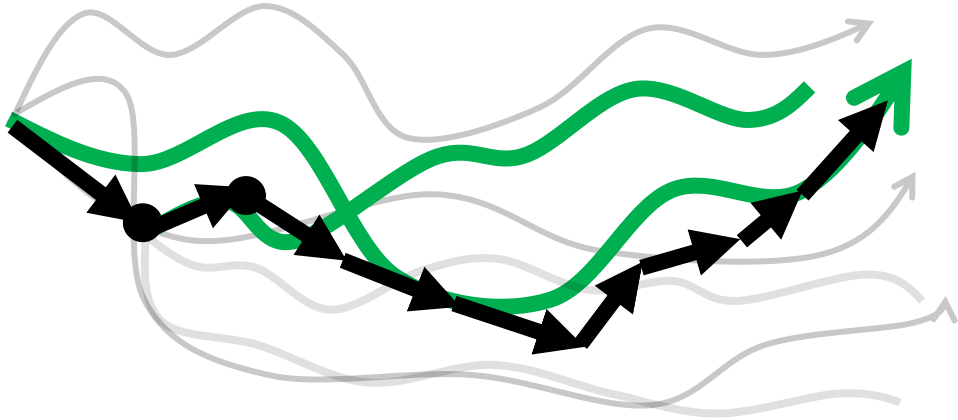
Can soften by taking softmax rather than argmax

Model Based RL – **Naïve Algorithm (MPC)**

Without feedback, an open loop controller can diverge even for minimal noise



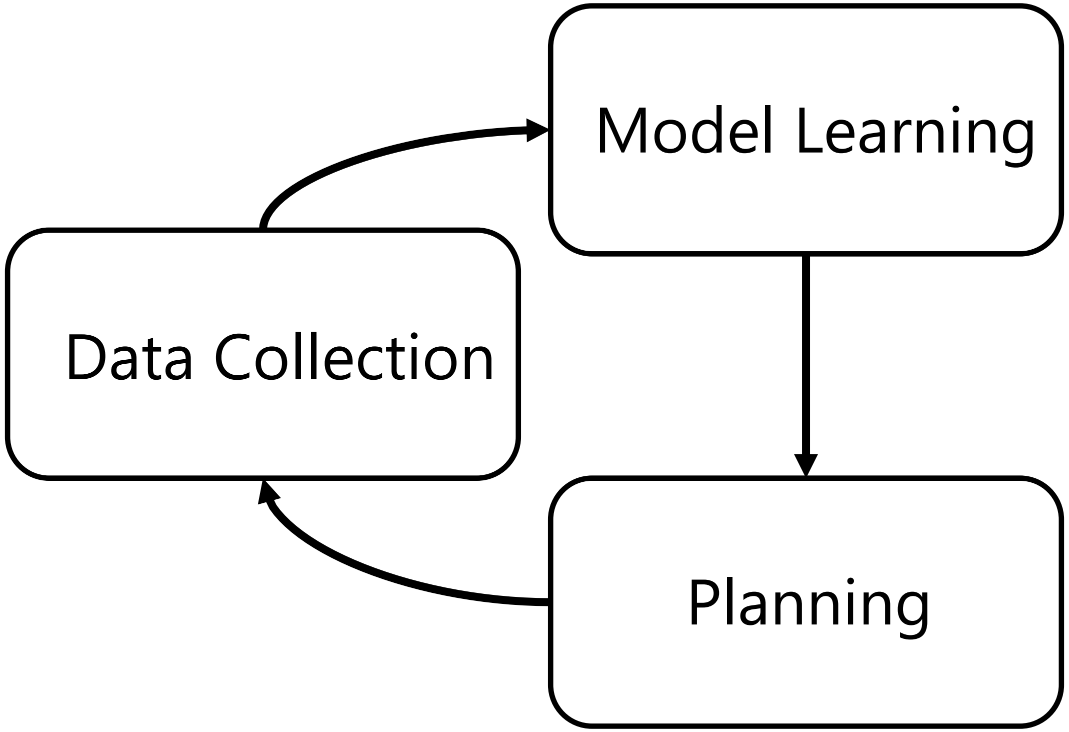
Replanning can help with divergence



Model-Predictive/Receding Horizon Control

1. Plan with random shooting from s_t
2. Execute the first action a_0 and reach s_{t+1}

Model Based RL – **Naïve Algorithm (v0)**

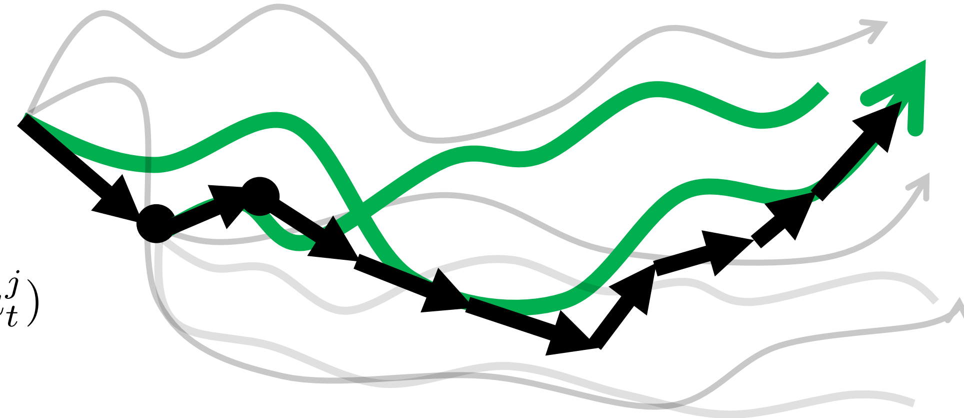


Maximum likelihood supervised Learnin

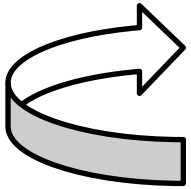
$$\max_{\theta} \mathbb{E}_{(s,a,s') \sim \mathcal{D}} \left[\log \hat{p}_\theta(s'|s,a)\right]$$

Data collected from planner

Planning with Shooting + MPC



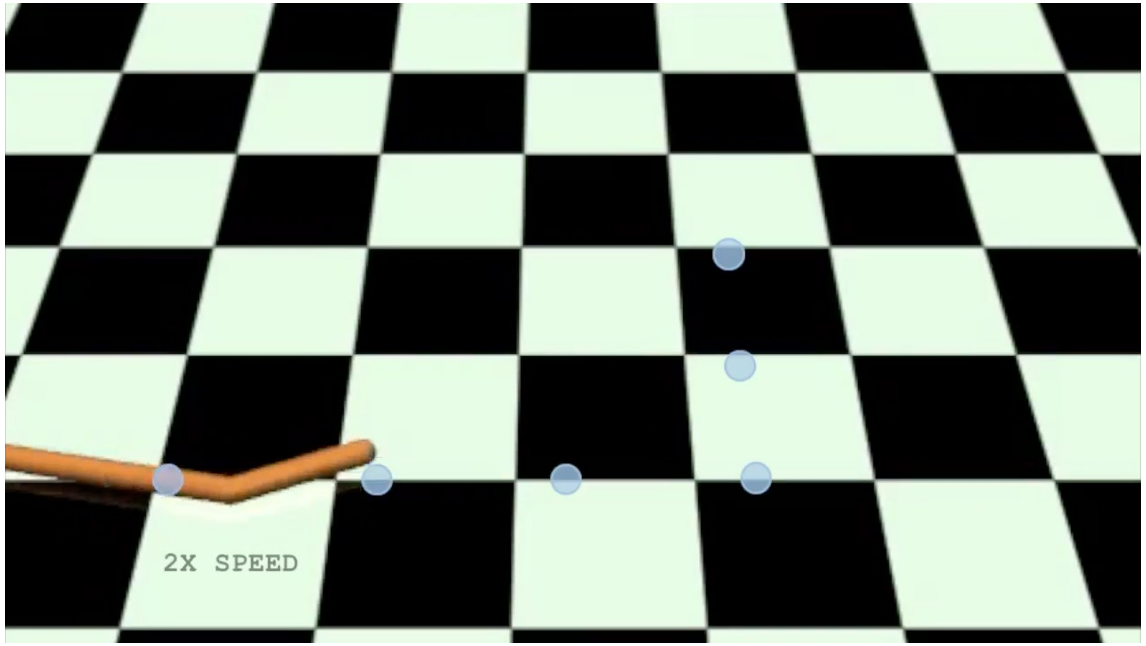
Better than open loop planning because of feedback



$$\arg \max_{a_0^j, a_1^j, \ldots, a_T^j} \sum_{t=0}^{T} r(\hat{s}_t^j, a_t^j)$$

$$\hat{s}_{t+1}^j \sim \hat{p}_\theta(.|\hat{s}_t^j, a_t^j)$$

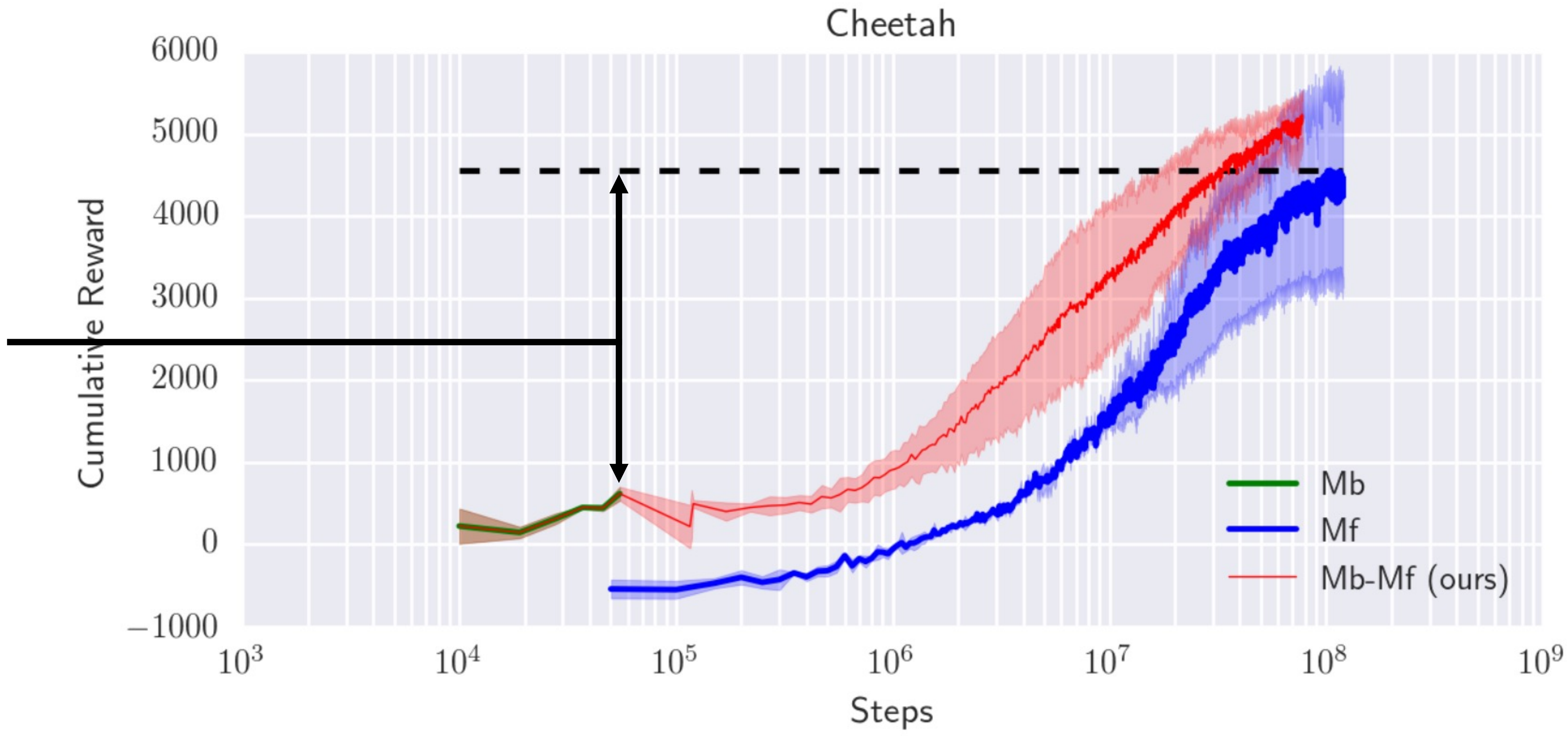
Does it work?



Just 20 minutes of training time with random data!

Does it work?

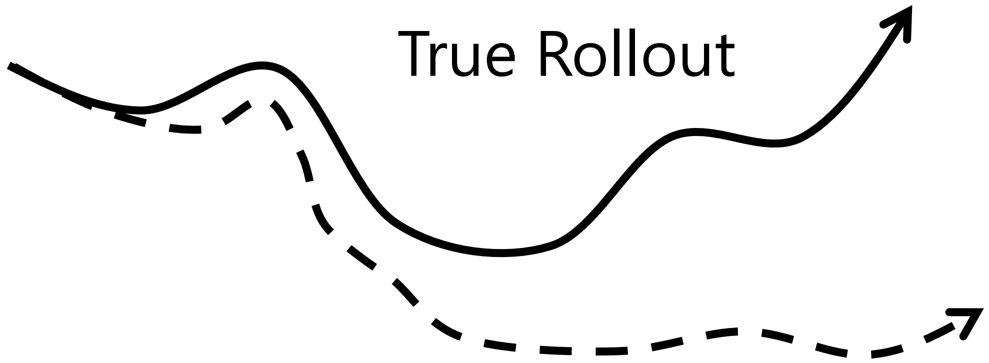
Significant gap from MFRL



What might be the issue?

Rollouts under learned model != Rollouts under true model

Model bias/compounding error



Why does this happen? → lack of data

1. Errors in state go to OOD next states
2. Deviations in actions go to OOD next states

Model is bad on OOD states!

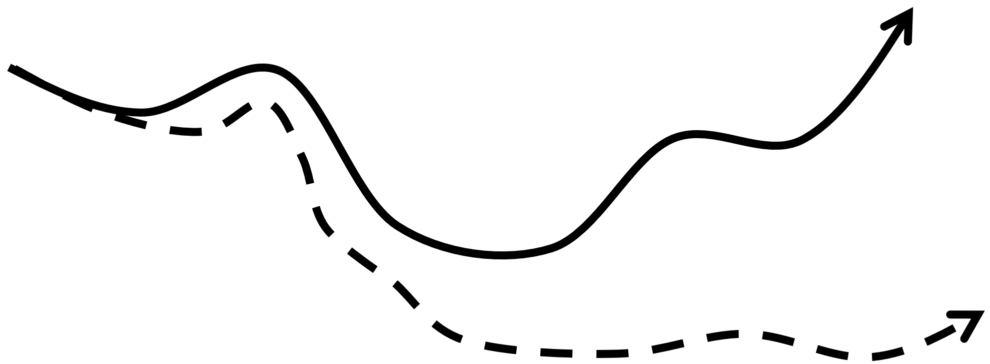
Most trained deep models can only roll out for 5-10 steps maximum!

How might we deal with compounding error?

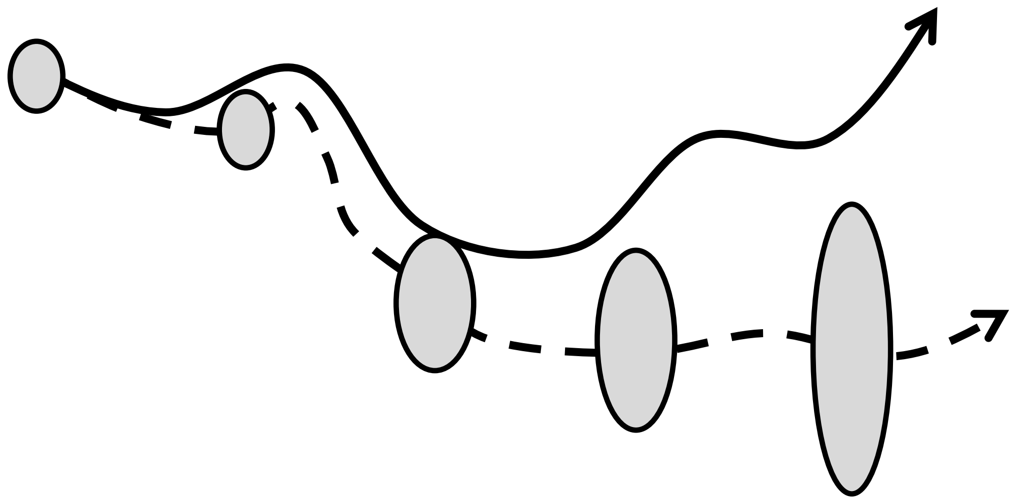
Idea: Estimate when OOD and account for it

→ Measure uncertainty!

Maximum likelihood models



Uncertainty-aware models



Being aware of uncertainty allows us to account for the effects of model bias!

What is uncertainty?

<u>Alleatoric Uncertainty</u>

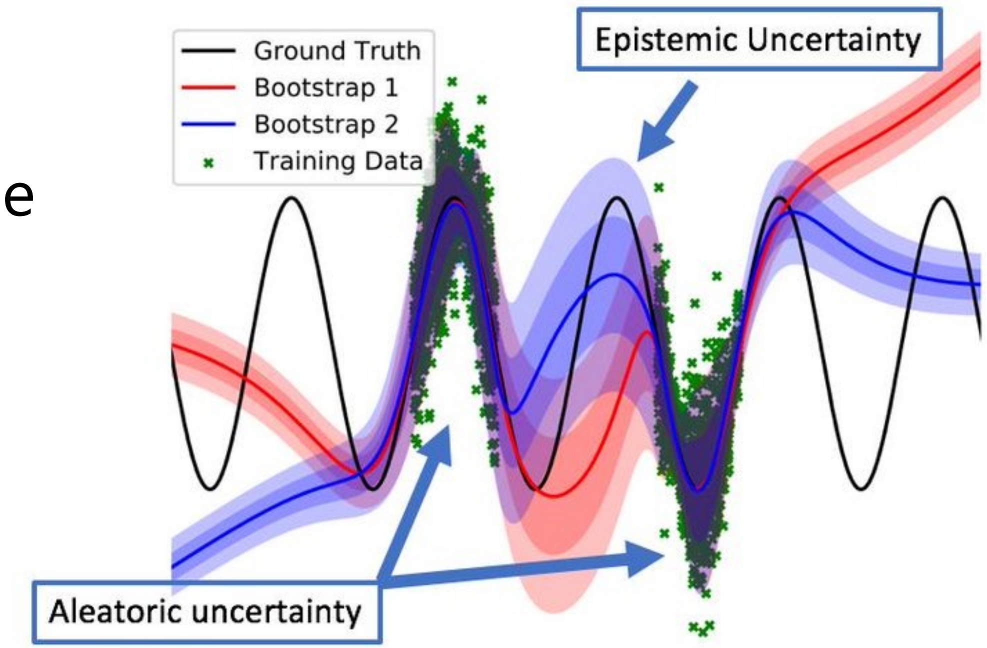
(environment stochasticity)

Easier, can use stochastic models

<u>Epistemic Uncertainty</u>

(Lack of data)

More challenging, need to compute posterior



Let's largely focus on epistemic uncertainty

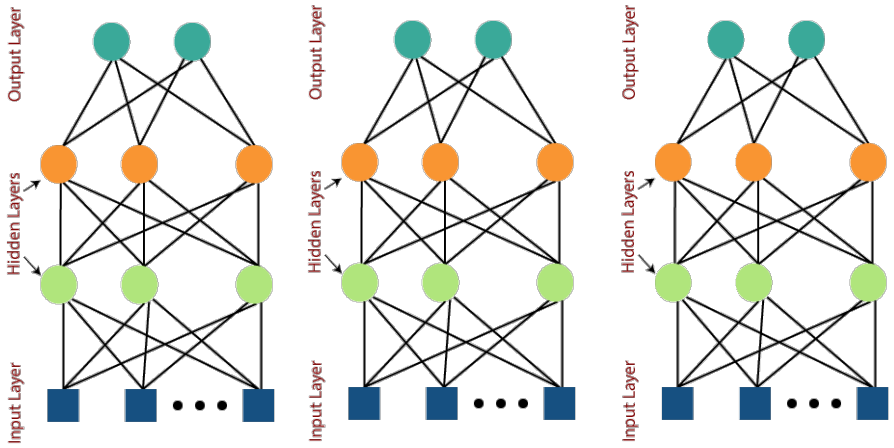
How might we measure uncertainty?

$$p(\theta|\mathcal{D})$$

Difficult to estimate directly!

Learn an ensemble of models

1. Bayesian neural networks
2. Ensemble methods
3. …

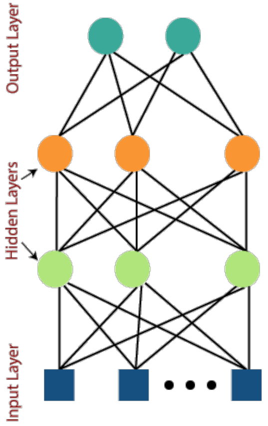


Low data regime → high ensemble variance

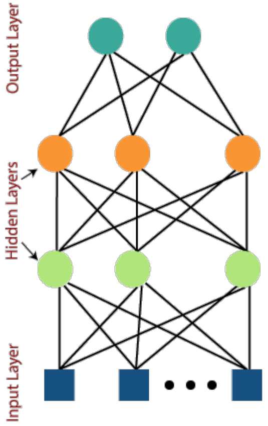
Approximate posterior

Model Based RL – Learning Ensembles of Dynamics Models

Learn ensembles of dynamics models with MLE rather than a single model

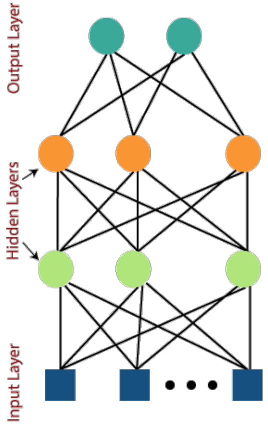


$$\max_\theta \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\left[\log \hat{p}_\theta(s'|s,a)\right]$$



$$\max_\theta \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\left[\log \hat{p}_\theta(s'|s,a)\right]$$

…

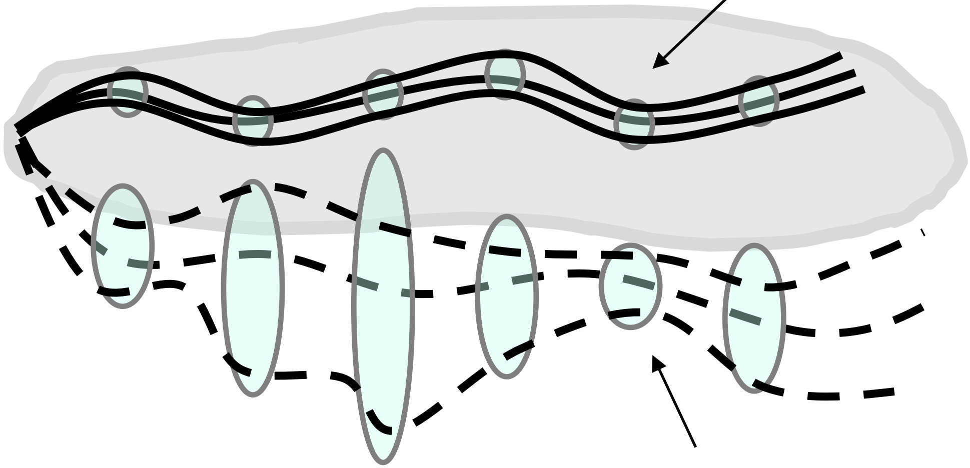


$$\max_\theta \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\left[\log \hat{p}_\theta(s'|s,a)\right]$$

Learn ensembles by either subsampling the data or having different initializations

Model Based RL – **Integrating Uncertainty into MBRL (v2)**

Take expected value under the uncertain dynamics



Expected value over ensemble

$$\arg \max_{(a_0^j, a_1^j, \ldots, a_T^j)_{j=1}^N} \sum_{i=1}^{K} \sum_{t=0}^{T} r((\hat{s}_t^j)^i, a_t^j)$$

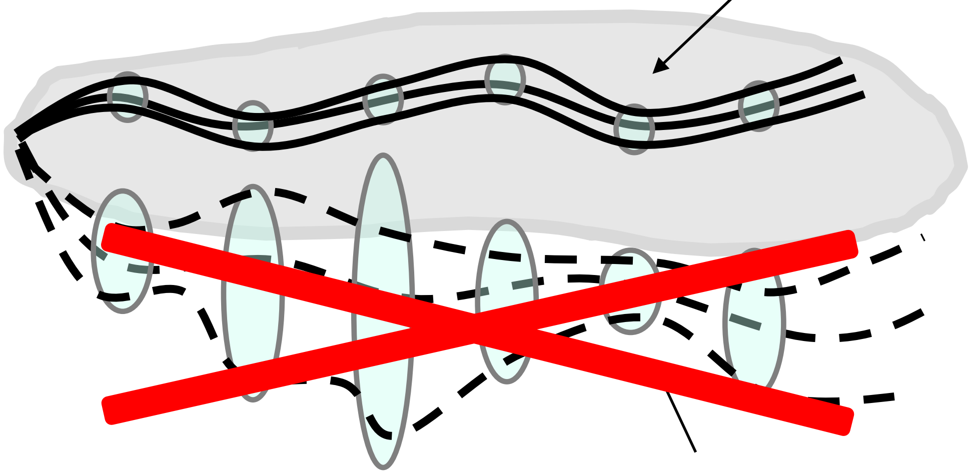
$$(\hat{s}_{t+1}^j)^i \sim \hat{p}_{\theta_i}(.|(\hat{s}_t^j)^i, a_t^j)$$

Can also swap which ensemble element is propagated at every step or just pick randomly amongst them

Avoids overly OOD settings since the expected reward is affected by uncertainty

Model Based RL – Integrating Uncertainty into MBRL (v2)

Take **pessimistic** value under the uncertain dynamics



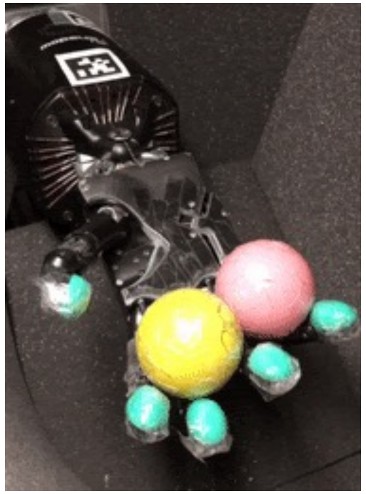
Penalize ensemble variance

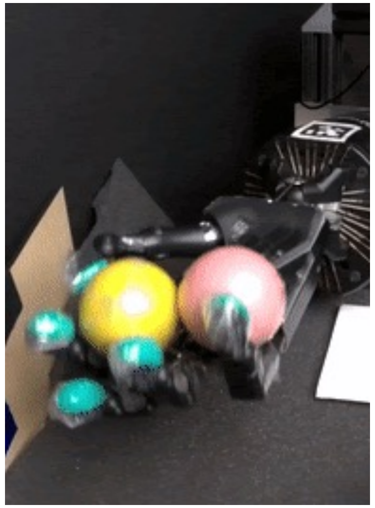
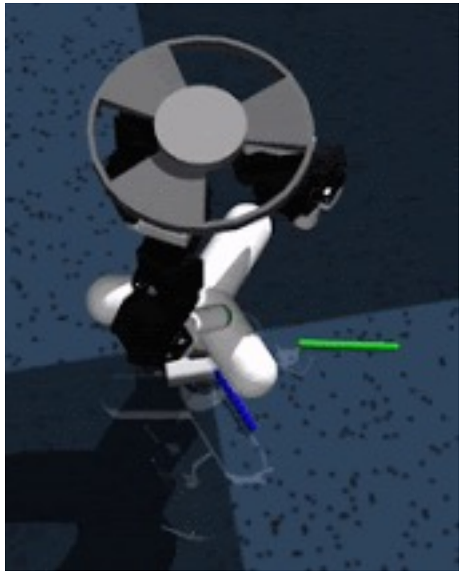
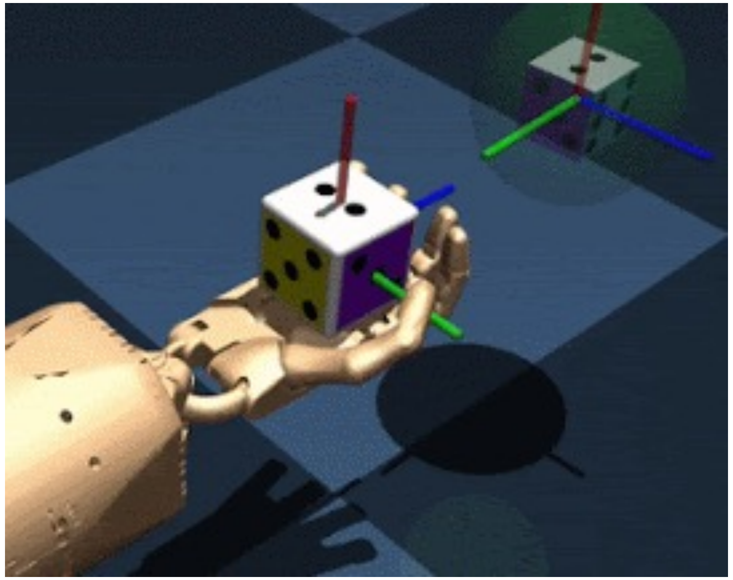
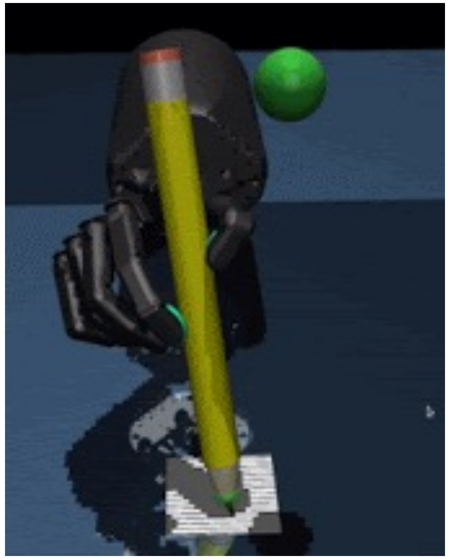
$$\arg \max_{(a_0^j, a_1^j, \ldots, a_T^j)_{j=1}^N} \sum_{i=1}^{K} \sum_{t=0}^{T} r((\hat{s}_t^j)^i, a_t^j) - \lambda \mathrm{Var}((\hat{s}_t^j)^i)$$

$$(\hat{s}_{t+1}^j)^i \sim \hat{p}_{\theta_i}(.|(\hat{s}_t^j)^i, a_t^j)$$

Avoids overly OOD settings since these states are explicitly penalized

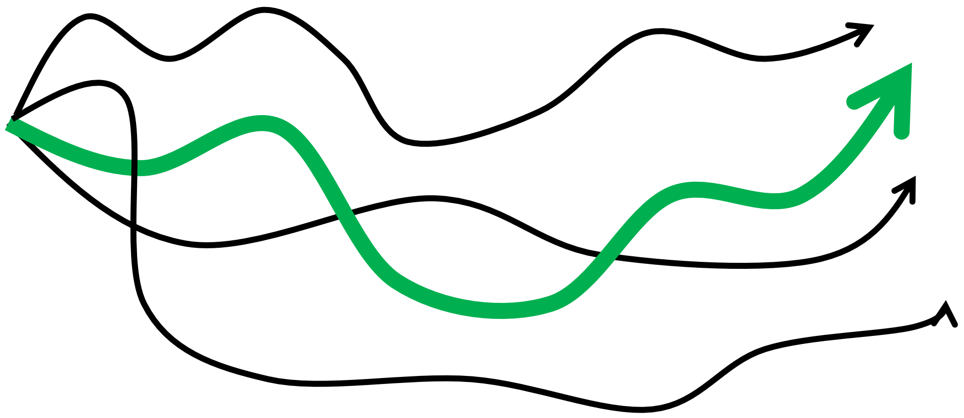
Does this work?



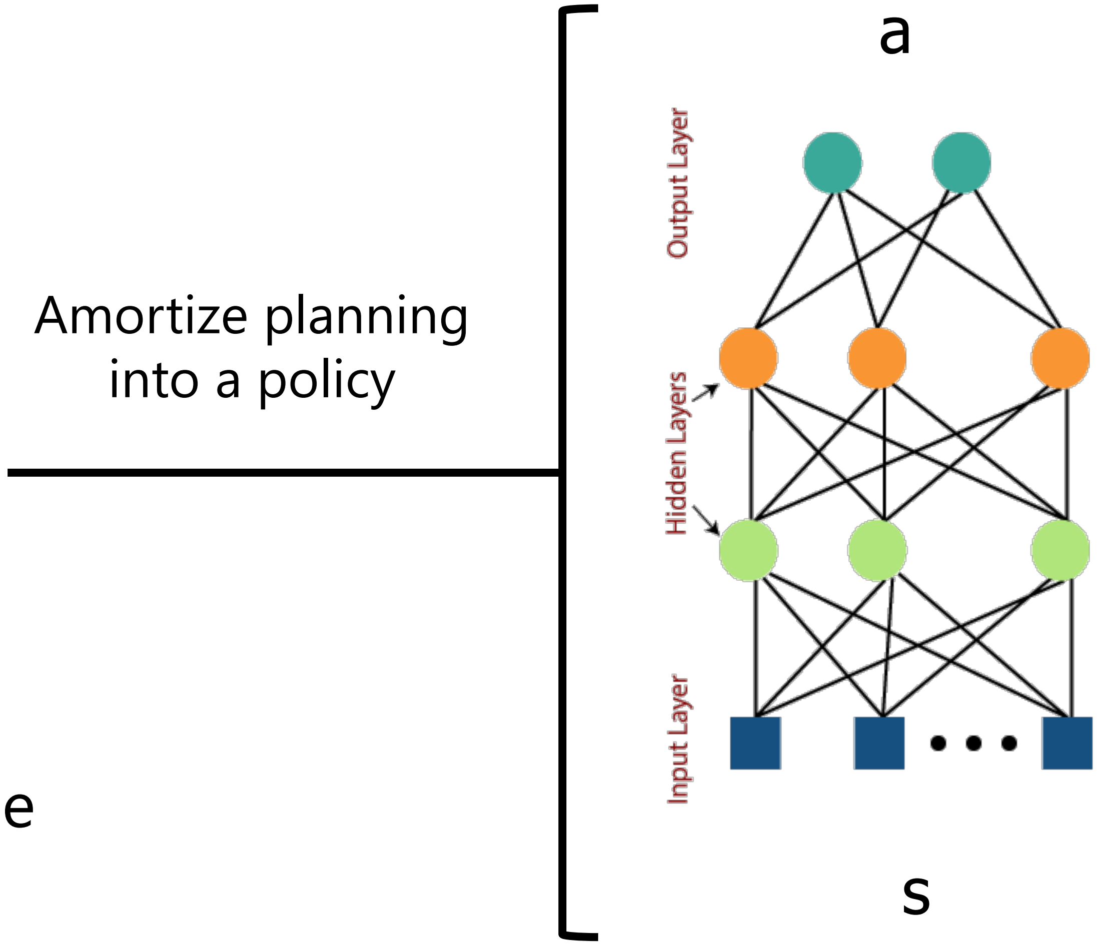
What might be the issue?

Huge number of samples needed to reduce variance



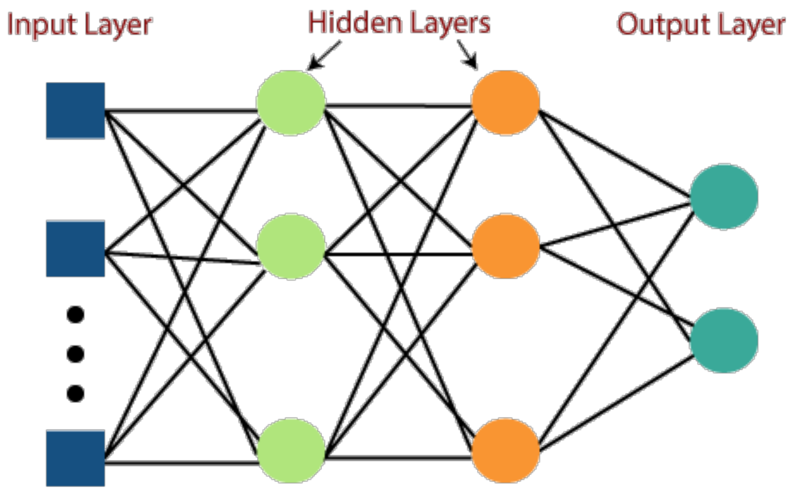
Extremely slow, hard to run in real time

Amortize planning into a policy

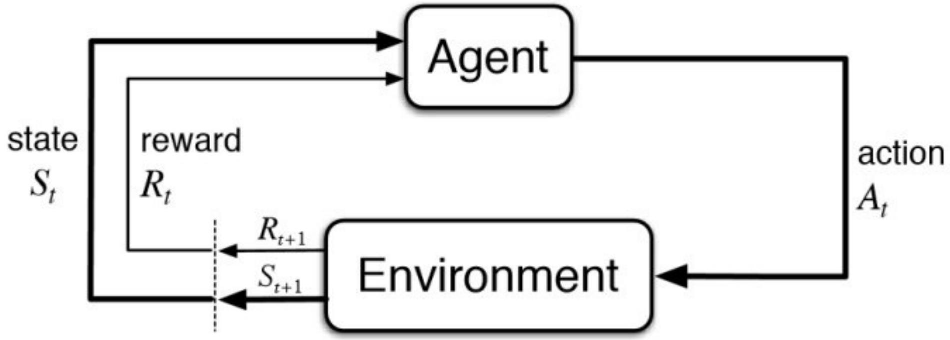


Speeding Up Model-Based Planning

$$\max_{\theta} \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\left[\log \hat{p}_{\theta}(s'|s,a)\right]$$

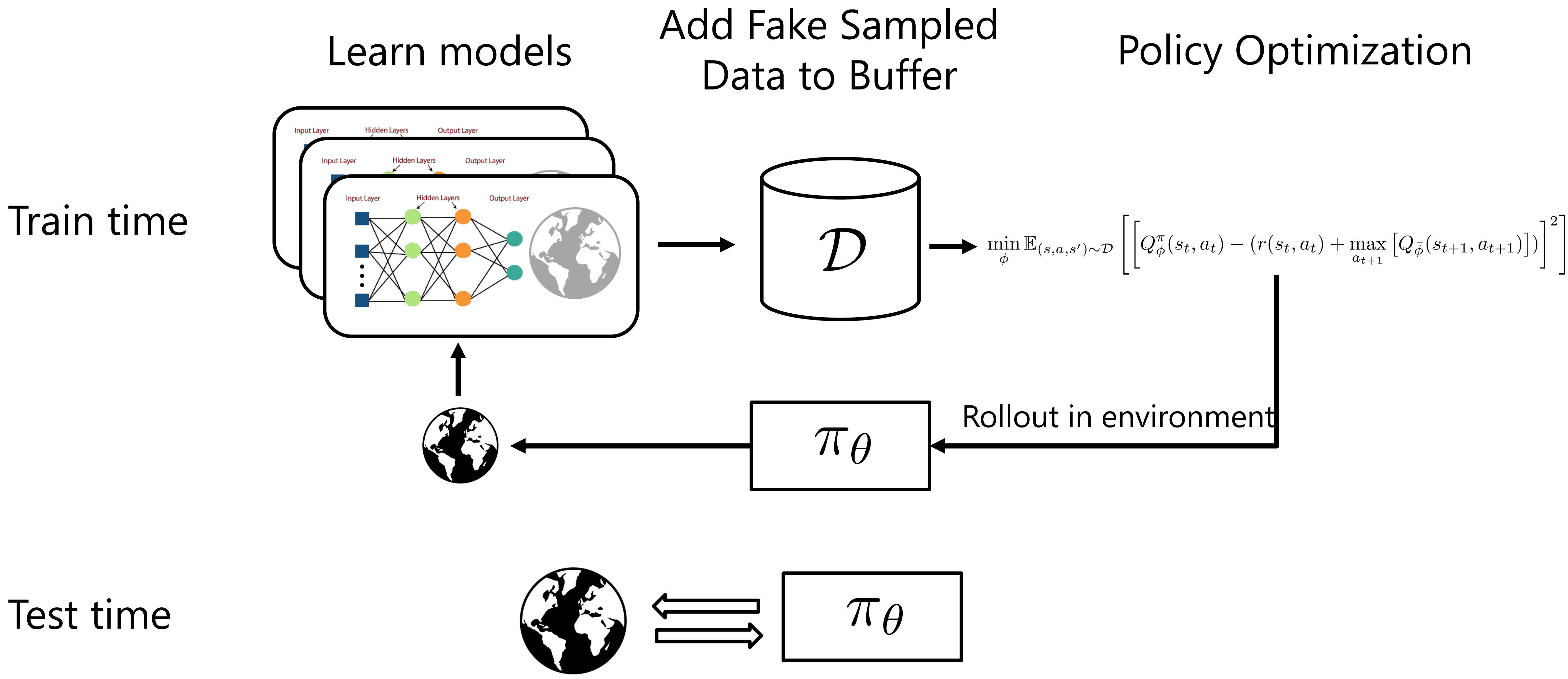


Use model(s) to generate data for policy optimization

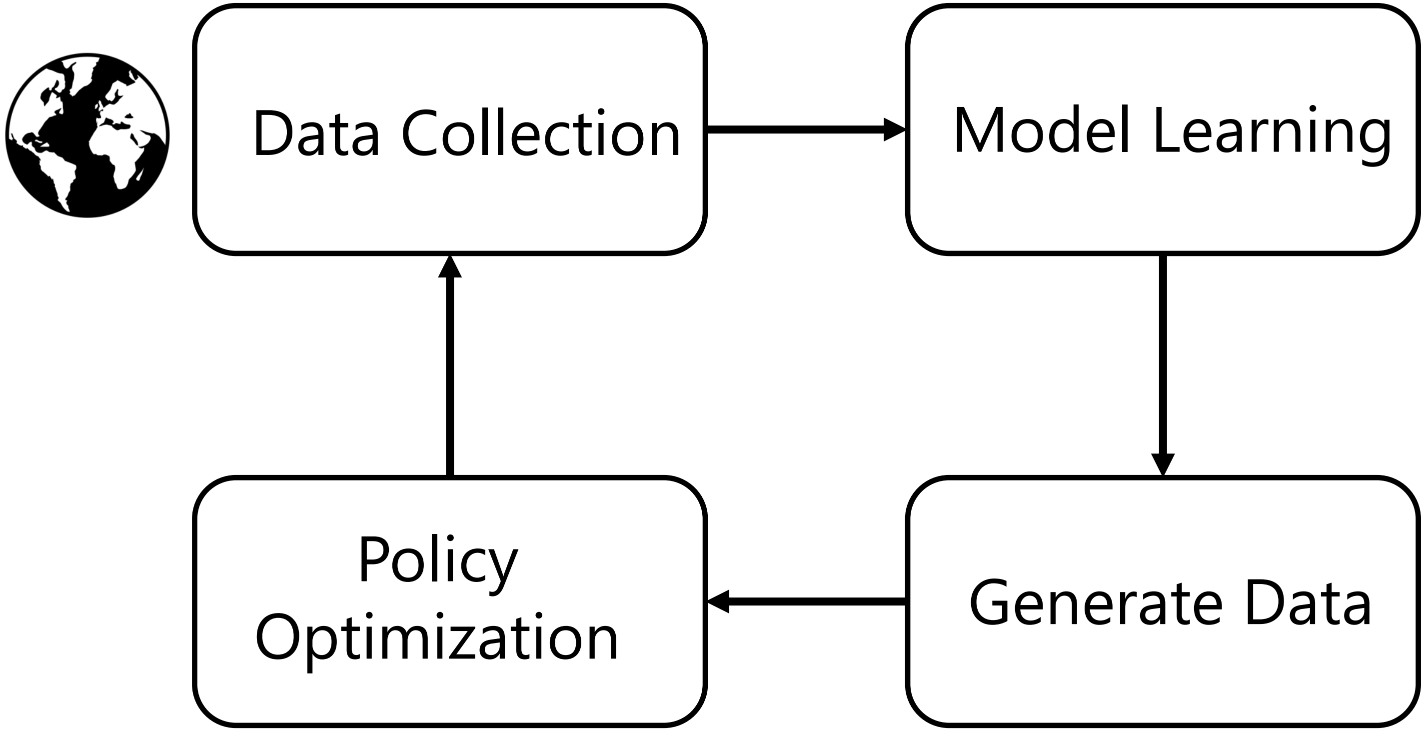


Can use PG or off-policy!

Generating Data for Policy Optimization

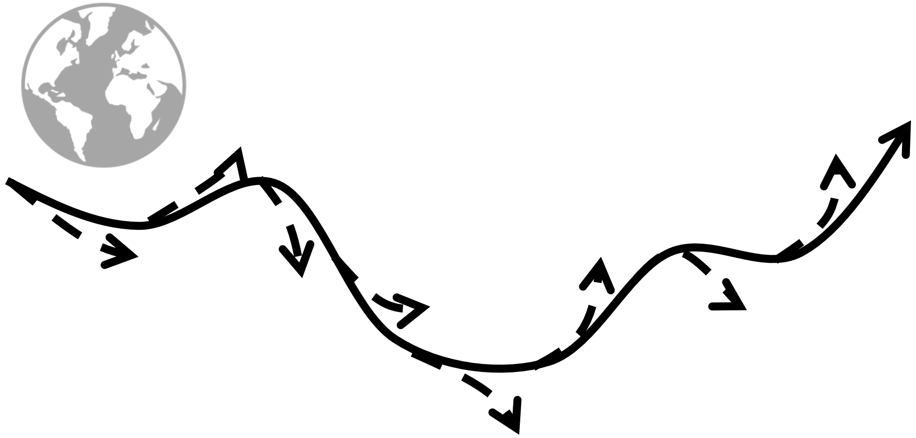


Model Based RL – Using Models for Policy Optimization (v3)



Maximum likelihood supervised Learning

$$\max_{\theta} \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \left[\log \hat{p}_\theta(s'|s,a)\right]$$



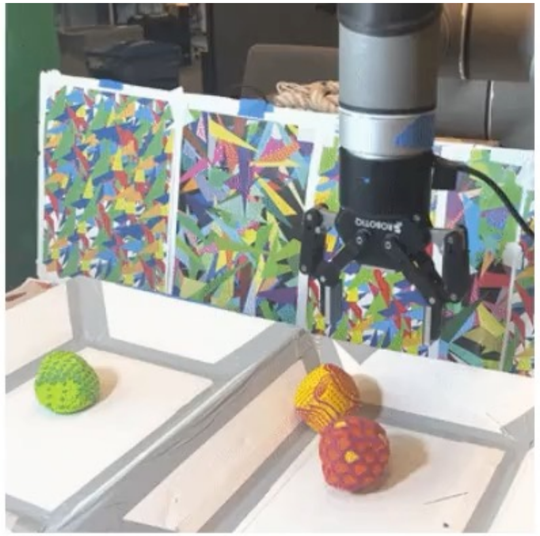
$$\min_{\phi} \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \left[\left[Q^{\pi}_{\phi}(s_t, a_t) - \left(r(s_t, a_t) + \max_{a_{t+1}} \left[Q_{\bar{\phi}}(s_{t+1}, a_{t+1})\right]\right)\right]^2\right]$$

More expensive/harder at training time, faster at test time

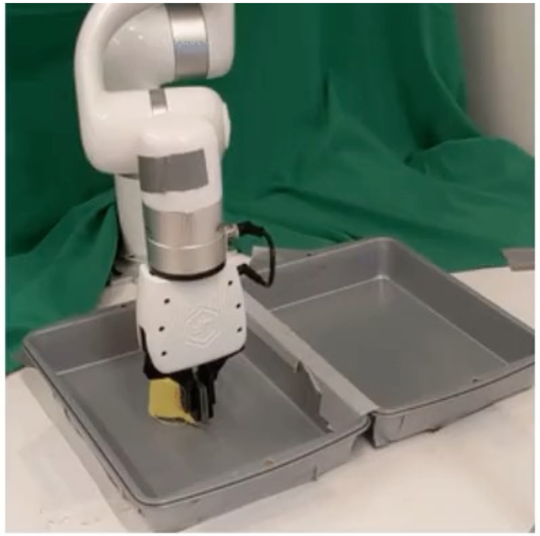
Does this work?

Does this work?

A1 Quadruped Walking



UR5 Multi-Object Visual Pick Place



XArm Visual Pick and Place

Sphero Ollie Visual Navigation

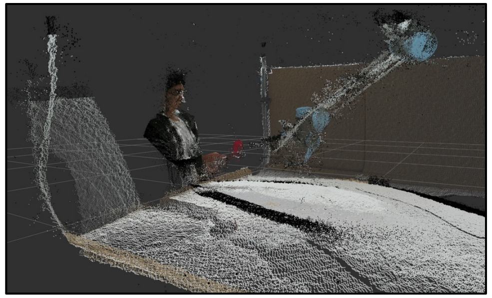
Course Overview

- Filtering/Smoothing
- Localization
- Mapping
- SLAM

- Search
- Motion Planning
- TrajOpt
- Stability/Certification

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

What we covered in this class – modular robotics pipelines



State Estimation → Modeling and Prediction → High-level planning / Low-level planning → Low-level control →

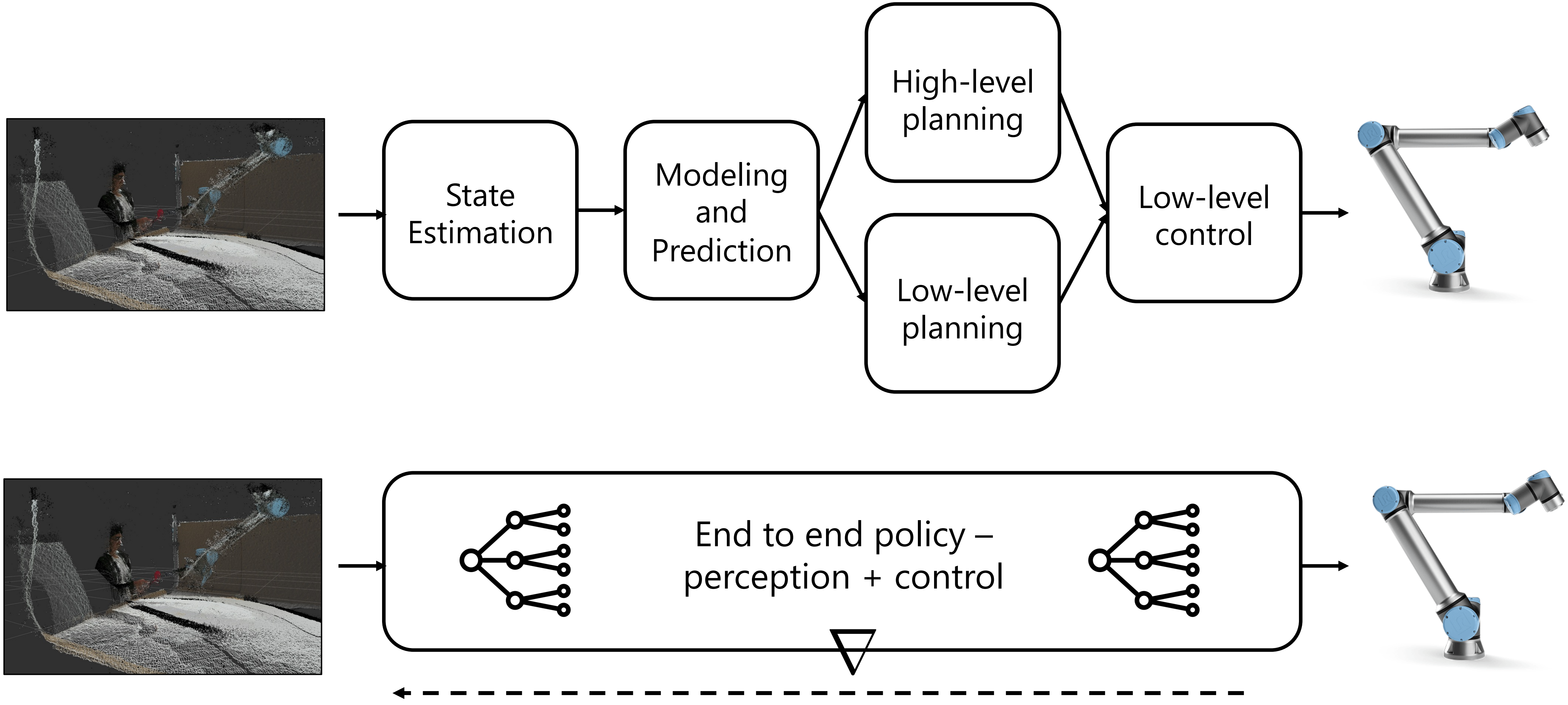
EKF/UKF/Particle Filter

SLAM

Motion Planning

LQR/PID

What we covered in this class – end to end RL/IL



Policy Gradient **Actor Critic** **Model-Based RL** **Imitation Learning**

Thank you!