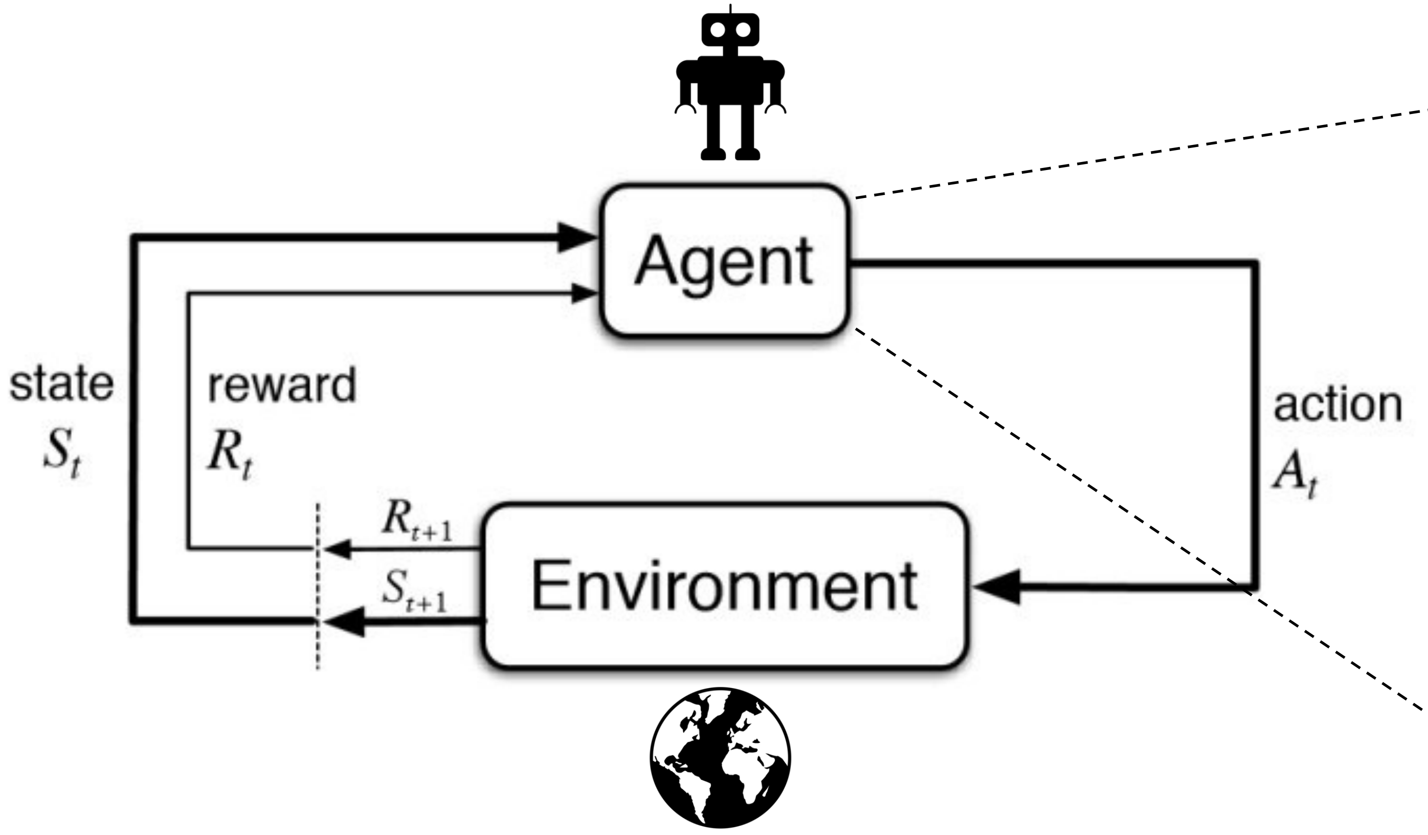
Autonomous Robotics
Winter 2026

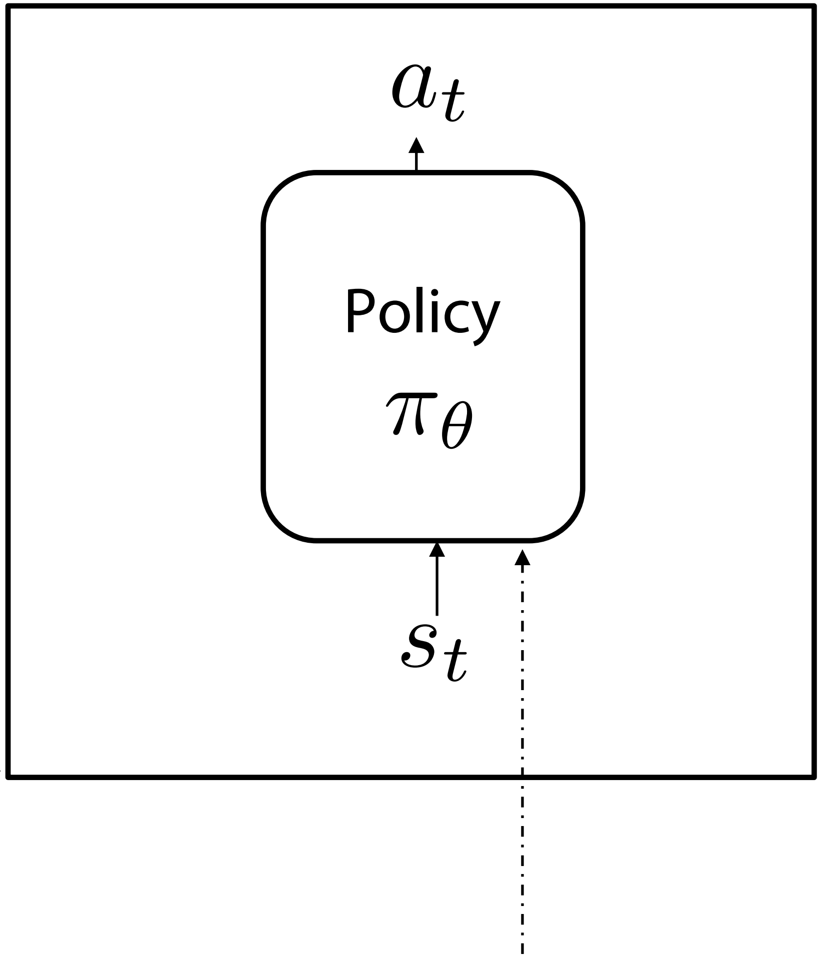
Abhishek Gupta, Siddhartha Srinivasa
TAs: Carolina Higuera, Entong Su, Rishabh Jain

Recap

Reinforcement Learning Formalism



Rules for choosing actions



Needs to be learned

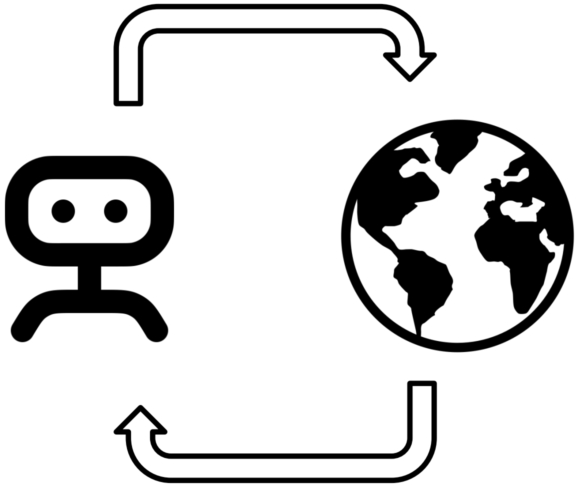
$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} r(s_t, a_t) \right]$$

Trajectory sampled using policy

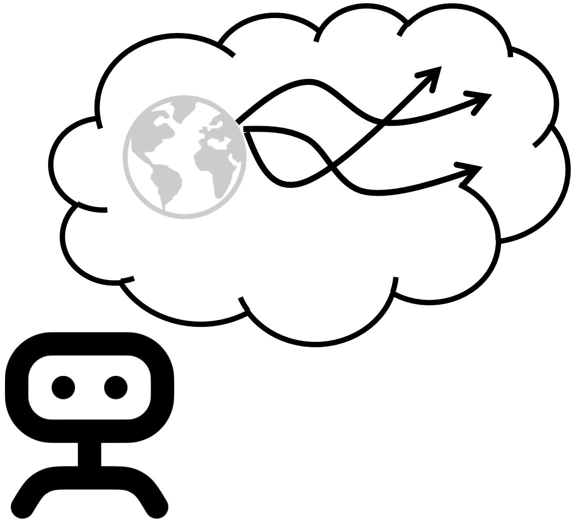
Ok so how can we learn policies?

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

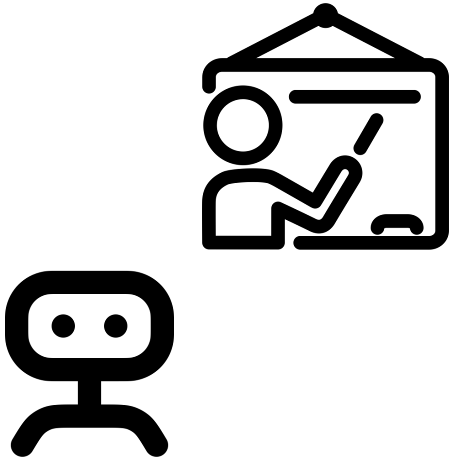
Model-free RL



Model-based RL



Imitation Learning



What if we just performed gradient ascent?

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

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

<u>Standard gradient descent (supervised learning)</u>

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

<u>REINFORCE gradient descent (RL)</u>

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

Gradient wrt expectation variable, not of integrand!

Taking the gradient of sum of rewards

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

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

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

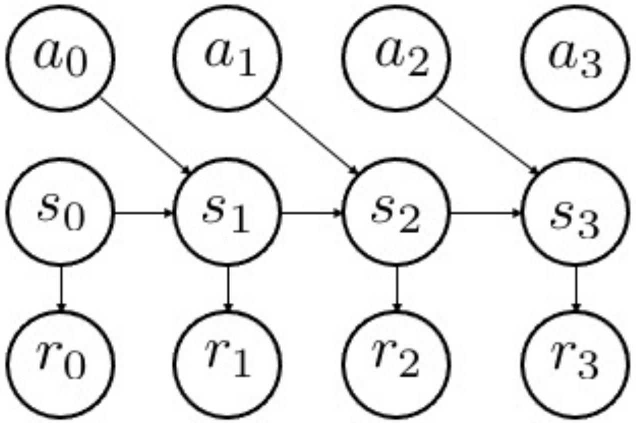
$$= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) R(\tau) d(\tau) \quad = \mathbb{E}_{p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) R(\tau) \right]$$

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

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

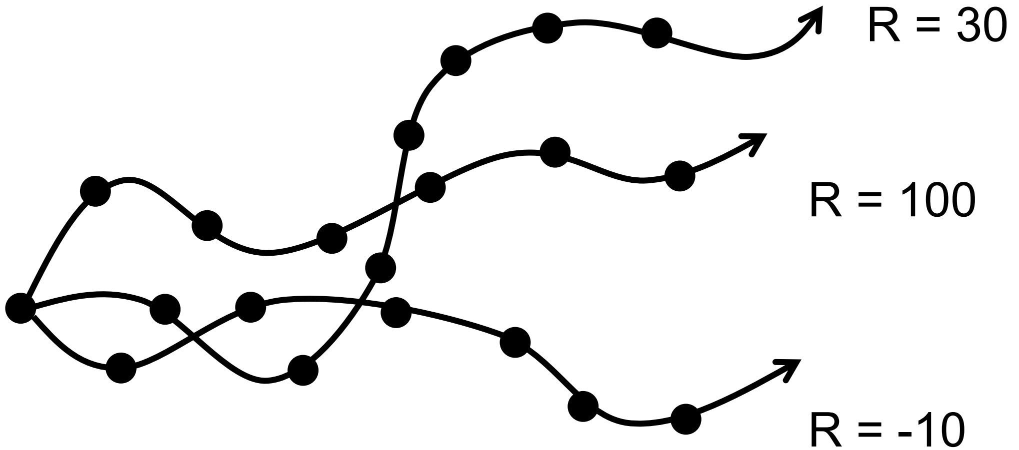
$$\nabla_\theta J(\theta) = \mathbb{E}_{\substack{s_0 \sim p(s_0) \\ s_{t+1} \sim p(s_{t+1}|s_t, a_t) \\ a_t \sim \pi(a_t|s_t)}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=0}^{T} r(s_t, a_t) \right]$$

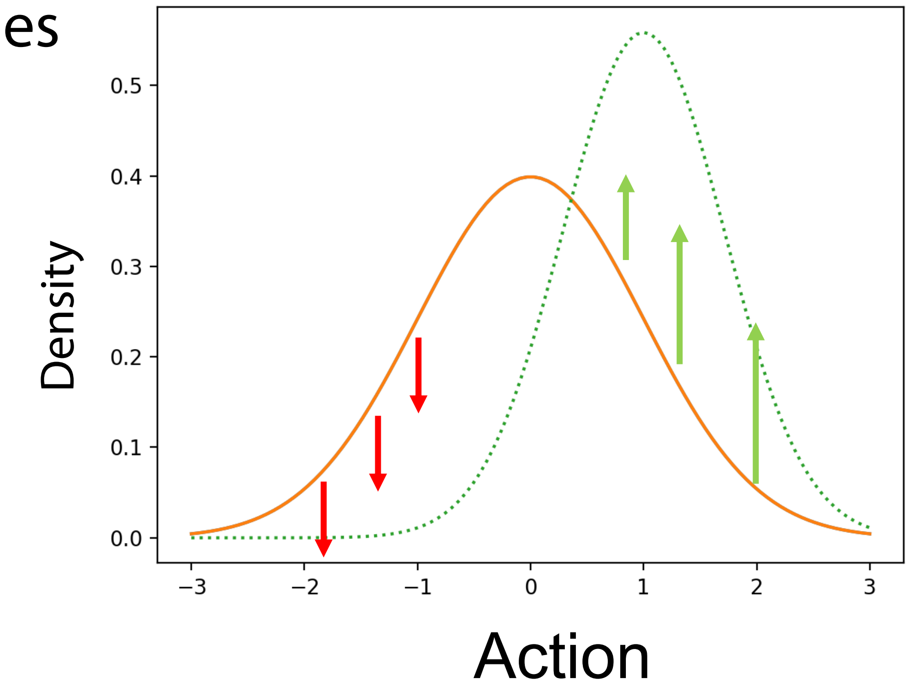
$$\approx \frac{1}{N} \sum_{i=0}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \sum_{t'=0}^{T} r(s_{t'}^i, a_{t'}^i)$$ (approximating using samples)

What does this mean?

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

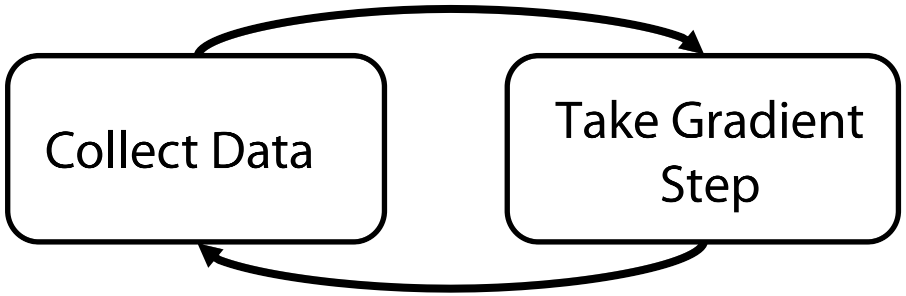
Increase the likelihood of actions in high return trajectories





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

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>

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

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

PG = select good data + increase likelihood of selected data

Lecture Outline

Recap

↓

Improving Policy Gradient

↓

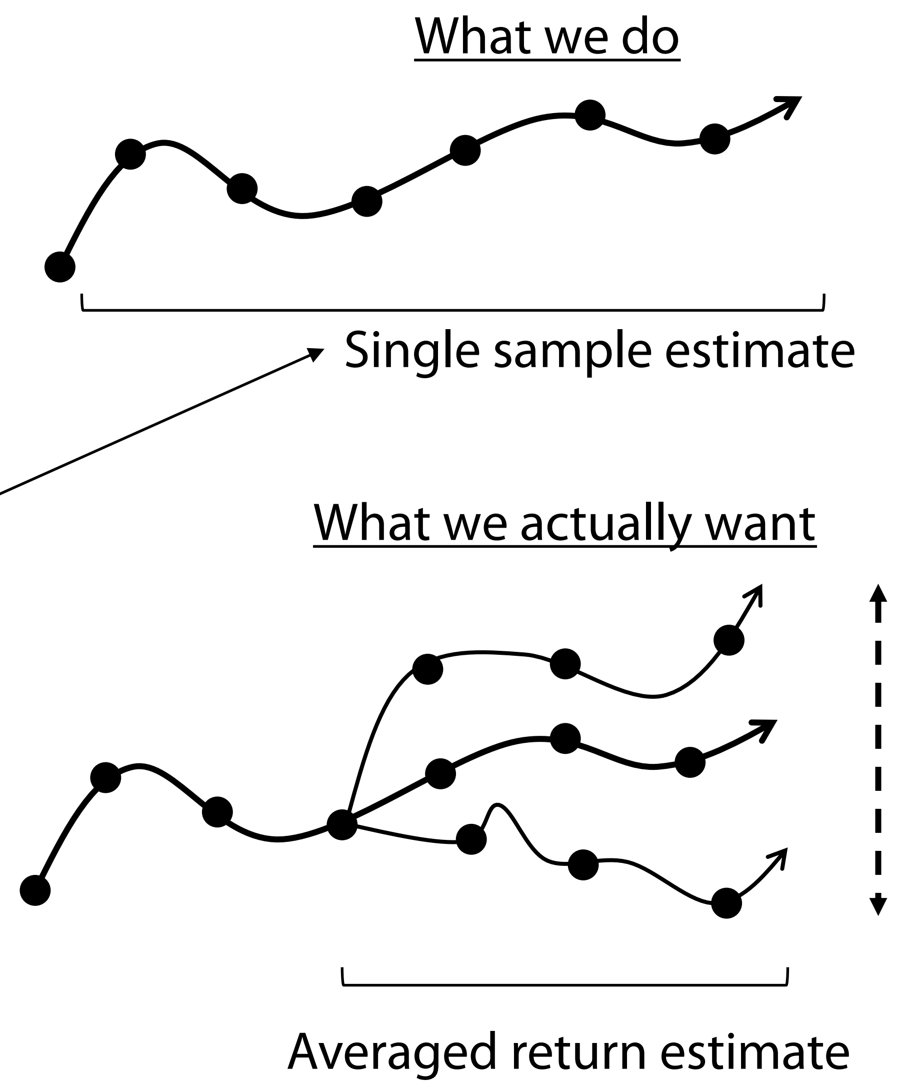
Model-based RL

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!!



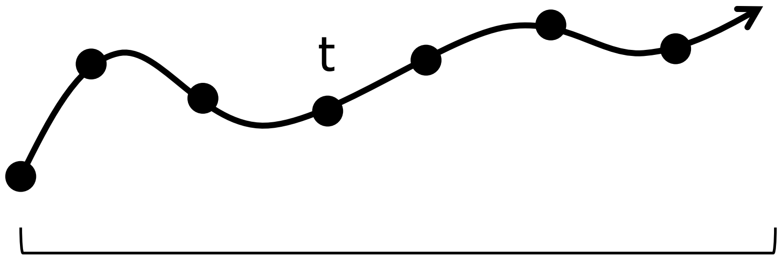
Hard to tell what matters without many samples

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"**

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

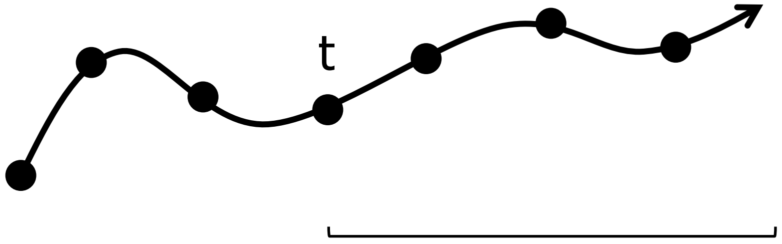
Includes t' < t



Full trajectory return

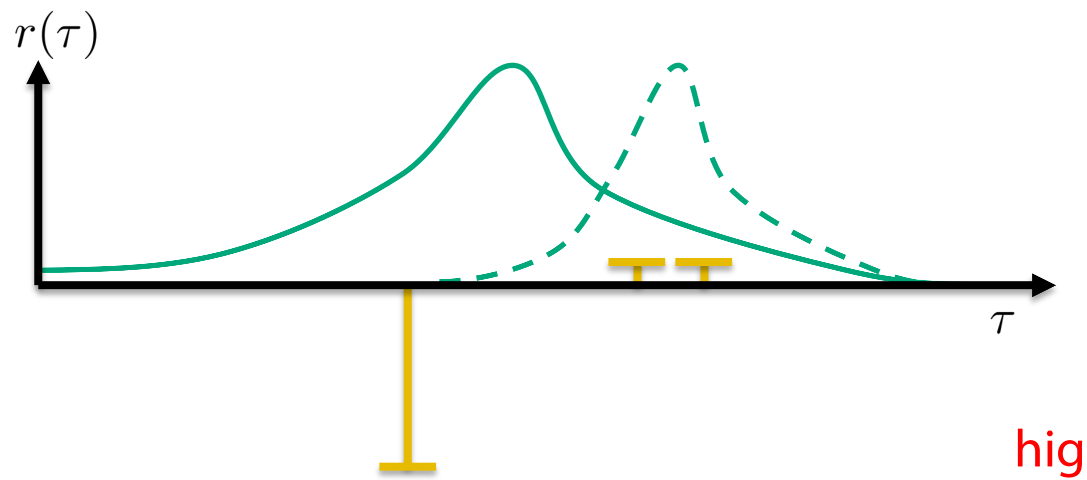
Ignore past terms

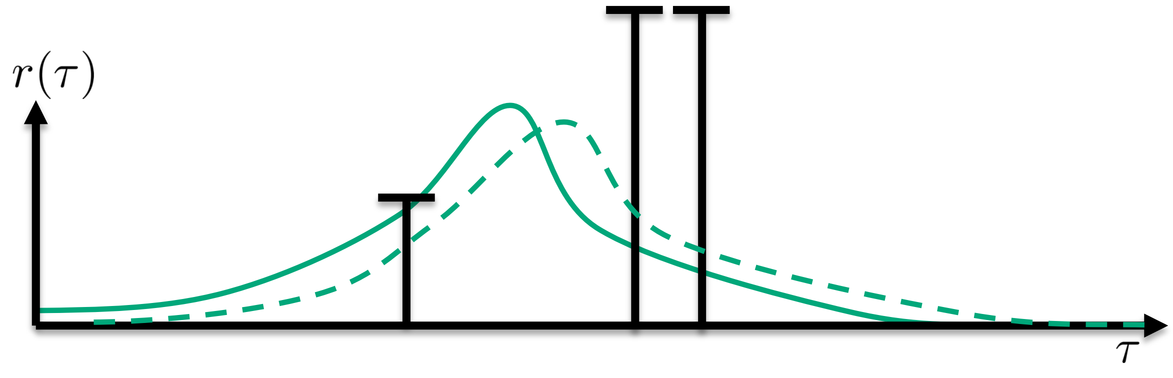
$$\frac{1}{N}\sum_{i=0}^{N}\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\sum_{t'=t}^{T} r(s_t^i, a_t^i)$$



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 positive)

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

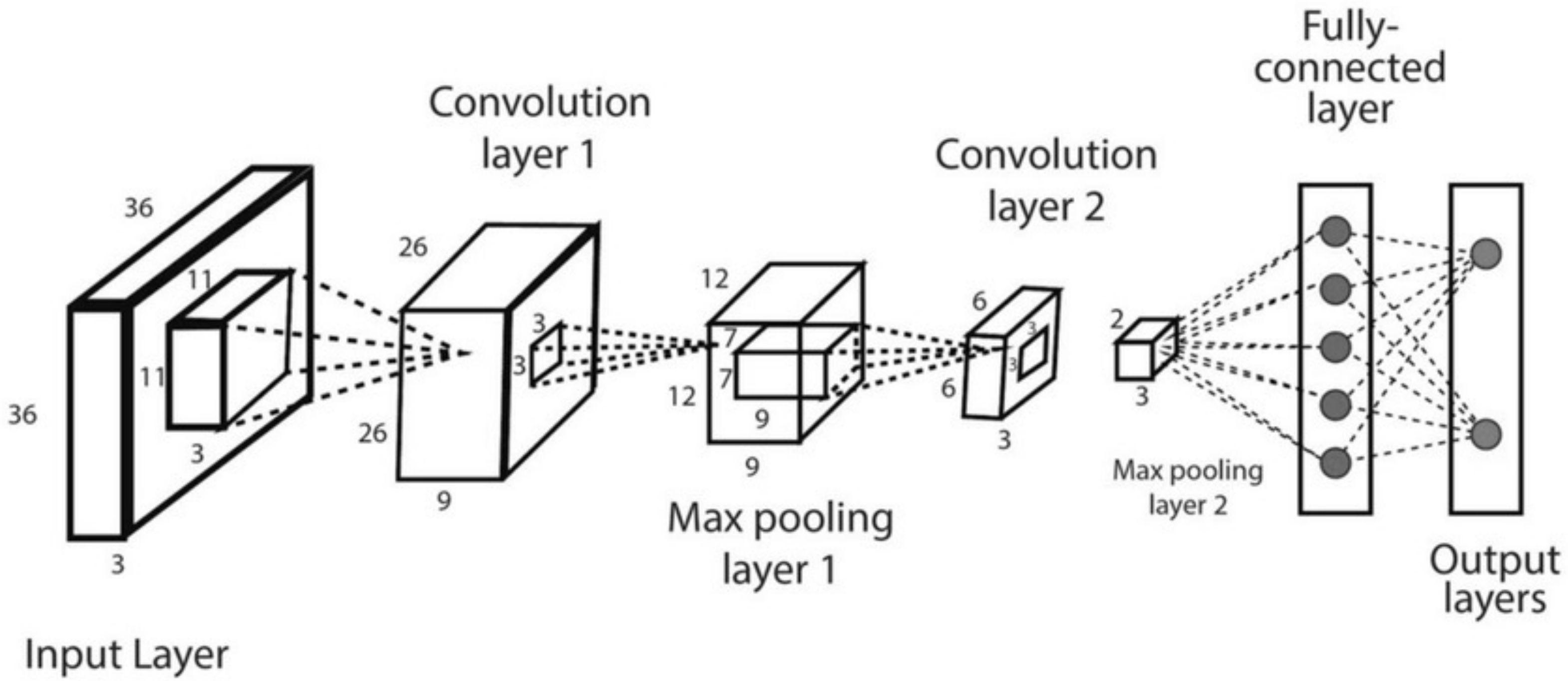
$$\frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\left[\sum_{t'=t}^{T} r(s_{t'}^i, a_{t'}^i) - b(s_t)\right]$$

Baseline: Centers the returns, reduces variance

We can show this is unbiased!

Learning Baselines

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



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

Average MC return under latest policy

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

Allows us to define advantages

Further Improvements on Policy Gradient

Control Step Size

Prevent excessive step size

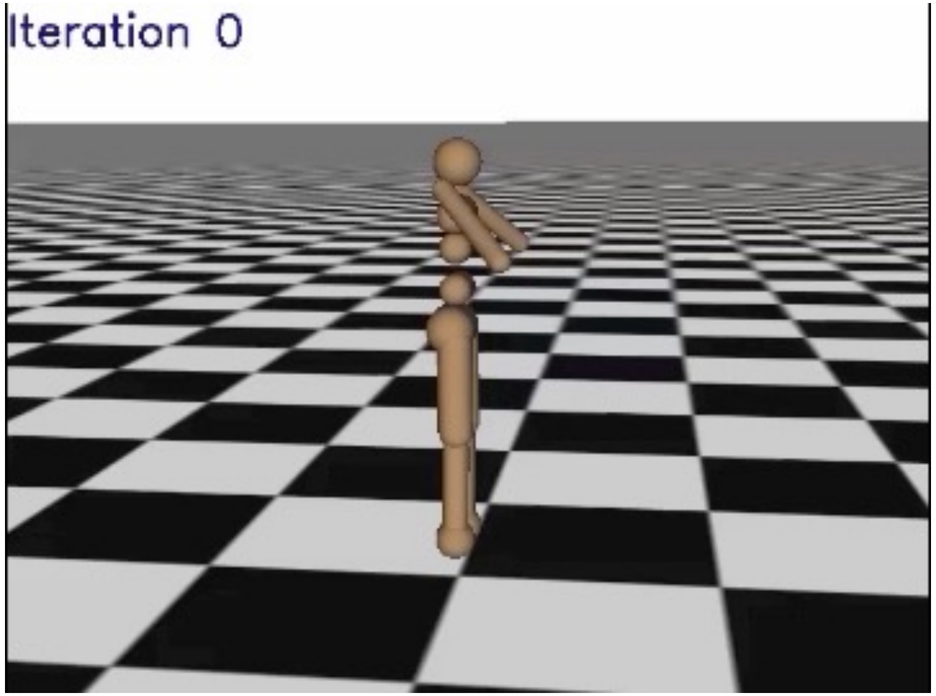
Proximal Policy Optimization

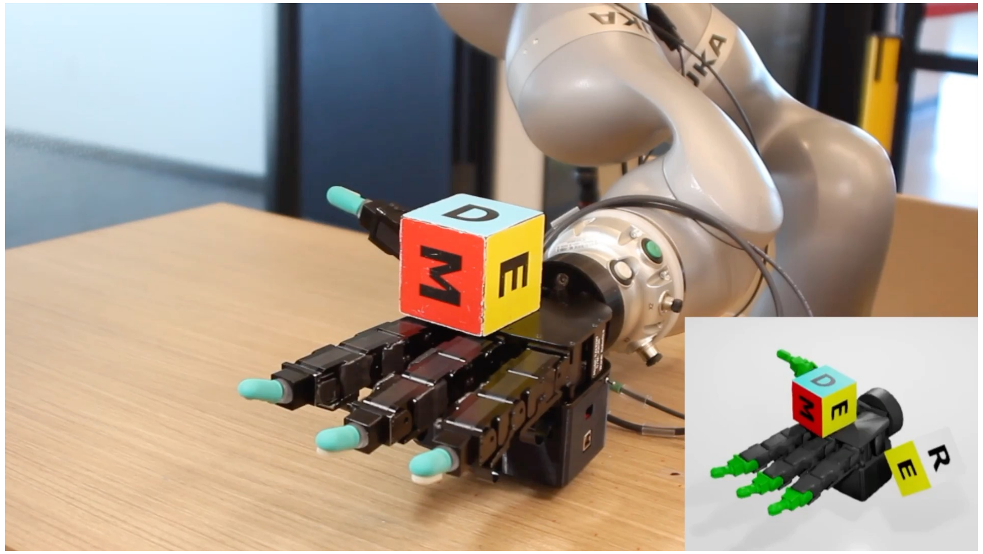
$$\mathcal{L}(s, a, \theta_i, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_i}(a|s)} A(s,a), \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_i}(a|s)}, 1-\epsilon, 1+\epsilon\right) A(s,a)\right)$$

Don’t let the policy change too much

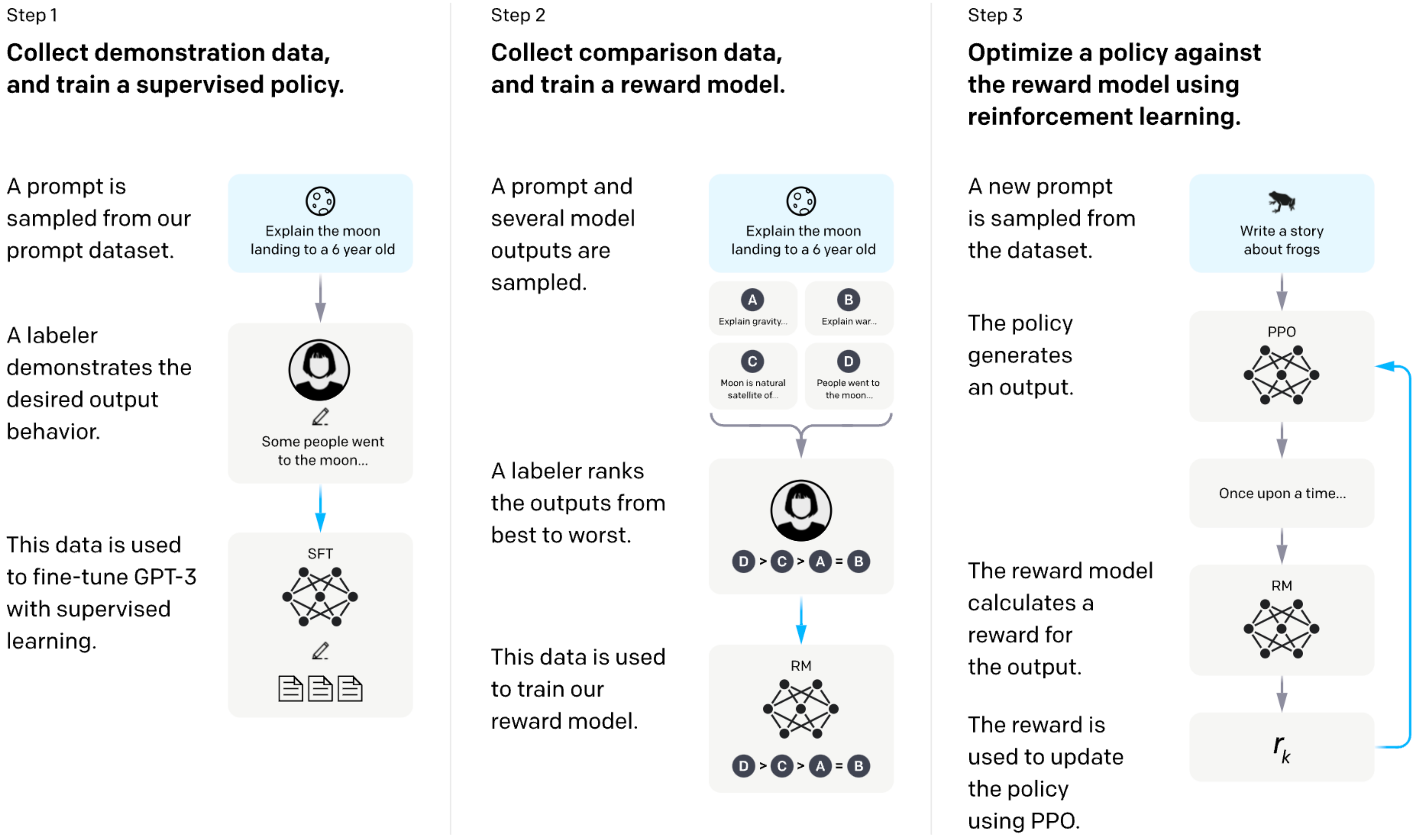
This allows for more gradient steps and stable updates

Advanced Policy Gradient in Action: Sim Control





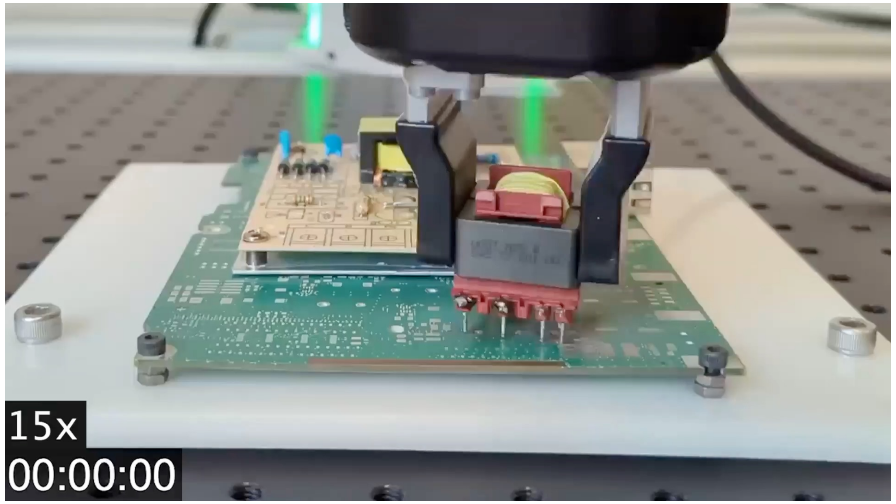
Advanced Policy Gradient in Action: LLMs



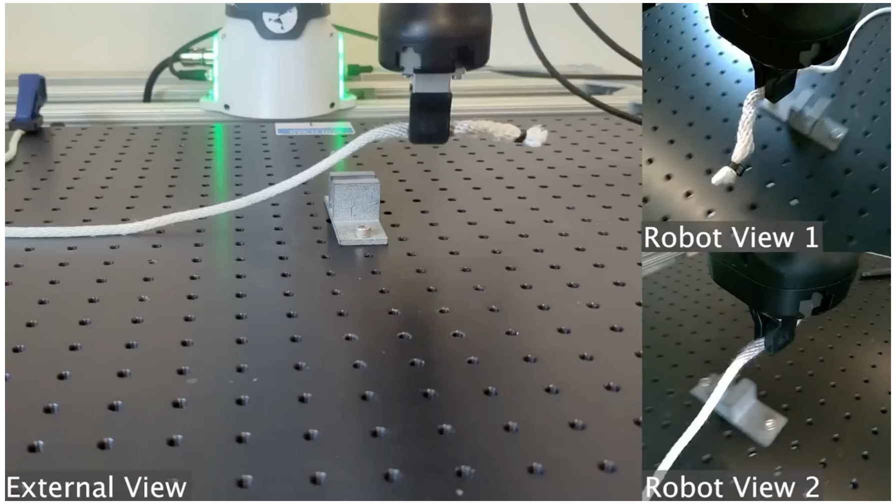
Model-Free RL in Action: Real-World Robots

With improvements in return estimation - can work on robots!

Smith et al



Luo et al



Luo et al

Lecture Outline

Recap

↓

Improving Policy Gradient

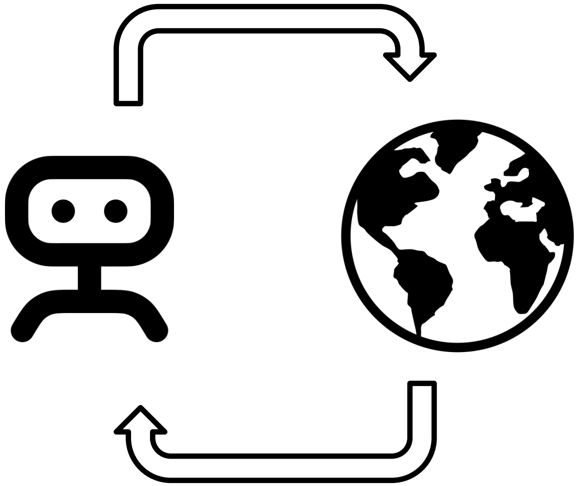
↓

Model-based RL

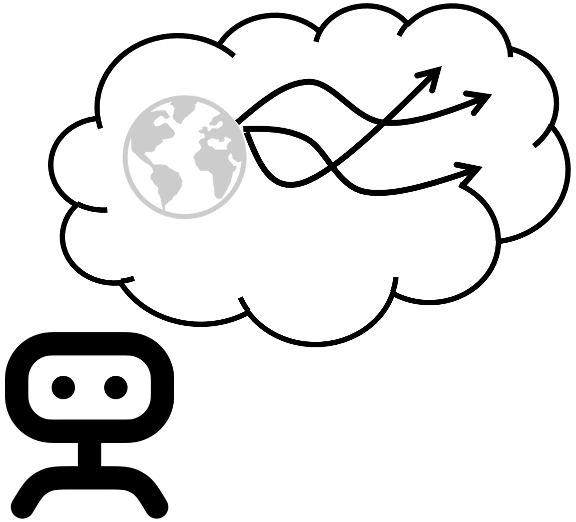
Ok so how can we learn policies?

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

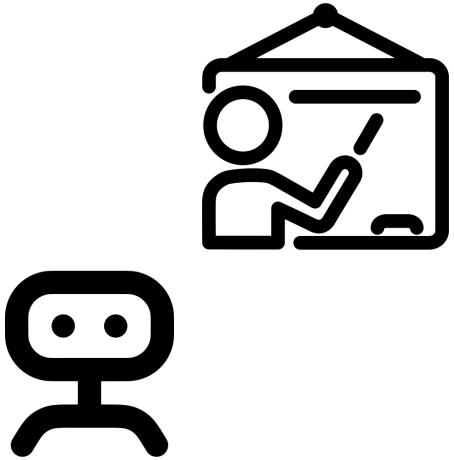
Model-free RL



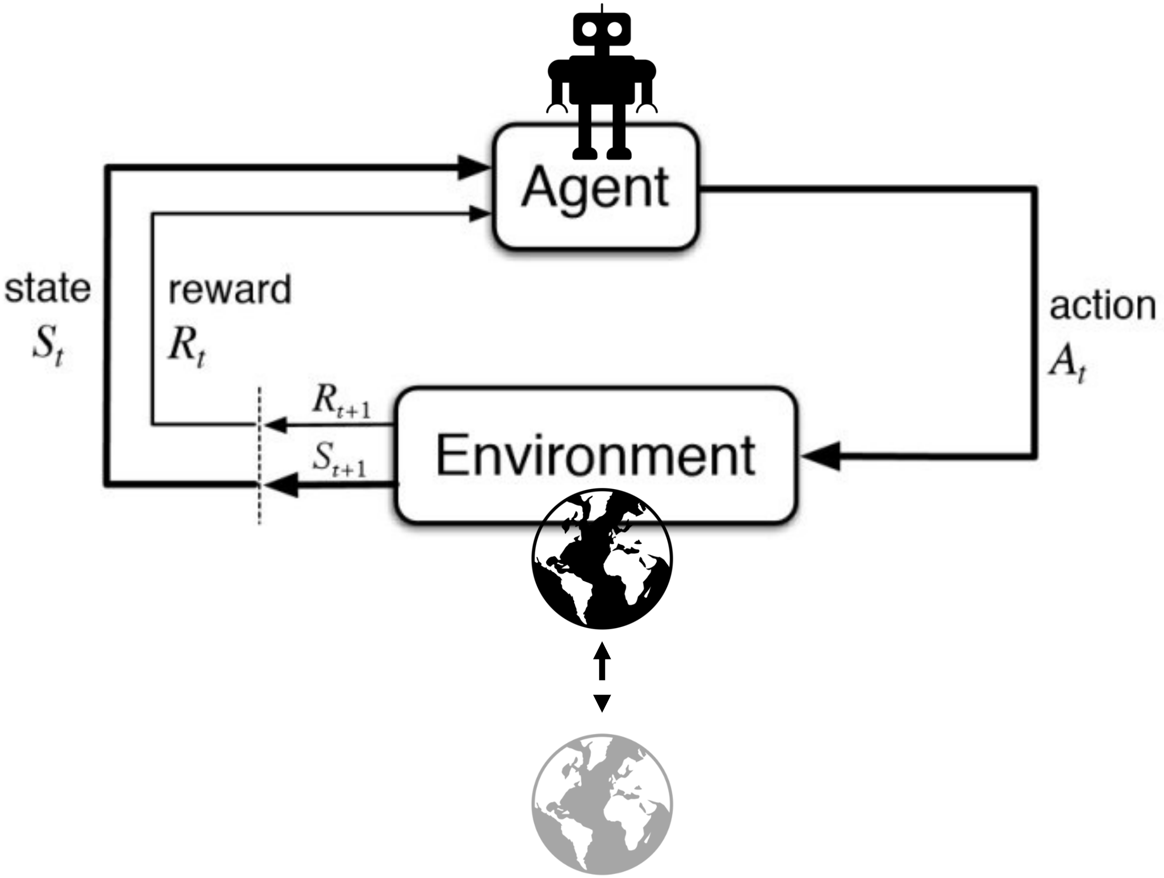
Model-based RL



Imitation Learning



What if we just learned how the world worked?



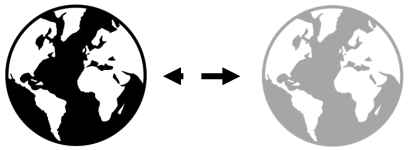
$$\max_\theta \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} r(s_t, a_t) \right]$$

1. <u>Learn</u> 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

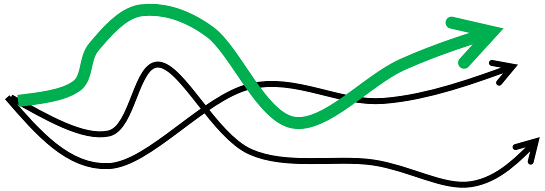
Model Based RL – **P**roblem Statement

Model Learning



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

Planning

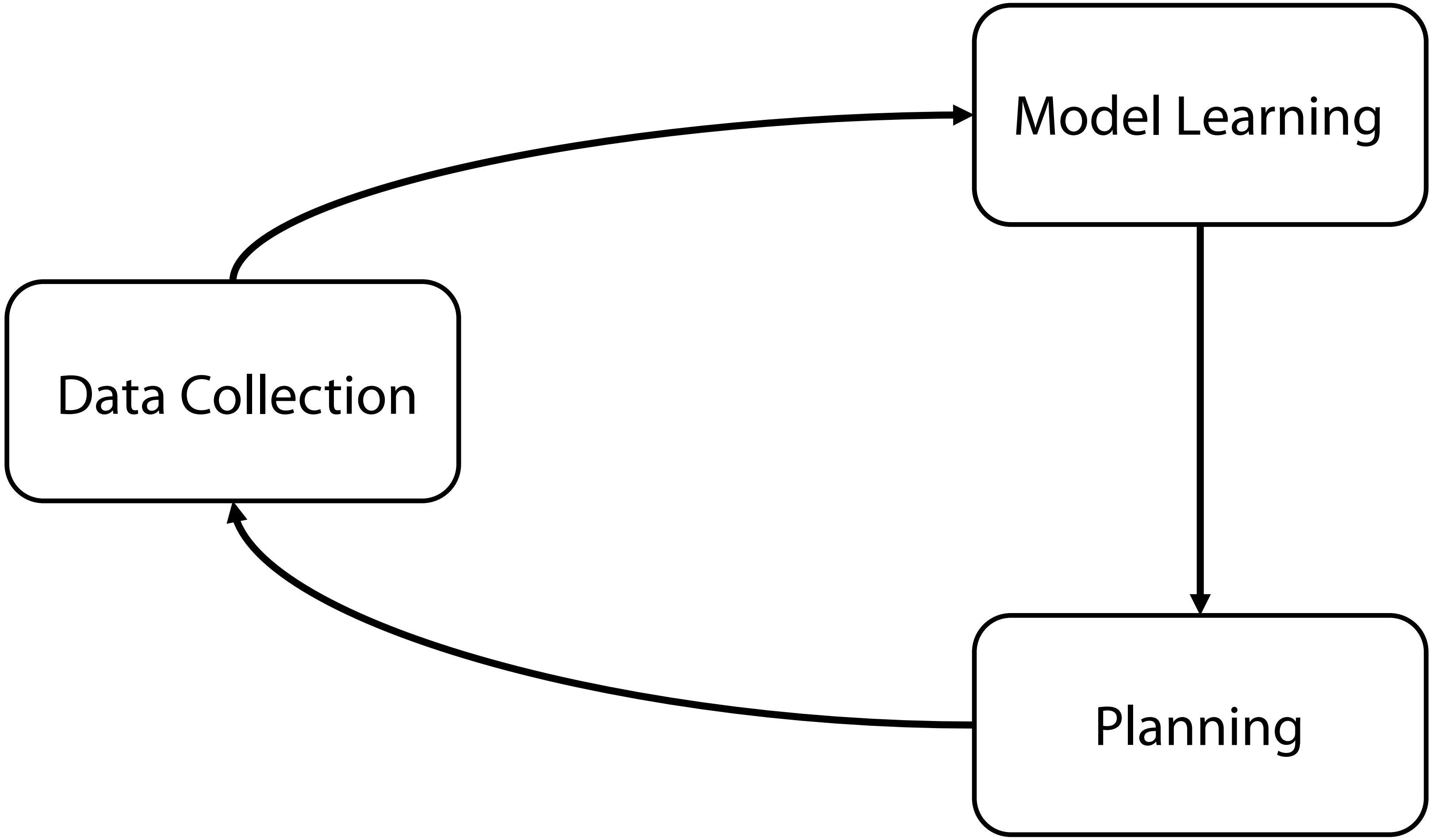


$$\arg\max_{\pi} \mathbb{E}_{\hat{p},\pi}\left[\sum_t r(s_t, a_t)\right]$$

Can also just be a single trajectory

How should we instantiate these?

Model Based RL – **A template**



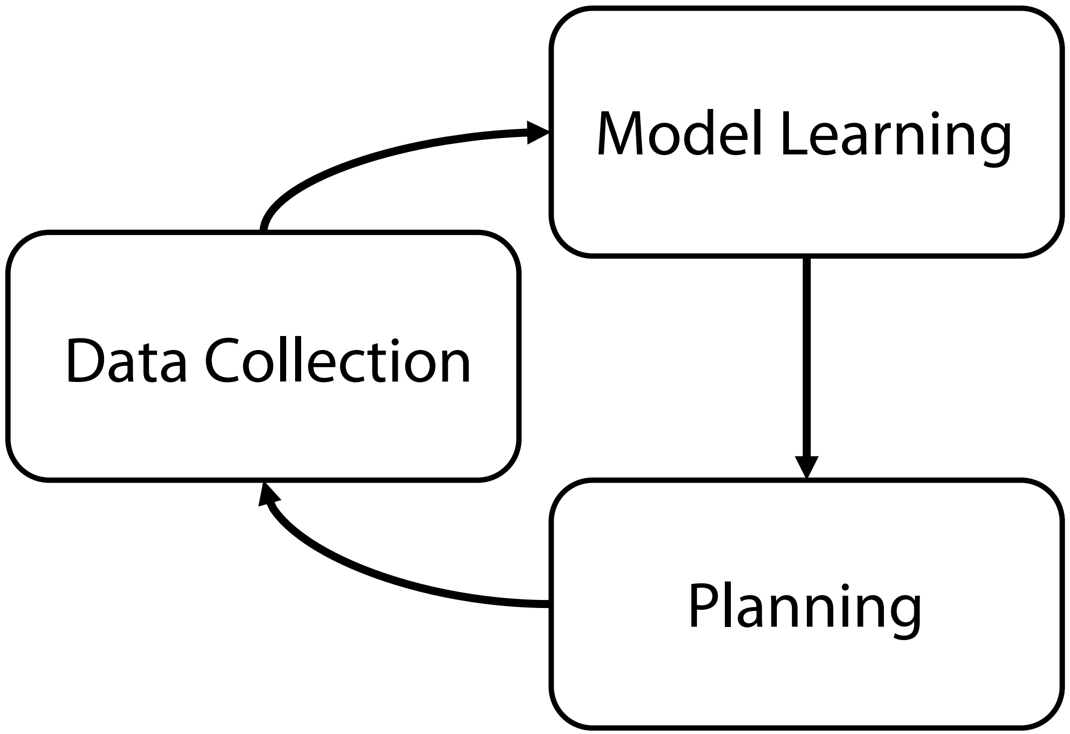
Model-Based RL Outline

⇨ Model based RL v0 → random shooting + MPC

Model based RL v1 → MPPI + MPC

Model based RL v2 → uncertainty based models

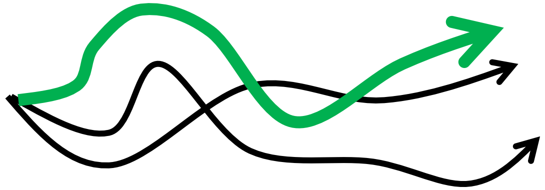
Model Based RL – **Naïve Algorithm (v0)**



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 (Planning)**

Planning

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

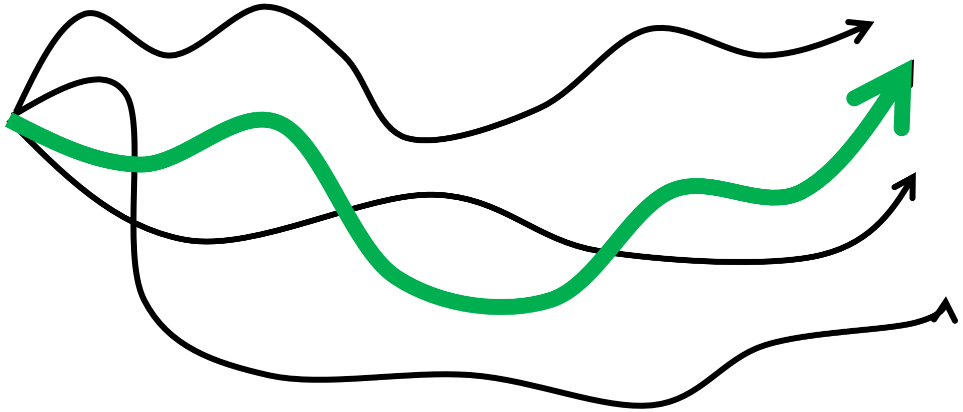
$$\hat{s}_{t+1} \sim \hat{p}_\theta(s_{t+1} | \hat{s}_t, a_t)$$

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

⇓ Just do random search!

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

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

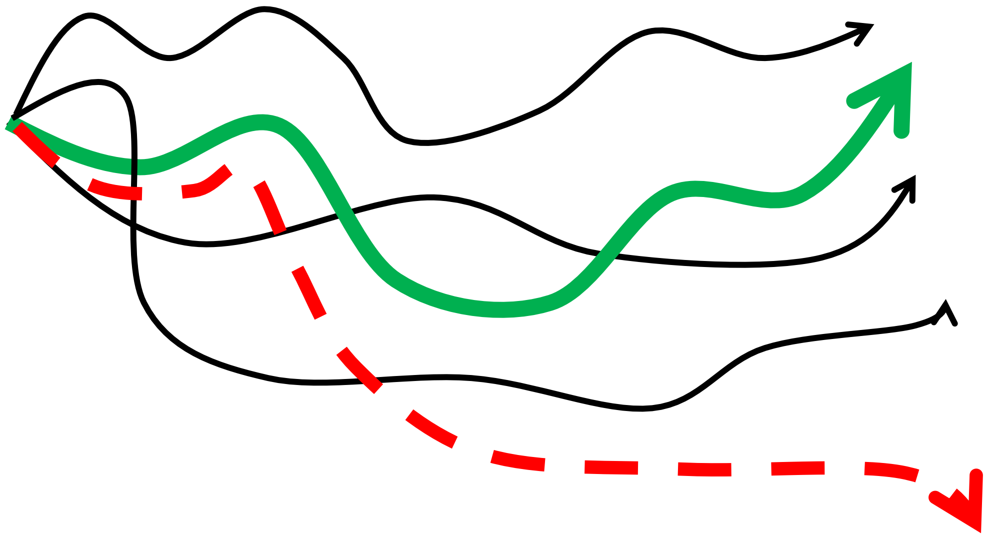


Just execute actions open loop!

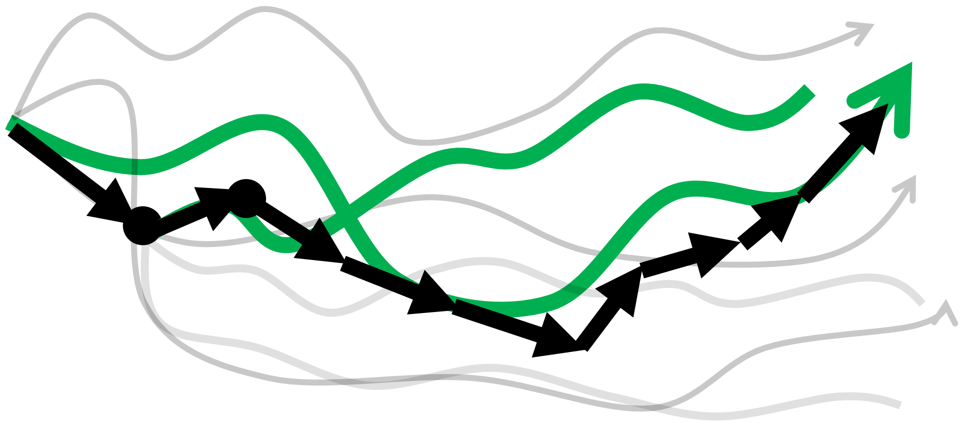
Can soften by taking softmax rather than argmax

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

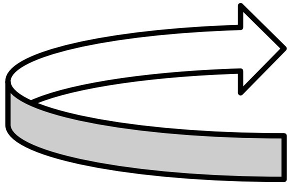
Without feedback, an open loop controller can diverge even for minimal noise



Replanning can help with divergence



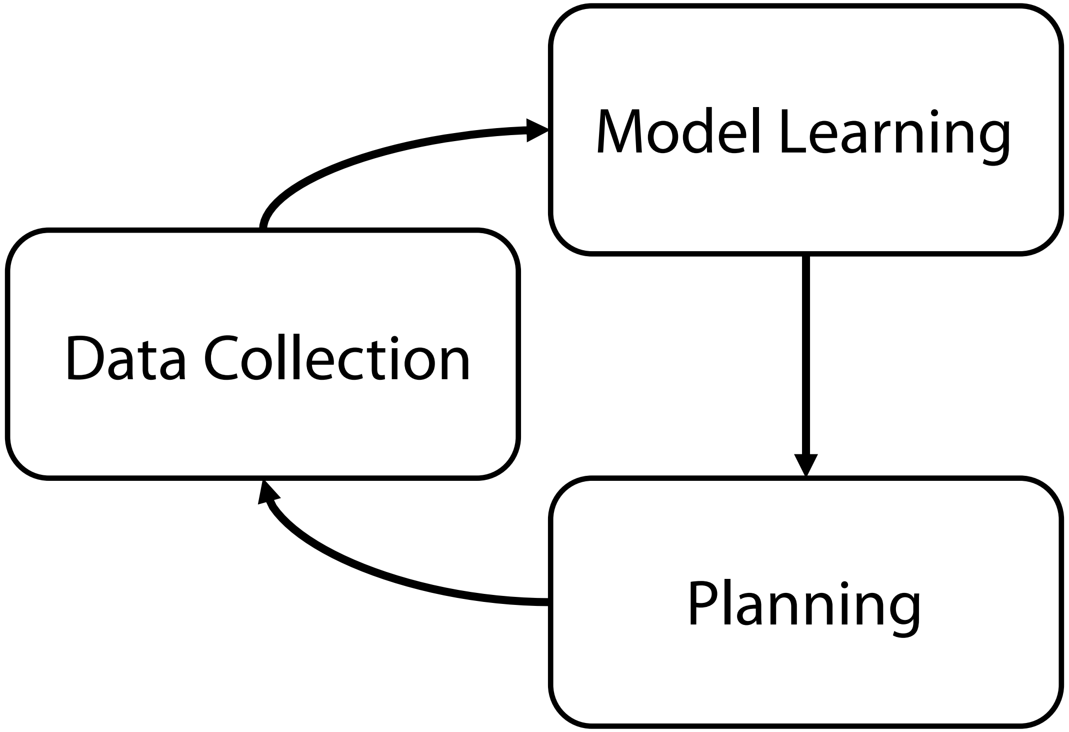
Model-Predictive/Receding Horizon Control



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)**

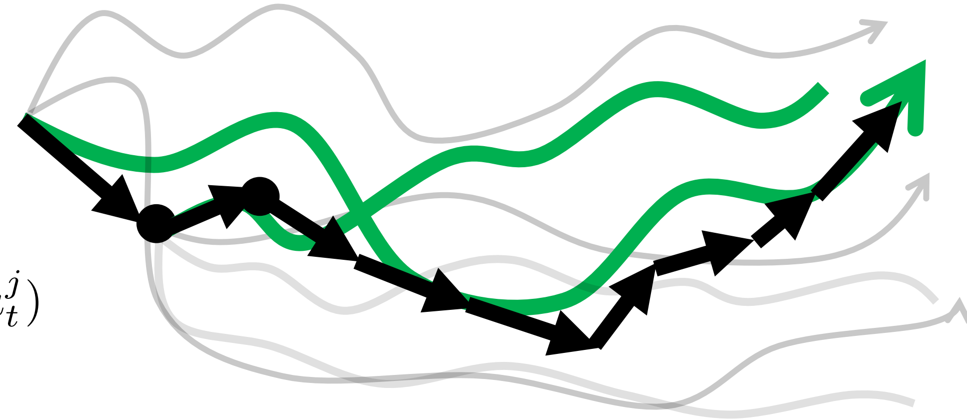
Data collected from planner



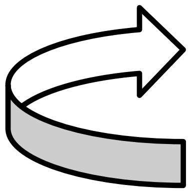
Maximum likelihood supervised Learning

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

Planning with Shooting + MPC



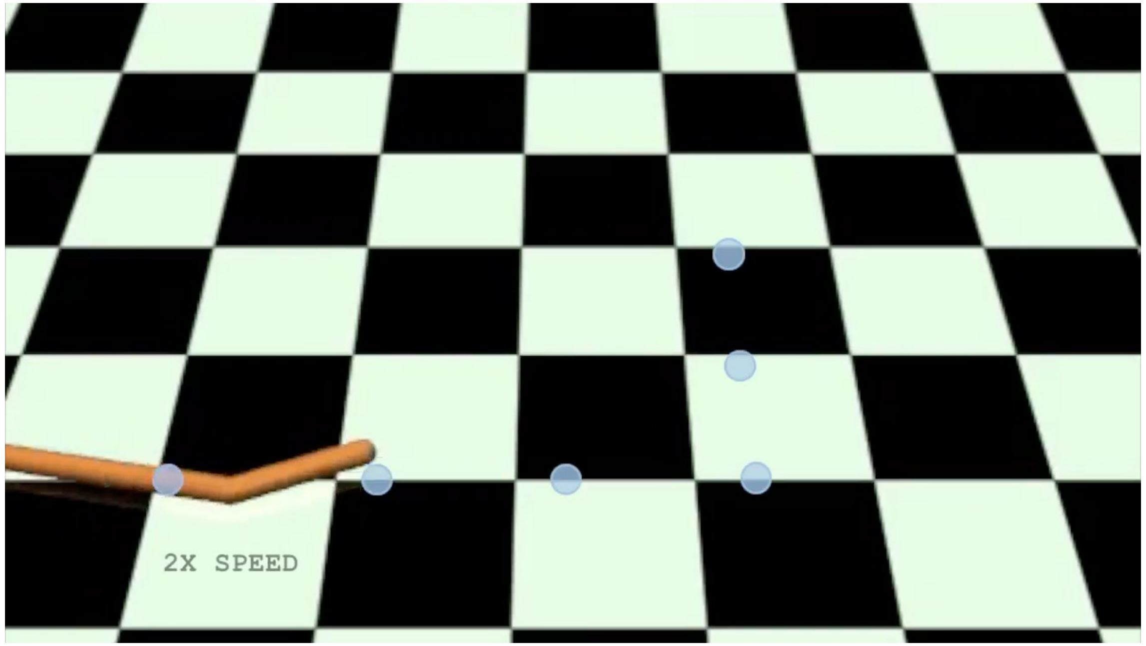
Better than open loop planning because of feedback



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

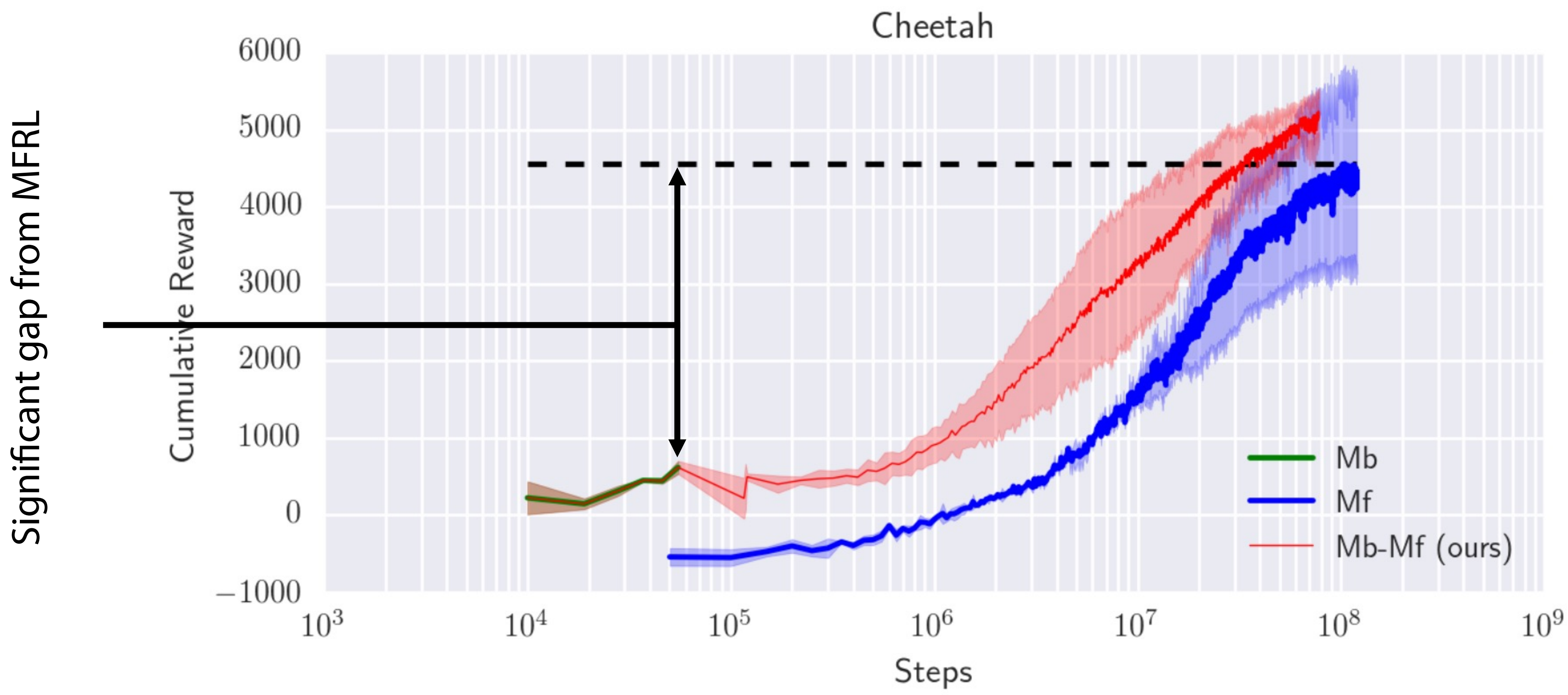
$$\hat{s}_{t+1}^j \sim \hat{p}_\theta(.|\hat{s}_t^j, a_t^j)$$

Does it work?



Just 20 minutes of training time with random data!

Does it work?


Significant gap from MFRL
Cheetah
6000
5000
4000
3000
2000
1000
0
−1000
Cumulative Reward
10^3
10^4
10^5
10^6
10^7
10^8
10^9
Steps
Mb
Mf
Mb-Mf (ours)

Model-Based RL Outline

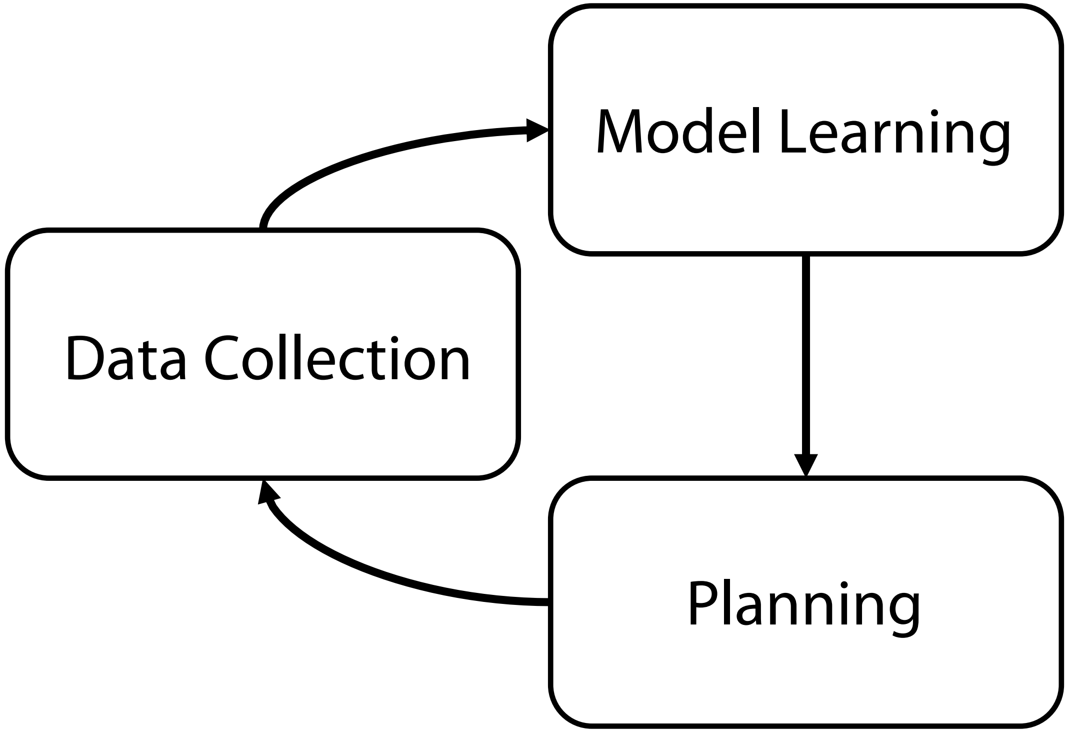
Model based RL v0 → random shooting + MPC

⇨ Model based RL v1 → MPPI + MPC

Model based RL v2 → uncertainty based models

What might be the issue?

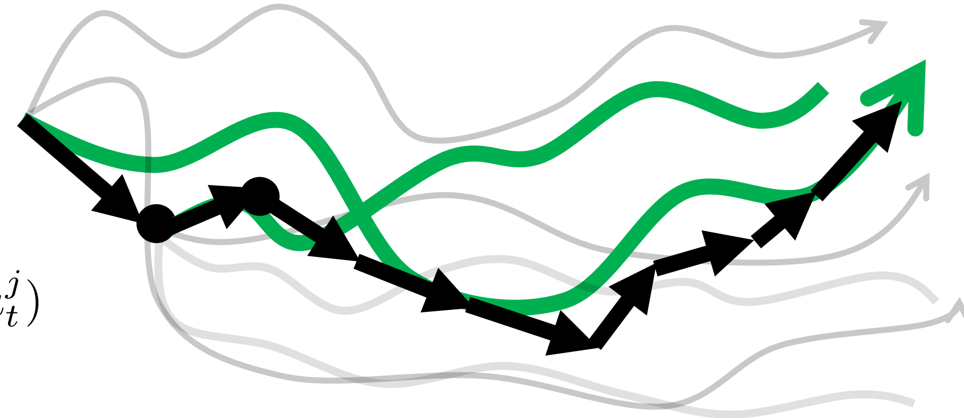
Data collected from planner



Maximum likelihood supervised Learning

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

Planning with Shooting + MPC



Searching for a needle in a haystack by random shooting, high variance!

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

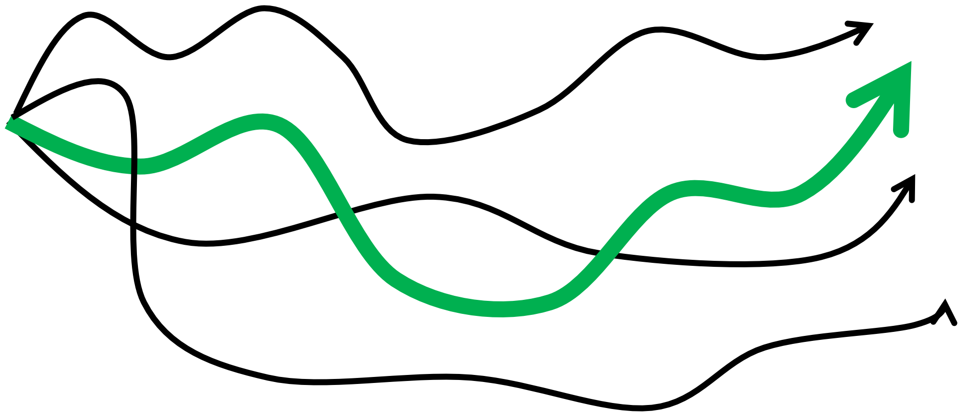
$$\hat{s}_{t+1}^j \sim \hat{p}_\theta(.|\hat{s}_t^j, a_t^j)$$

Better Sampling Techniques for Shooting

Sampled from stationary uniform/gaussian distribution

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

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

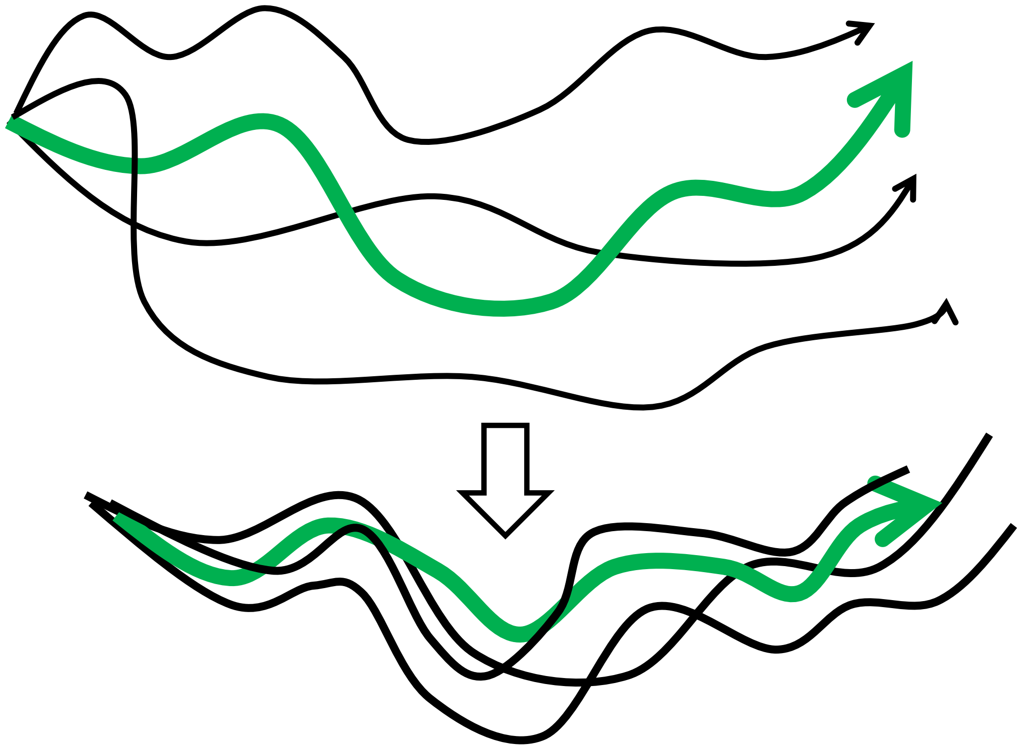


Can we inform the sampling function with the reward function?

Idea: Iteratively upweight sampling distribution around the things that are higher returns

Better Sampling Techniques for Shooting - MPPI

Idea: Iteratively upweight sampling distribution around the things that are higher returns



Referred to as **MPPI**, lower variance!

Sample N action sequences

$$(a_0^j, a_1^j, \ldots, a_T^j)_{j=1}^N \sim p(a)$$

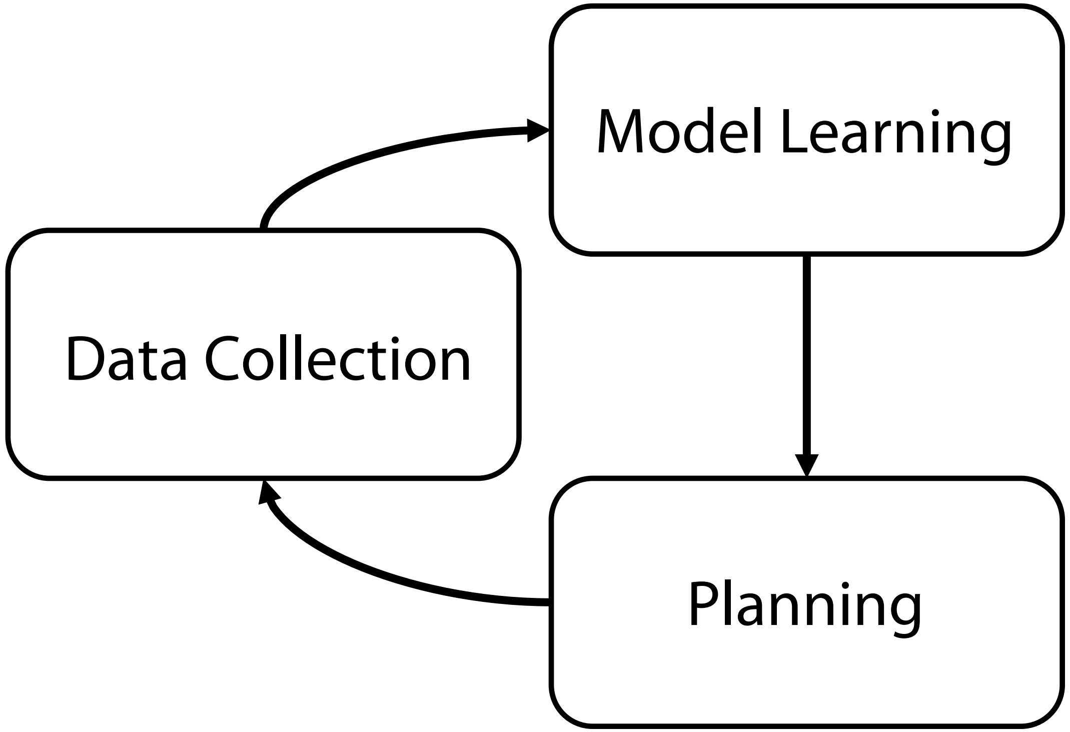
Sample trajectories using these action sequences with the model $\hat{p}_\theta$

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

Update action sampler by upweighting high return actions

$$p(a) \leftarrow p(a)\frac{\exp(\sum_t r(s_t, a_t))}{Z}$$

Model Based RL – **Better Sampling Methods (v1)**

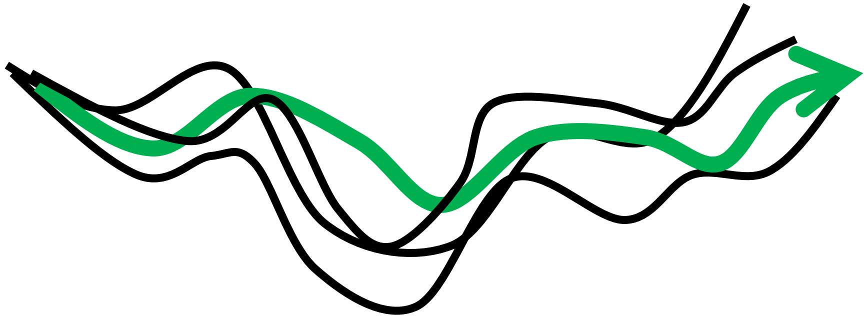


Maximum likelihood supervised learning

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

Data collected from planner

Planning with MPPI + MPC



Better than random shooting + MPC, since lower variance!

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

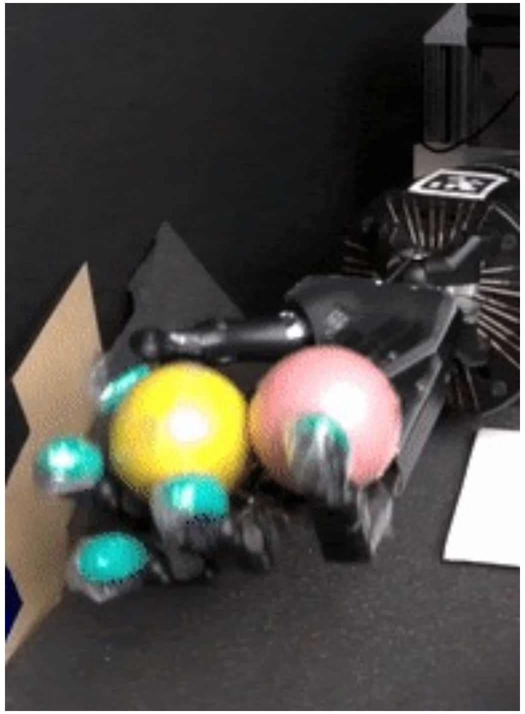
$$\hat{s}_{t+1}^j \sim \hat{p}_\theta(.|\hat{s}_t^j, a_t^j)$$

$$p(a) \leftarrow p(a) \frac{\exp(\sum_t r(s_t, a_t))}{Z}$$

Does it work?

Does it work?



Just 2 hours of real robot training

Model-Based RL Outline

Model based RL v0 → random shooting + MPC

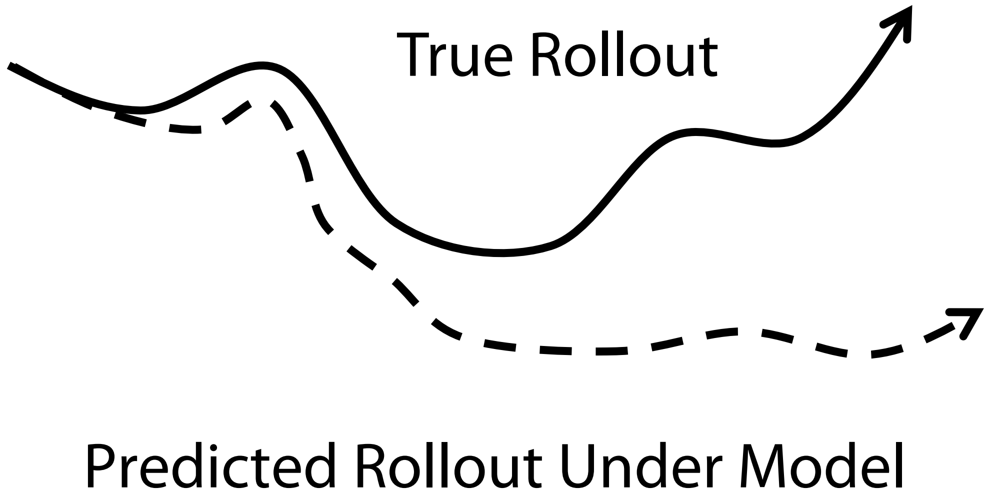
Model based RL v1 → MPPI + MPC

⇨ Model based RL v2 → uncertainty based models

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!

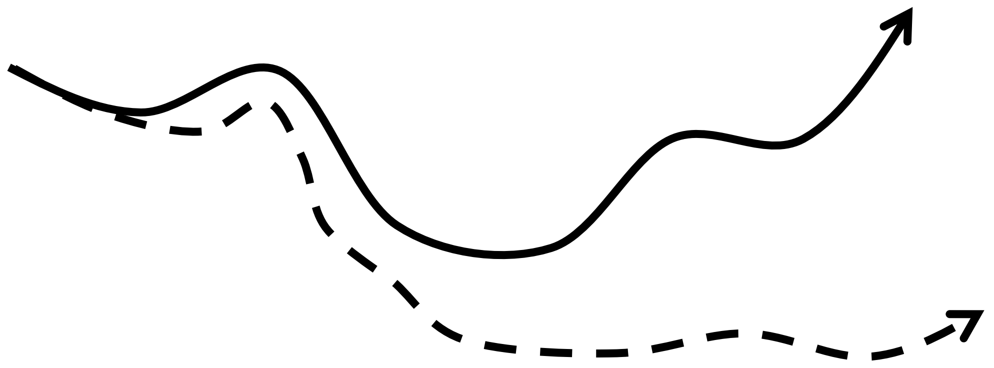
Most trained deep models can only roll out for 5-10 steps maximum!

How might we deal with compounding error?

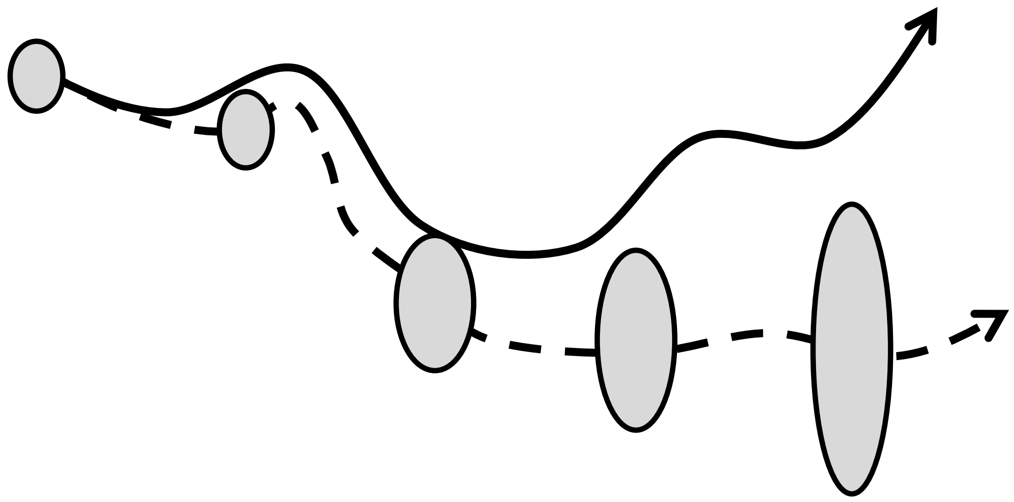
Idea: Estimate when OOD and account for it

→ Measure uncertainty!

Maximum likelihood models



Uncertainty-aware models



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

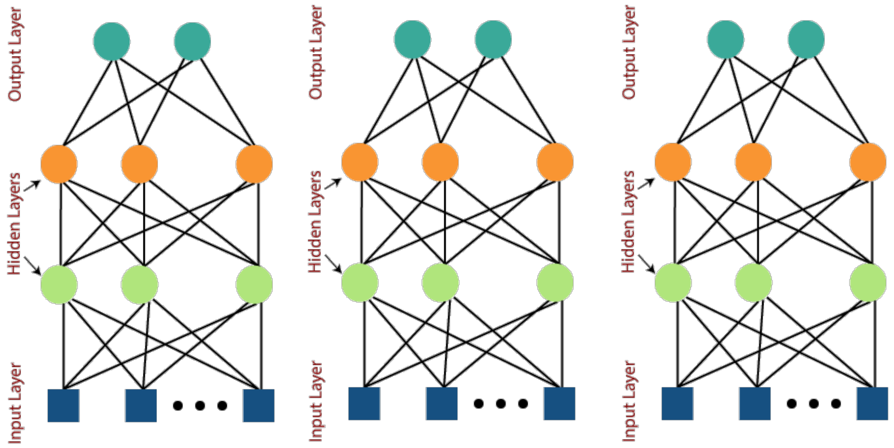
How might we measure uncertainty?

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

Difficult to estimate directly!

Learn an ensemble of models

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

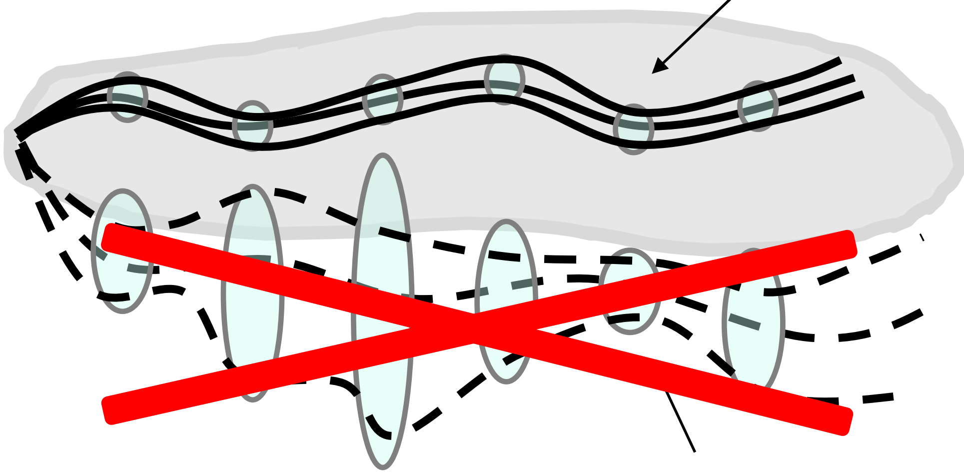


Low data regime → high ensemble variance

Approximate posterior

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

Take **pessimistic** value under the uncertain dynamics



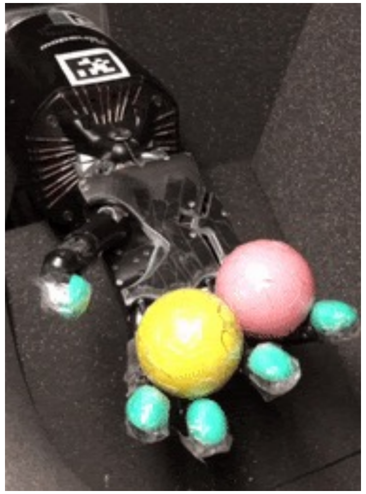
Penalize ensemble variance

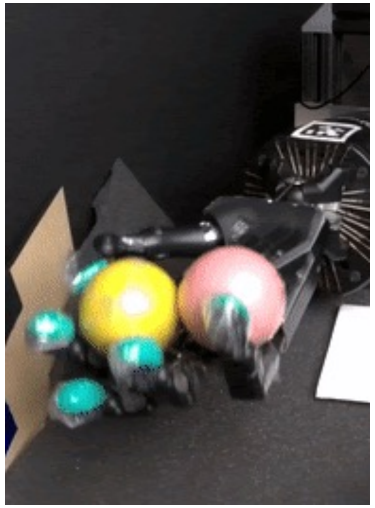
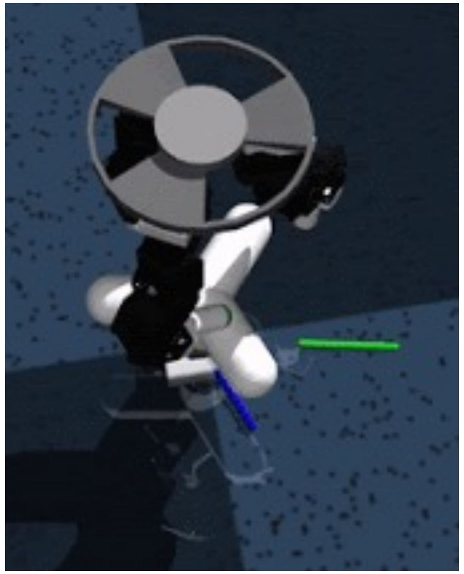
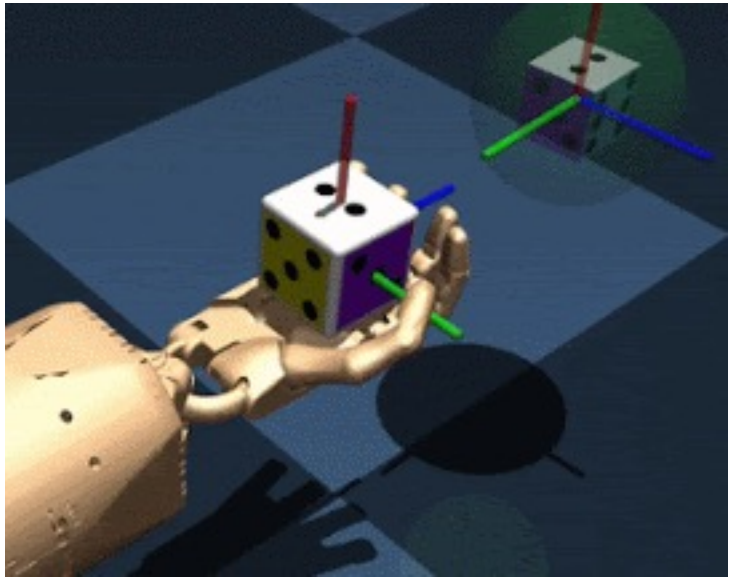
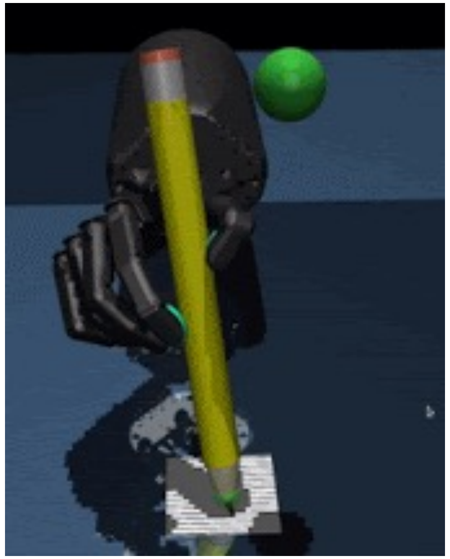
$$\arg \max_{(a_0^j, a_1^j, \ldots, a_T^j)_{j=1}^N} \sum_{i=1}^{K} \sum_{t=0}^{T} r((\hat{s}_t^j)^i, a_t^j) - \lambda \mathrm{Var}((\hat{s}_t^j)^i)$$

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

Avoids overly OOD settings since these states are explicitly penalized

Does this work?



Class Outline

State Estimation

- Robotic System Design
- Filtering
- Localization
- SLAM

Control

- Feedback Control
- PID Control
- MPC
- LQR

Planning

- Search
- Heuristic Search
- Motion Planning
- Lazy Search

Learning

- Imitation Learning
- Policy Gradient
- Model-Based RL