Autonomous Robotics

Winter 2025

Abhishek Gupta

TAs: Carolina Higuera, Entong Su, Bernie Zhu

Class Outline

State Estimation

- Robotic System Design
- Filtering
- Localization
- SLAM

Control

- Feedback Control
- PID Control
- MPC
- LQR

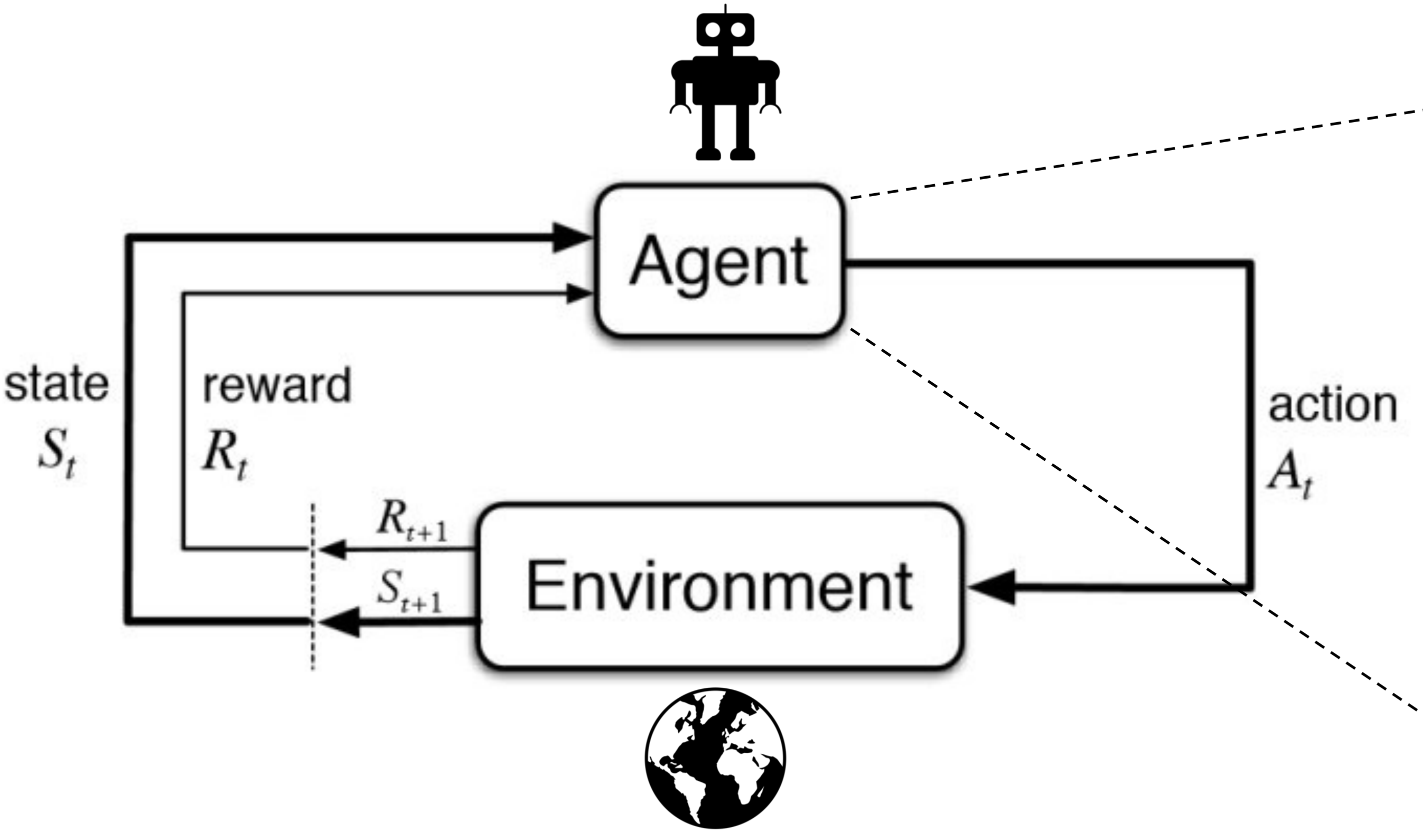
Planning

- Search
- Heuristic Search
- Motion Planning
- Lazy Search

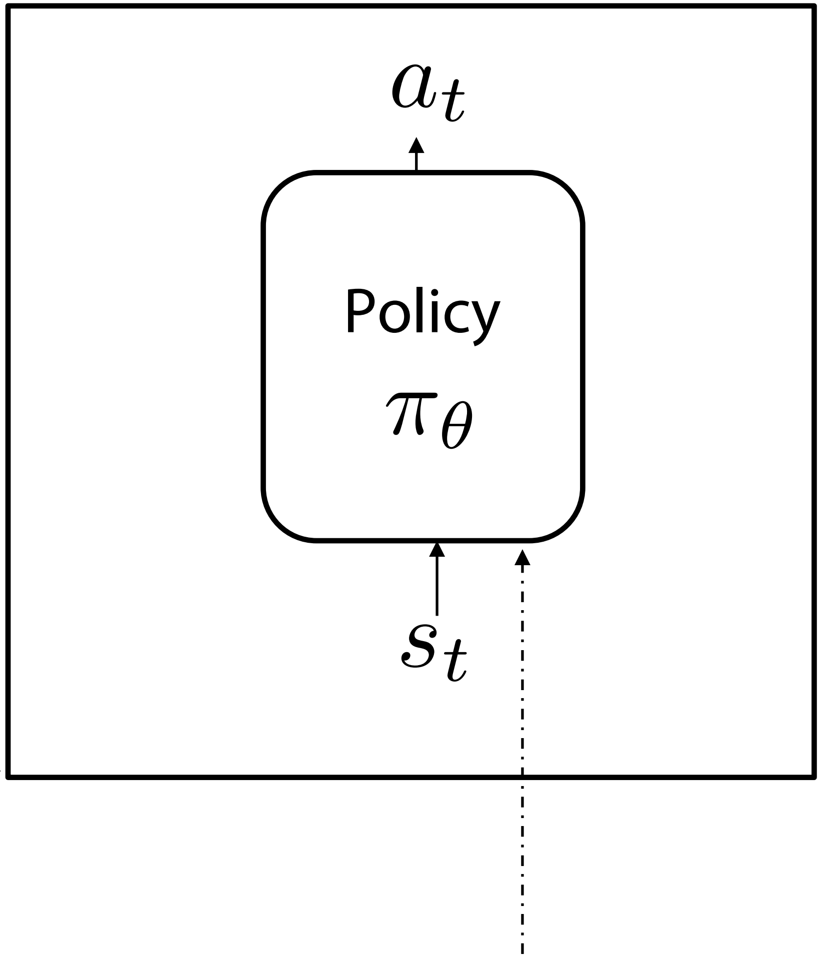
Learning

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

Reinforcement Learning Formalism



Rules for choosing actions



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

Trajectory sampled using policy

Needs to be learned

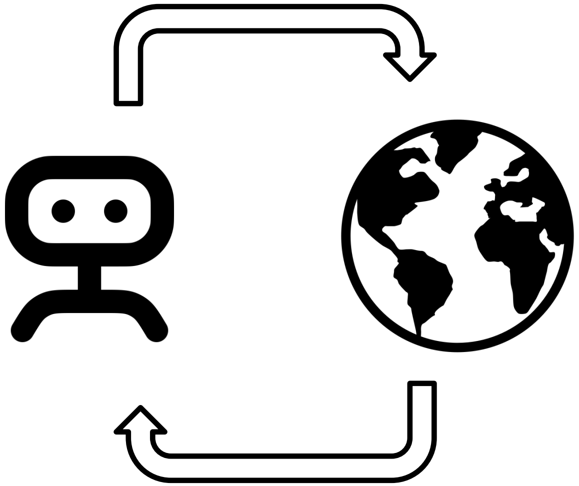
Ok so how can we learn policies?

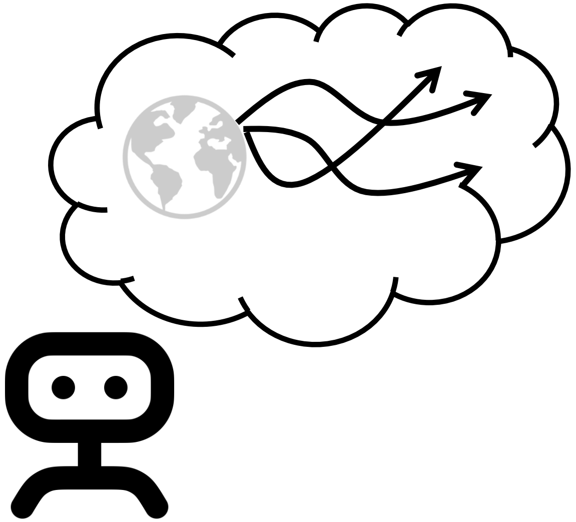
$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} r(s_t, a_t) \right]$$

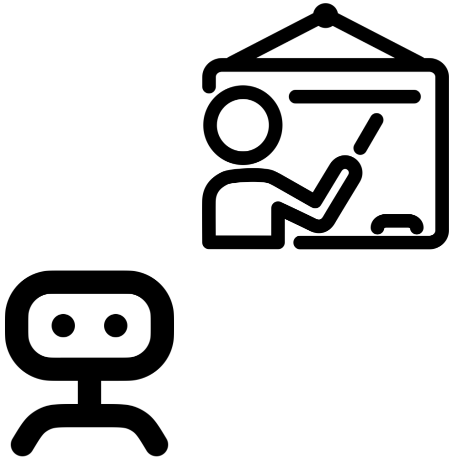
<u>Model-free RL</u>

<u>Model-based RL</u>

<u>Imitation Learning</u>







Behavior Cloning

Given: Demonstrations of optimal behavior

Goal: Train a policy to mimic the demonstrator

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

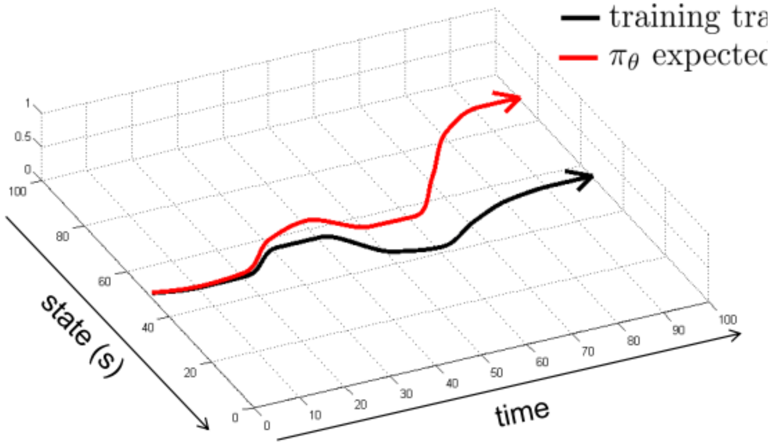
Discrete vs continuous

Maximum likelihood

```
if isinstance(env.action_space, gym.spaces.Box):
    criterion = nn.MSELoss()
else:
    criterion = nn.CrossEntropyLoss()
# Extract initial policy
model = student.policy.to(device)
def train(model, device, train_loader, optimizer):
   model.train()
   for batch_idx, (data, target) in enumerate(train_loader):
      data, target = data.to(device), target.to(device)
      optimizer.zero_grad()
      if isinstance(env.action_space, gym.spaces.Box):
         if isinstance(student, (A2C, PPO)):
            action, _, _ = model(data)
         else:
            action = model(data)
         action_prediction = action.double()
      else:
         dist = model.get_distribution(data)
         action_prediction = dist.distribution.logits
         target = target.long()
      loss = criterion(action_prediction, target)
      loss.backward()
      optimizer.step()
```

So does behavior cloning really work?

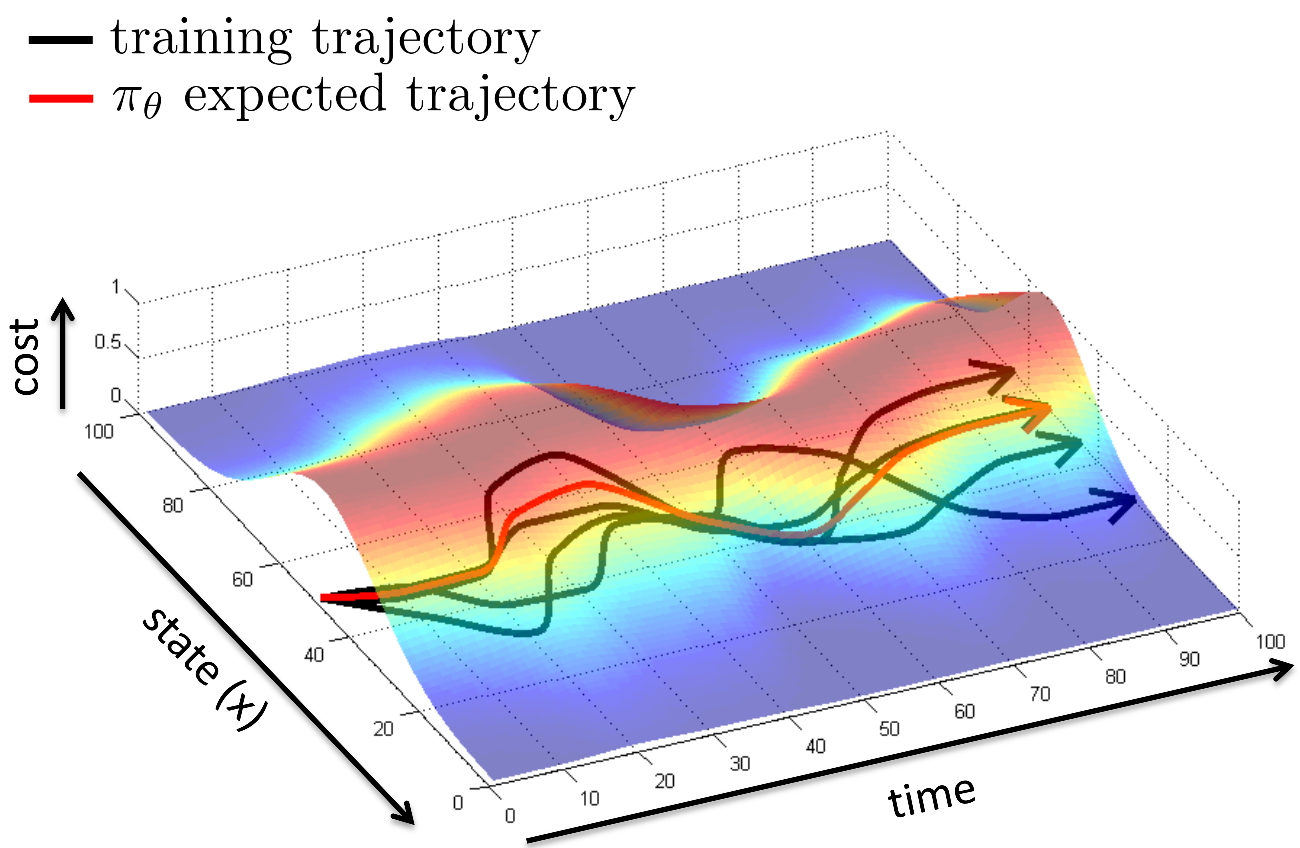
- Imitation Learning ≠ Supervised Learning



$$\arg\max_{\theta} \mathbb{E}_{(s^*,a^*)\sim\mathcal{D}} \left[\log \pi_\theta(a^*|s^*)\right] \qquad \mathbb{E}_{(s,a)\sim\rho(\pi)} \left[1(a = a^*)\right]$$

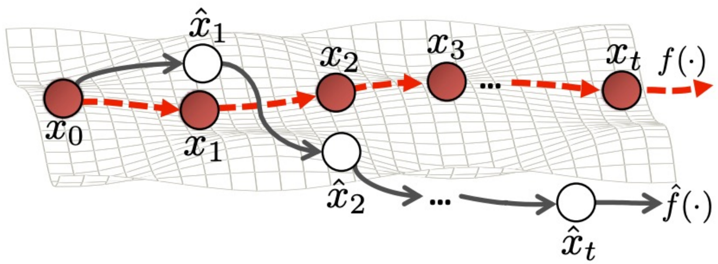
Not the same!

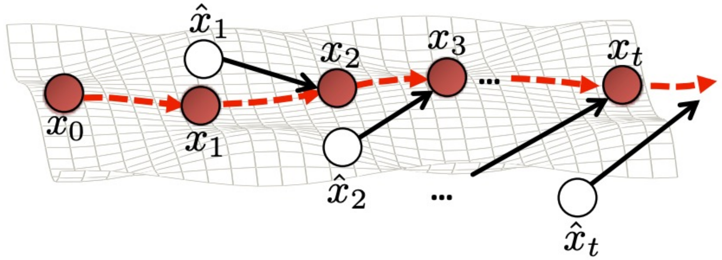
Correcting for Compounding Errors



stability

Corrective labels that bring you back to the data





Corecting Errors via DAgger

can we make $p_{\text{data}}(\mathbf{o}_t) = p_{\pi_\theta}(\mathbf{o}_t)$?

idea: instead of being clever about $p_{\pi_\theta}(\mathbf{o}_t)$, be clever about $p_{\text{data}}(\mathbf{o}_t)$!

DAgger: **D**ataset **A**ggregation

goal: collect training data from $p_{\pi_\theta}(\mathbf{o}_t)$ instead of $p_{\text{data}}(\mathbf{o}_t)$

how? just run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

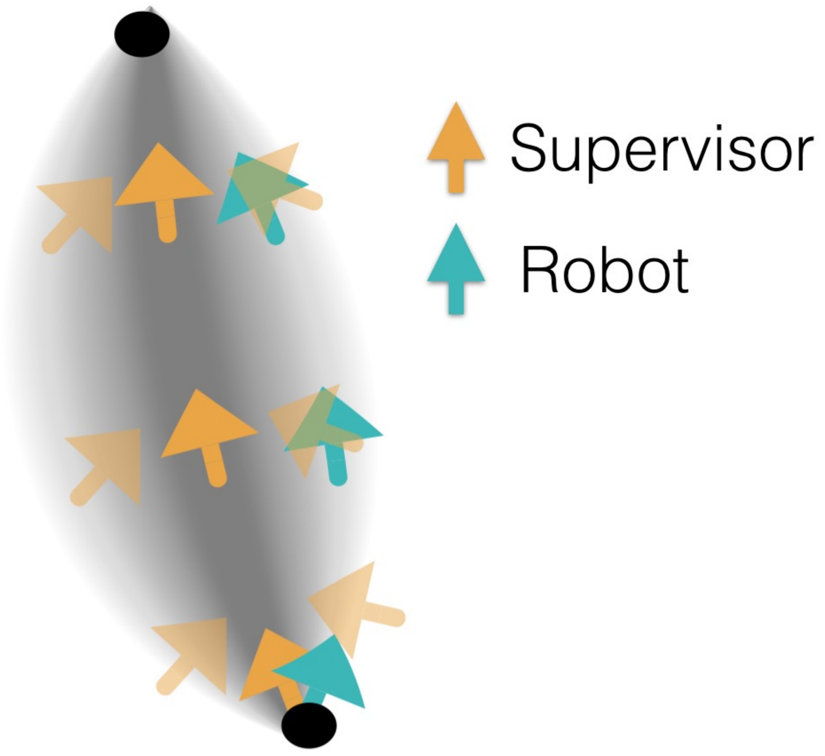
but need labels $\mathbf{a}_t$!

1. train $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \dots, \mathbf{o}_N, \mathbf{a}_N\}$
2. run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \dots, \mathbf{o}_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{a}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Ross et al. '11

DART: Noising the Data Collection Process

Key idea: force the human to correct for noise **during** training



Noise Injection

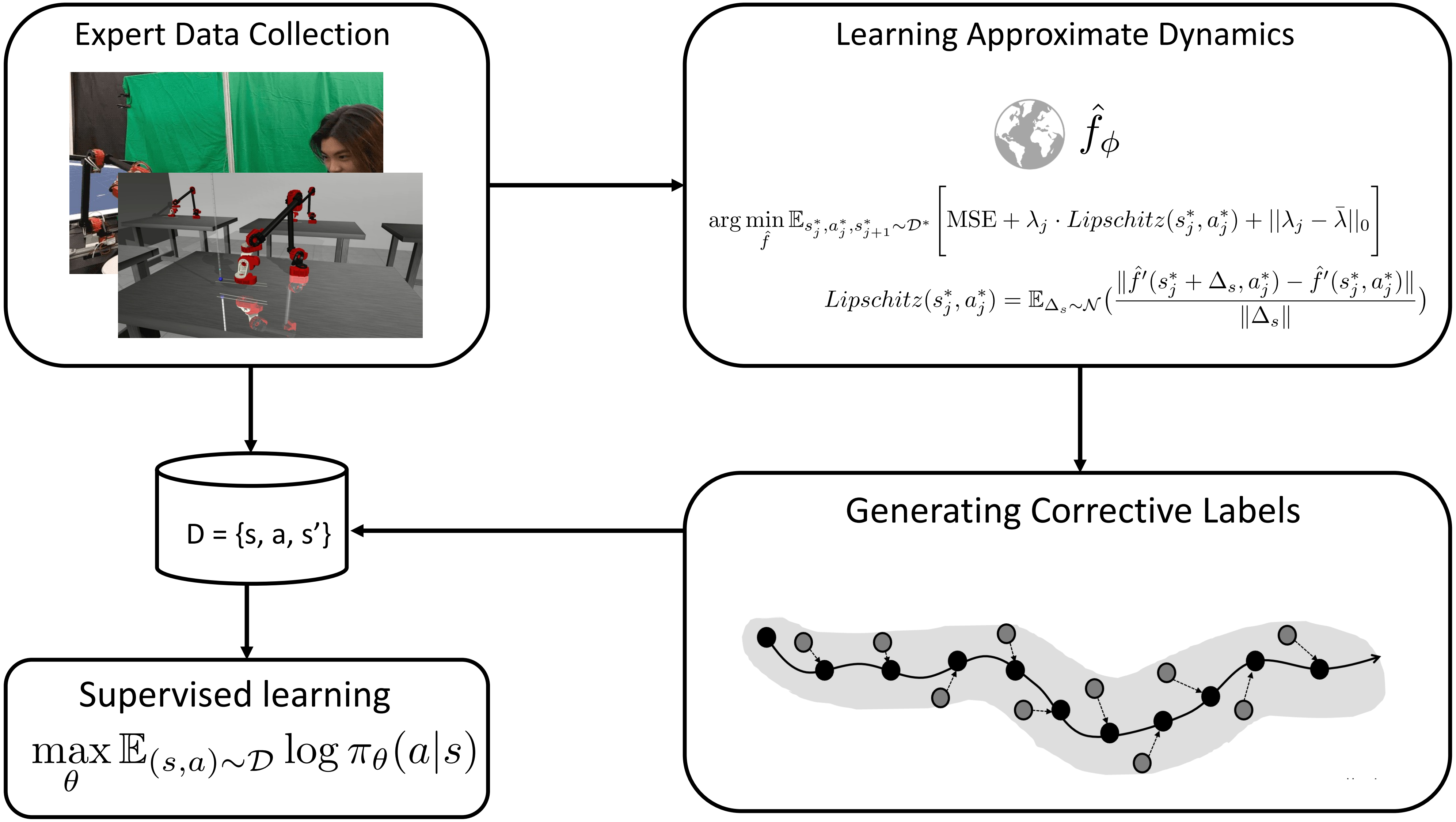
$$\hat{\psi}_{k+1} = \underset{\psi}{\operatorname{argmin}}\, E_{p(\xi|\pi_{\theta^*},\psi_k)} - \sum_{t=0}^{T-1} \log\left[\pi_{\theta^*}(\pi_{\hat{\theta}}(\mathbf{x_t})|\mathbf{x_t},\psi)\right]$$

Maximize likelihood

Under noise during data collection

DART: Noise Injection for Robust Imitation Learning, Laskey et al CoRL '17

CCIL: Generating Synthetic Corrective Labels



Lecture Outline

Recap

↓

Imitation Learning: Improvements – Multimodality

↓

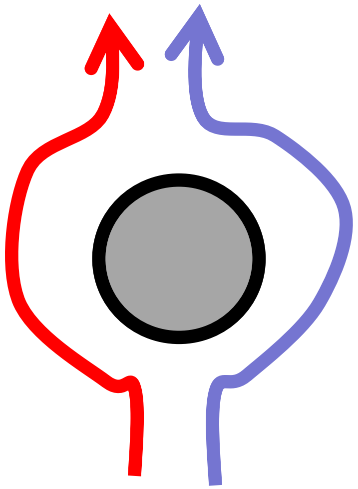
Policy Gradient

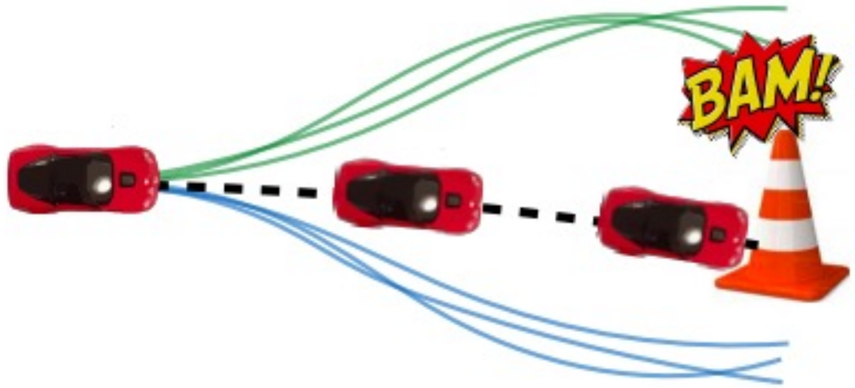
↓

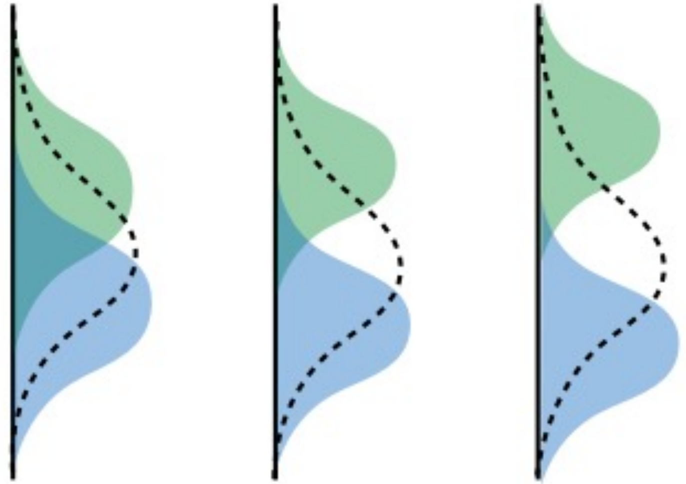
Improving Policy Gradient

Why might we fail to fit the expert?

Multimodal behavior.. amongst other reasons





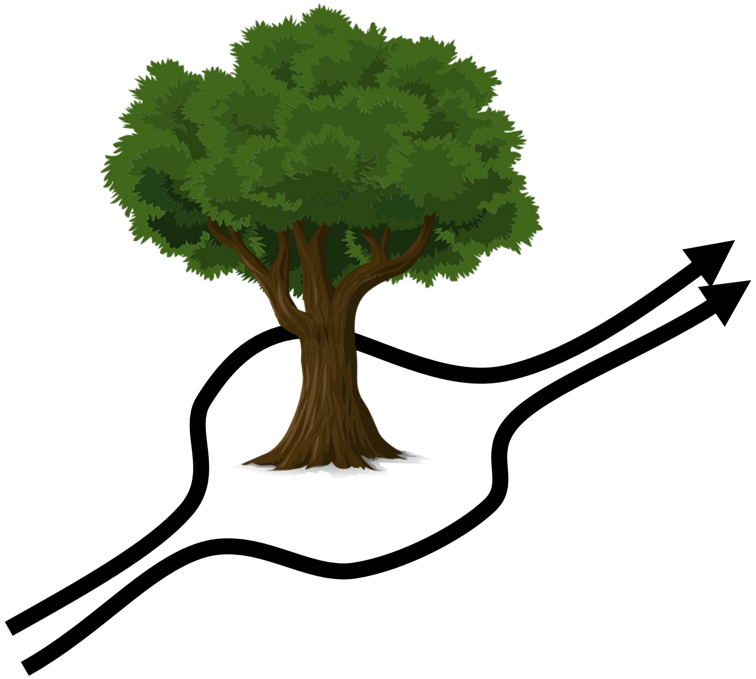


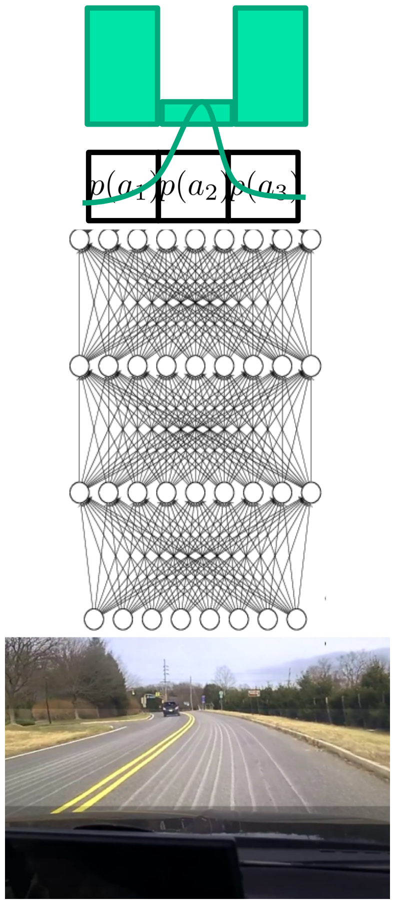
Not a matter of network size! It's about distributional expressivity

Why might we fail to fit the expert?

Multimodal behavior → use more **expressive** probability distributions

1. Output mixture of Gaussians
2. Latent variable models
3. Autoregressive discretization
4. Diffusion models
5. …

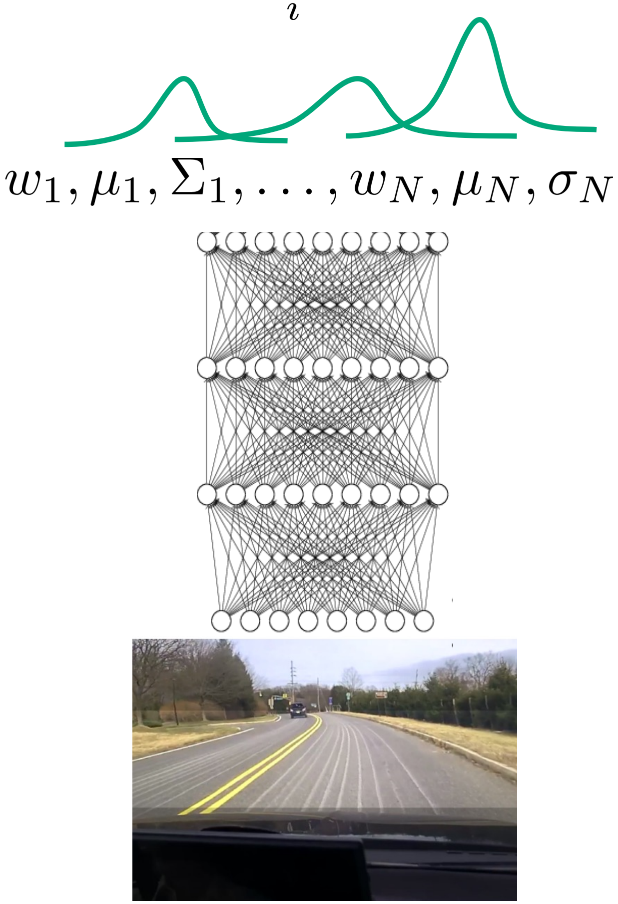




Why might we fail to fit the expert?

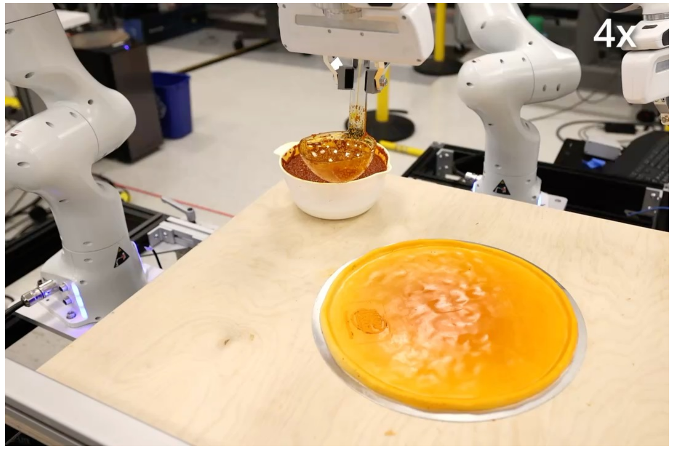
1. Output mixture of Gaussians
2. Latent variable models
3. Autoregressive discretization
4. Diffusion models
5. …

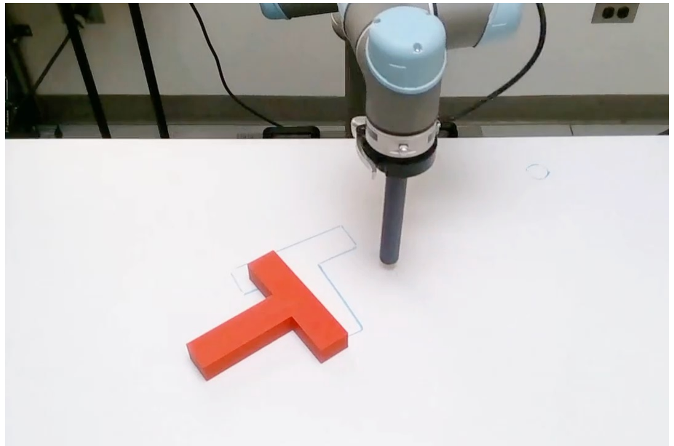
$$\pi(\mathbf{a}|\mathbf{o}) = \sum_i w_i \mathcal{N}(\mu_i, \Sigma_i)$$

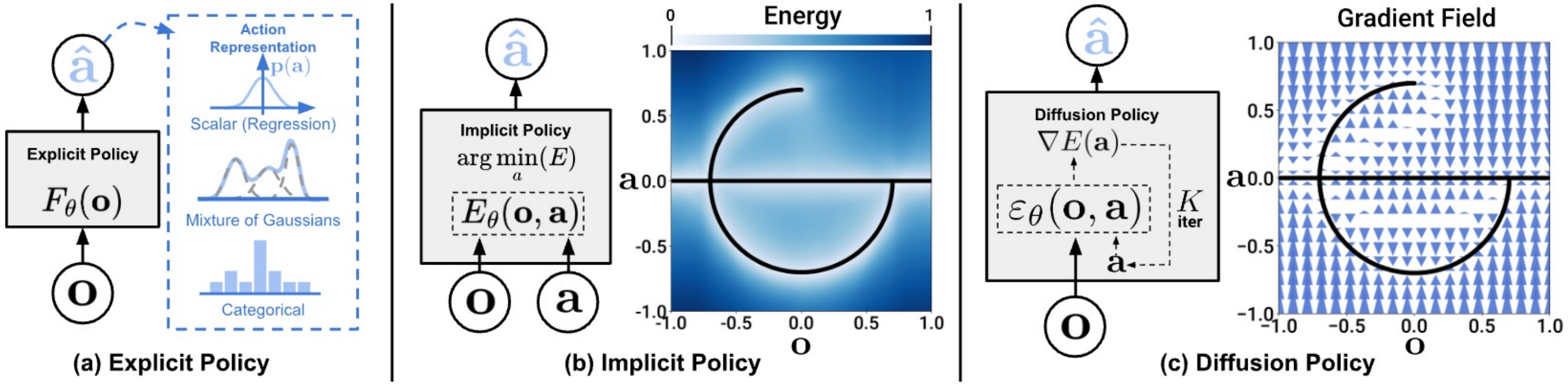


Why might we fail to fit the expert?

1. Output mixture of Gaussians
2. Latent variable models
3. Autoregressive discretization
4. Diffusion models
5. …







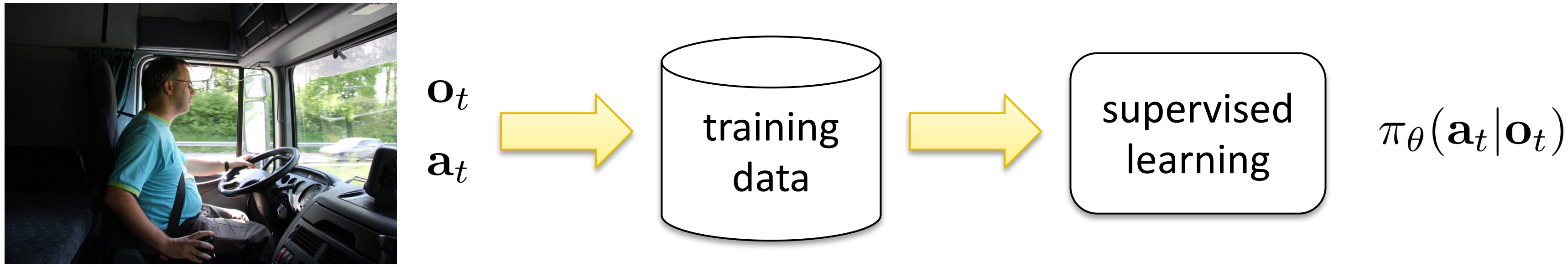
Some cool imitation videos

1x and tesla humanoid robots

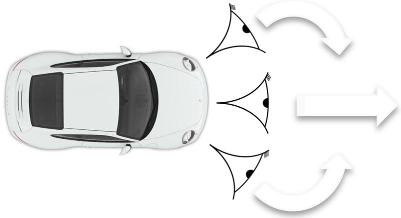
ALOHA Manipulation

TRI Diffusion Policies

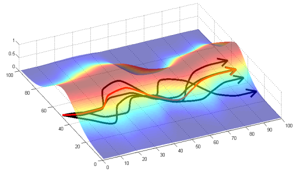
Perspectives on Imitation – don't believe everything you see online

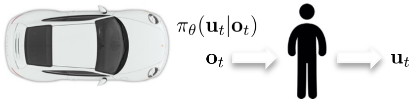


- Pros:
  - Easy to use, no additional infra
  - Can sometimes be unreasonably effective



- Cons:
  - Challenges of compounding error, multimodality
  - Doesn't really generalize
  - Expensive in terms of data collection!





Lecture Outline

Recap

↓

Imitation Learning: Improvements – Multimodality

↓

Policy Gradient

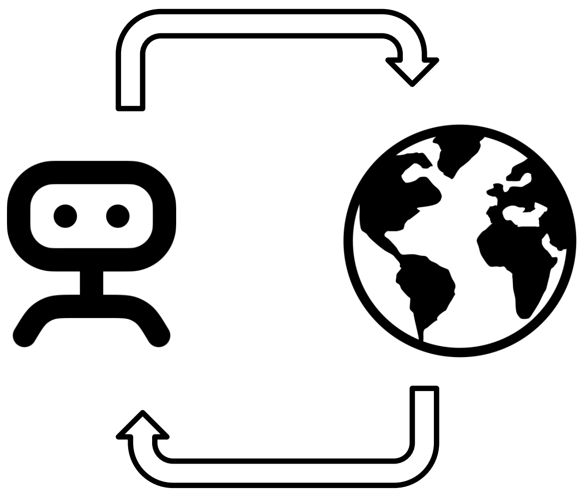
↓

Improving Policy Gradient

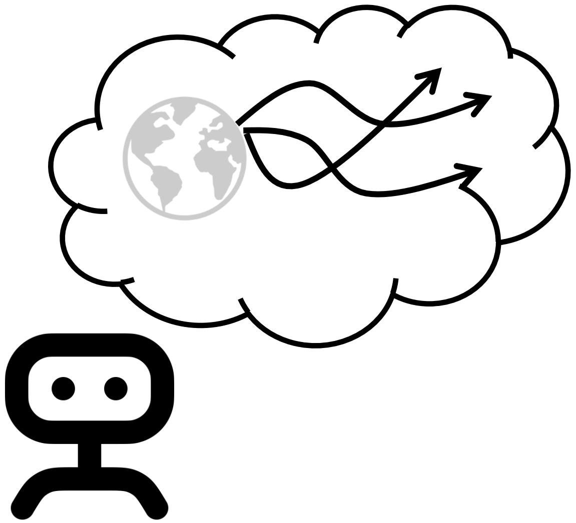
Ok so how can we learn policies?

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

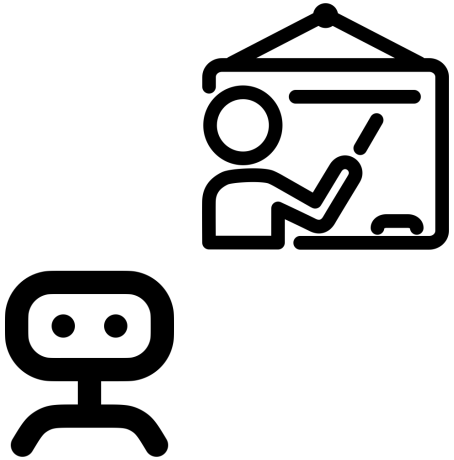
Model-free RL



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

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

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

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

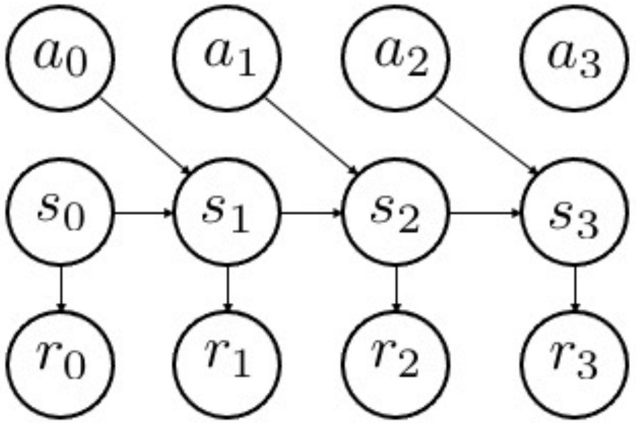
$$= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) R(\tau) d(\tau) \; = \mathbb{E}_{p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) R(\tau) \right]$$

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

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

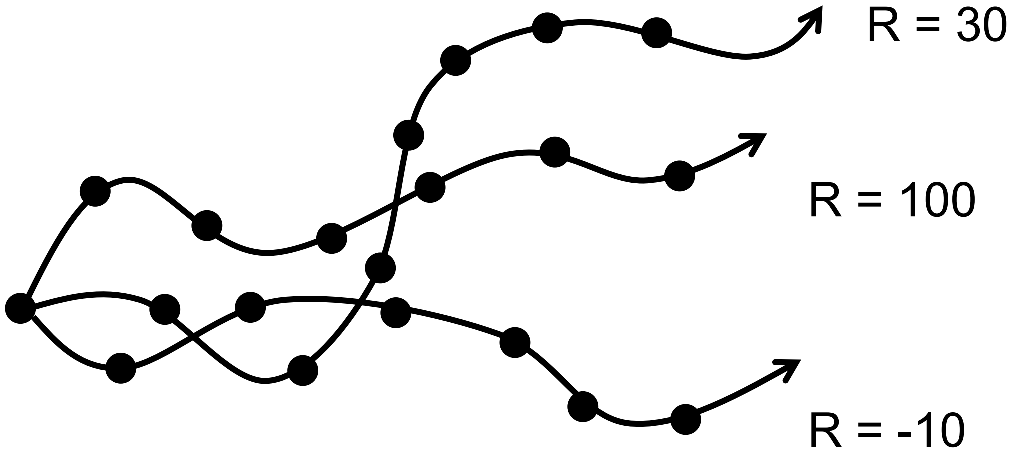
$$\nabla_\theta J(\theta) = \mathbb{E}_{\substack{s_0 \sim p(s_0) \\ s_{t+1} \sim p(s_{t+1}|s_t, a_t) \\ a_t \sim \pi(a_t|s_t)}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=0}^{T} r(s_t, a_t) \right]$$

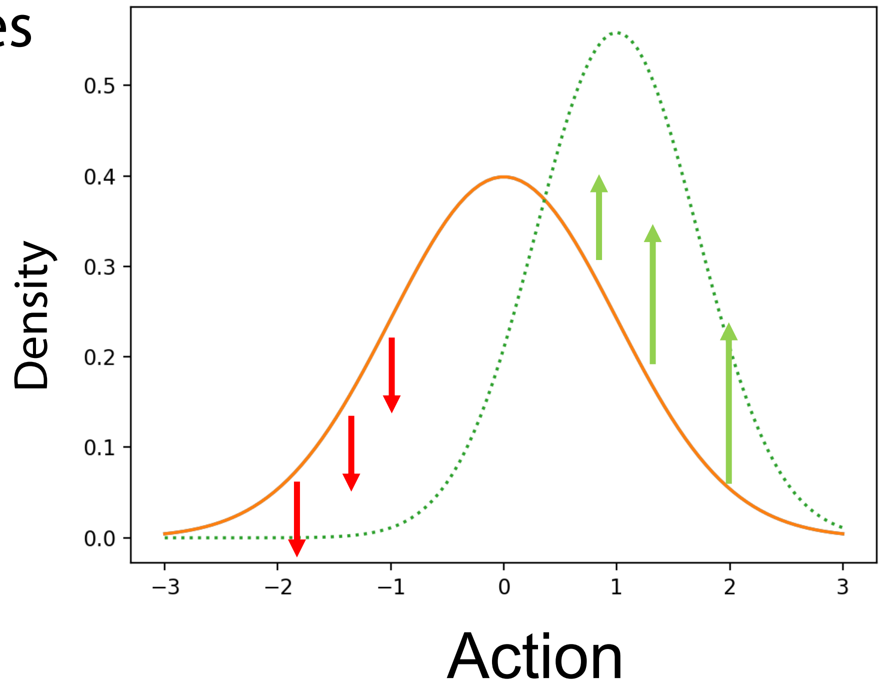
$$\approx \frac{1}{N} \sum_{i=0}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \sum_{t'=0}^{T} r(s_{t'}^i, a_{t'}^i)$$ (approximating using samples)

What does this mean?

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

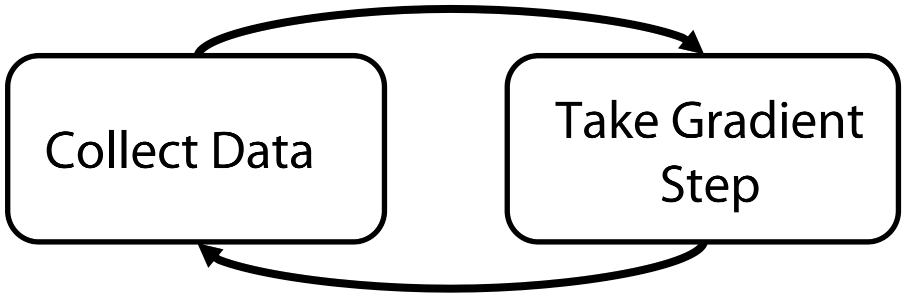
Increase the likelihood of actions in high return trajectories





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

Continuous Policy Gradient - Pseudocode

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

Pseudocode example (with continuous actions):

```
# 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:
pred_mean, pred_cov = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = -gaussian_log_likelihood(actions, mean= pred_mean, cov = pred_cov)
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, returns)
loss = tf.reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

Discrete Policy Gradient - Pseudocode

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

Pseudocode example (with discrete actions):

```
# 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)
```

How is this related to supervised learning?

Reinforcement Learning

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

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

Supervised Learning

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

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

PG = select good data + increase likelihood of selected data

Lecture Outline

Recap

↓

Imitation Learning: Improvements – Multimodality

↓

Policy Gradient

↓

Improving Policy Gradient

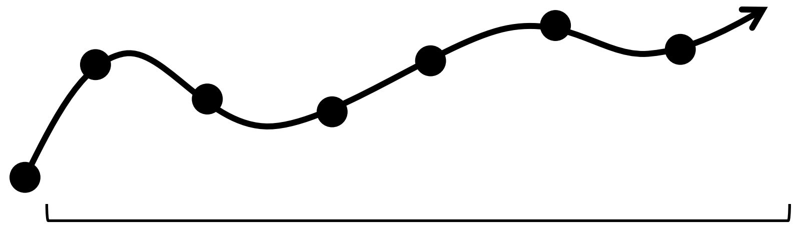
What makes policy gradient challenging?

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

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

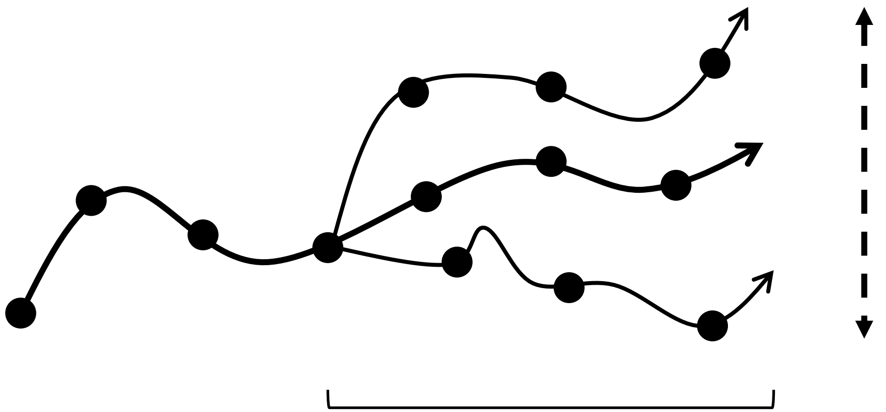
High variance estimator!!

What we do



Single sample estimate

What we actually want



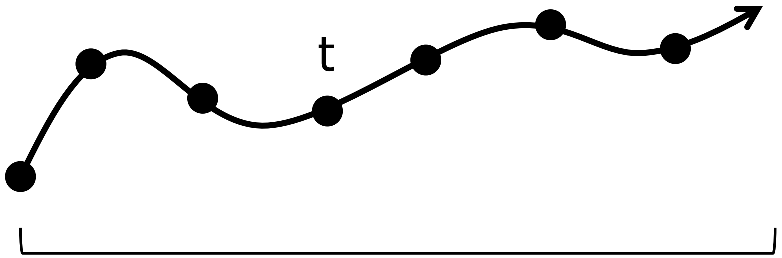
Averaged return estimate

Hard to tell what matters without many samples

Variance Reduction with Causality

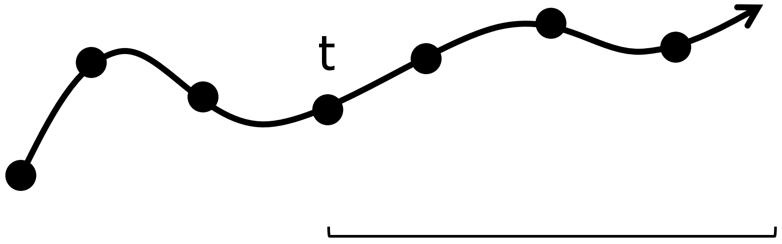
Idea: Trajectory returns depend on past and future, but we only care about the future, since actions cannot affect the past. Instead, consider **"return-to-go"**

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

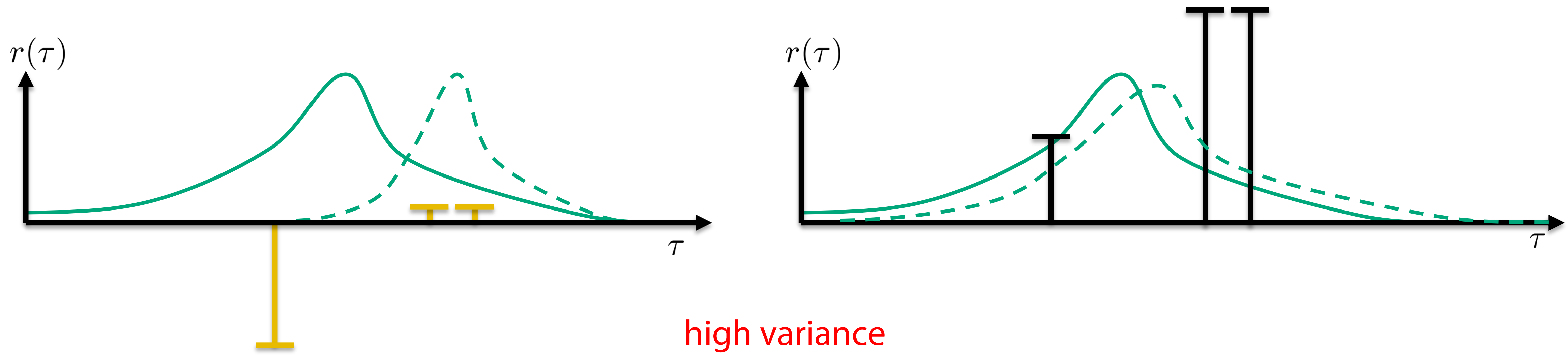


Ignore past terms ⇓

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



Can we reduce variance further?



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!

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

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

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

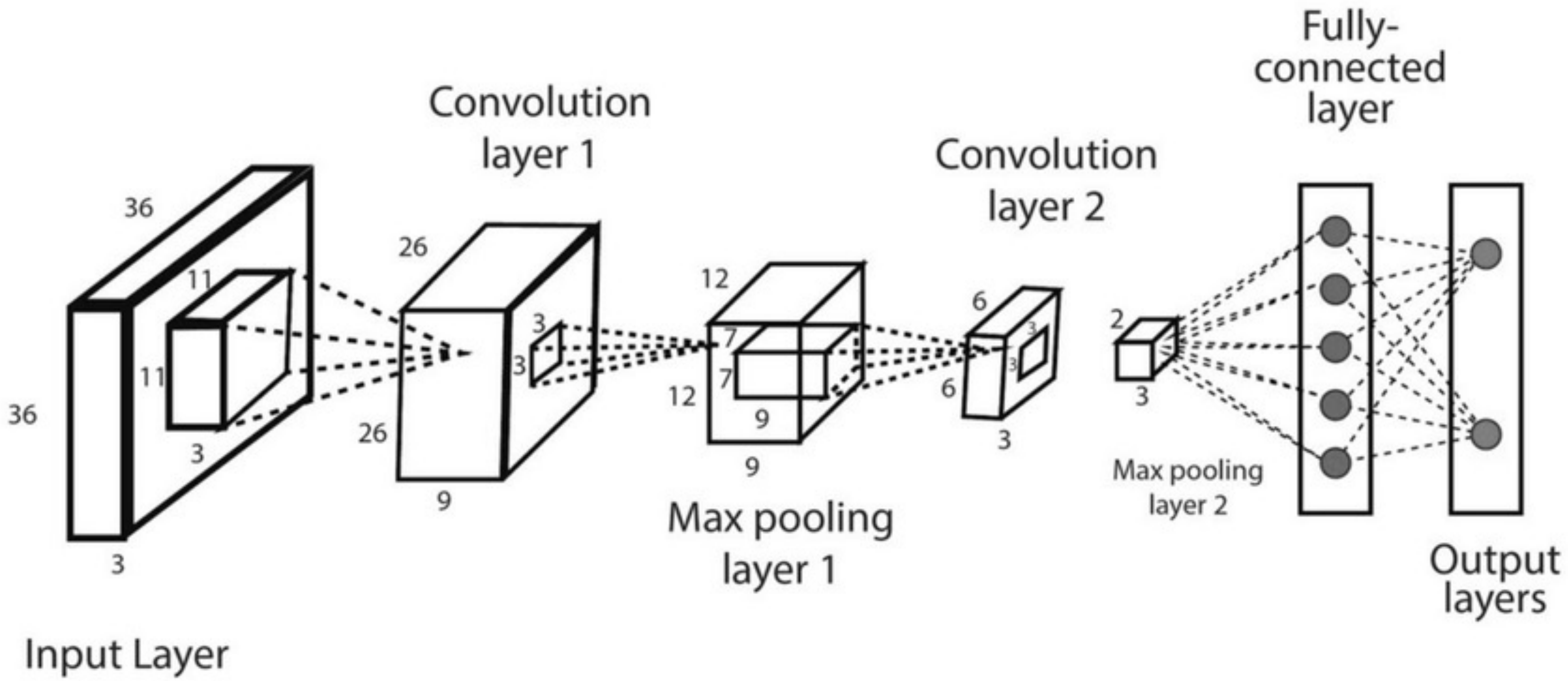
$$= \int p(s_t)b(s_t)\int \nabla_\theta \pi_\theta(a_t|s_t) da_t ds_t$$

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

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

Further Improvements on Policy Gradient

Control Step Size

Prevent excessive step size

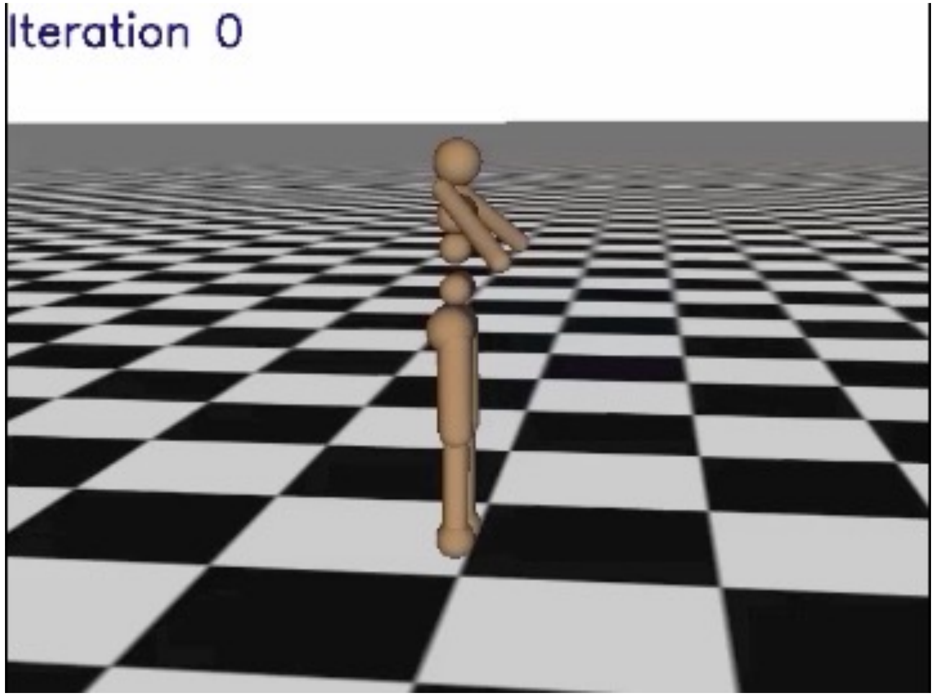
Proximal Policy Optimization

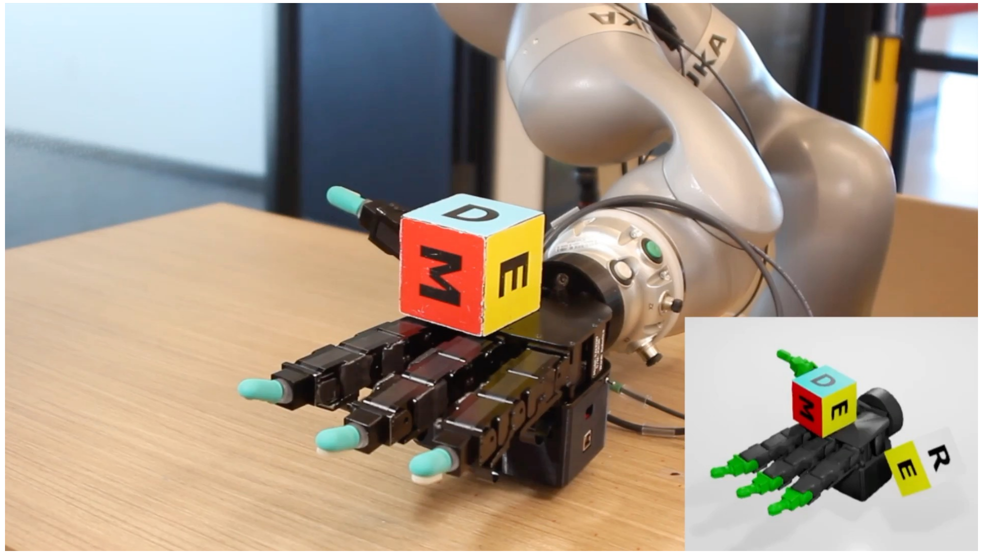
$$\mathcal{L}(s, a, \theta_i, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_i}(a|s)} A(s,a), \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_i}(a|s)}, 1-\epsilon, 1+\epsilon\right) A(s,a)\right)$$

Don't let the policy change too much

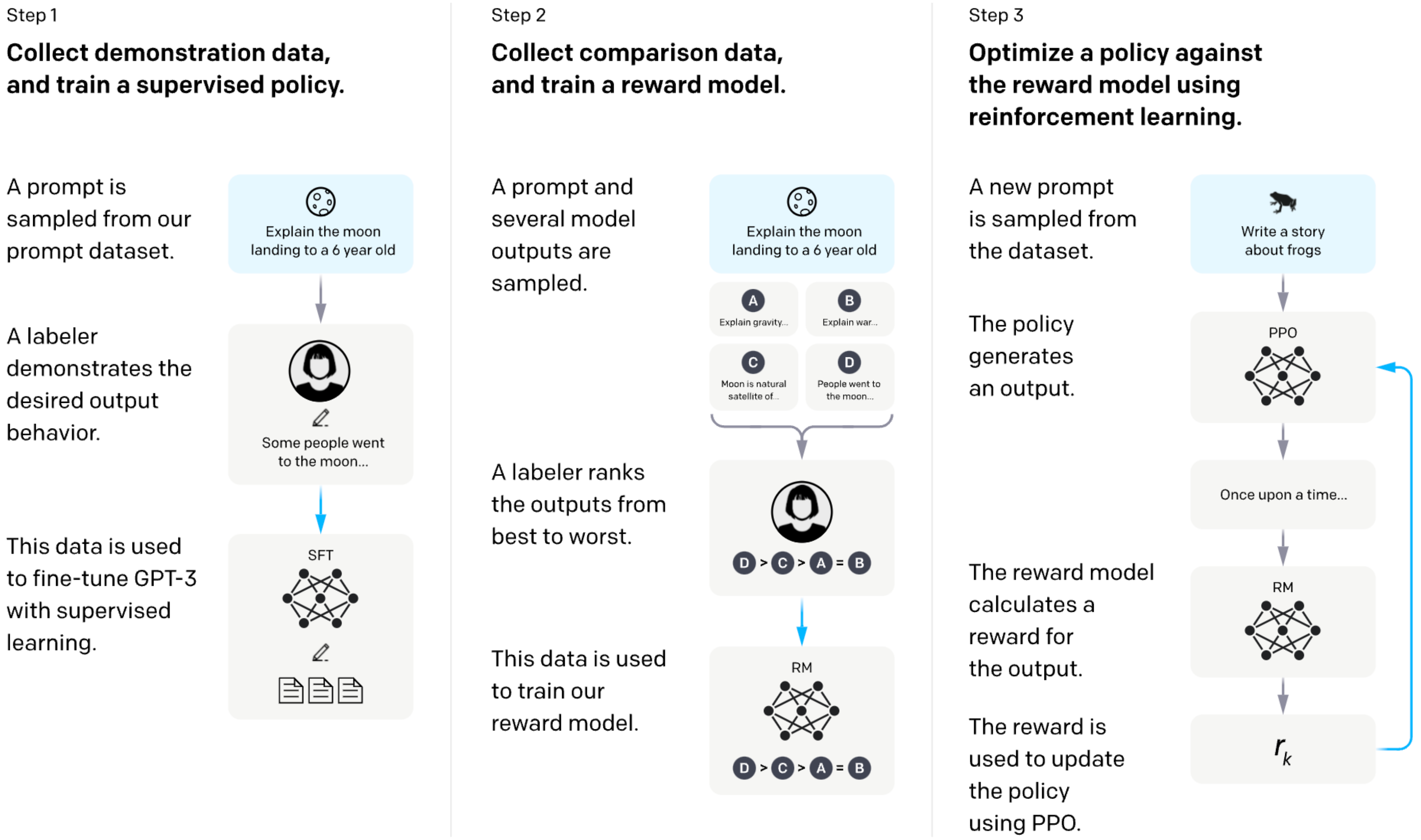
This allows for more gradient steps and stable updates

Advanced Policy Gradient in Action: Sim Control





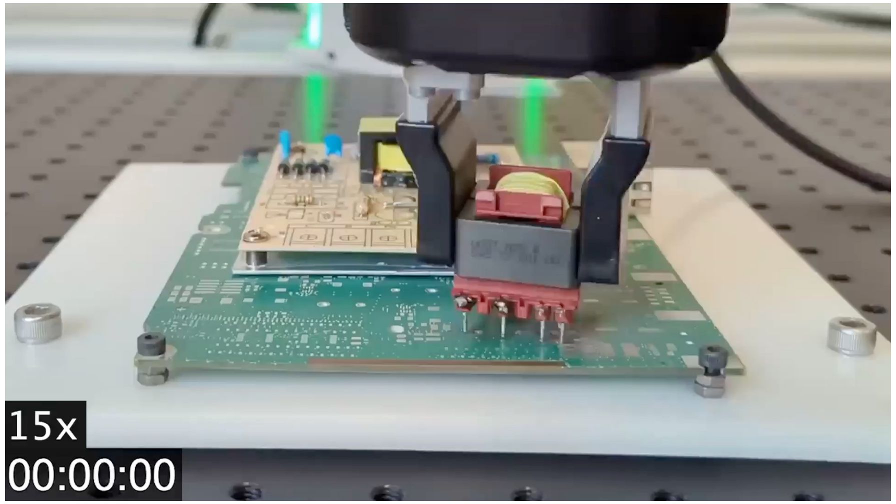
Advanced Policy Gradient in Action: LLMs



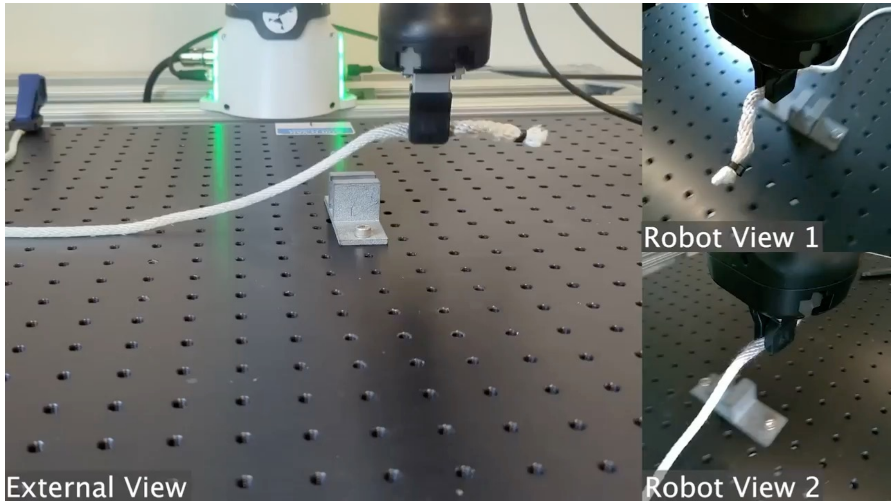
Policy Gradient (ish) in Action: Real-World Robots

With small improvements in estimation - can work on robots!

Smith et al



Luo et al



Luo et al

Lecture Outline

Recap

↓

Imitation Learning: Improvements – Multimodality

↓

Policy Gradient

↓

Improving Policy Gradient

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
- Actor-Critic
- Model-Based RL