Imitation Learning

Introduction

- Imitation Learning
  - Also known as learning by demonstration, apprenticeship learning
- An expert demonstrates how to solve the task
  - Machine can also interact with the environment, but cannot explicitly obtain reward.
  - It is hard to define reward in some tasks.
  - Hand-crafted rewards can lead to uncontrolled behavior
- Two approaches:
  - Behavior Cloning
  - Inverse Reinforcement Learning (inverse optimal control)

Behavior Cloning

Behavior Cloning

Yes, this is supervised learning.

- Self-driving cars as example

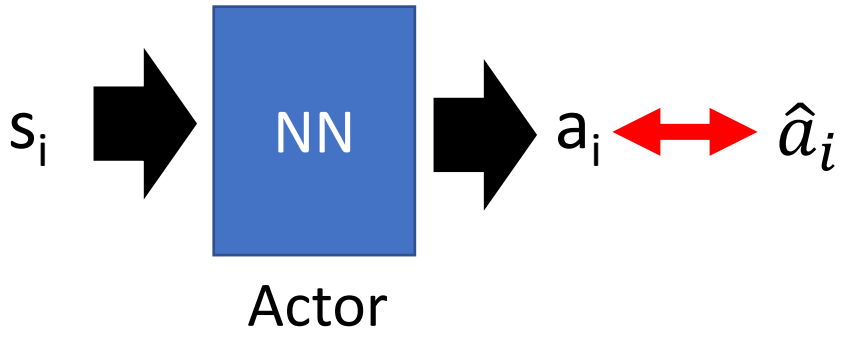
observation

Expert (Human driver): 向前

Machine: 向前

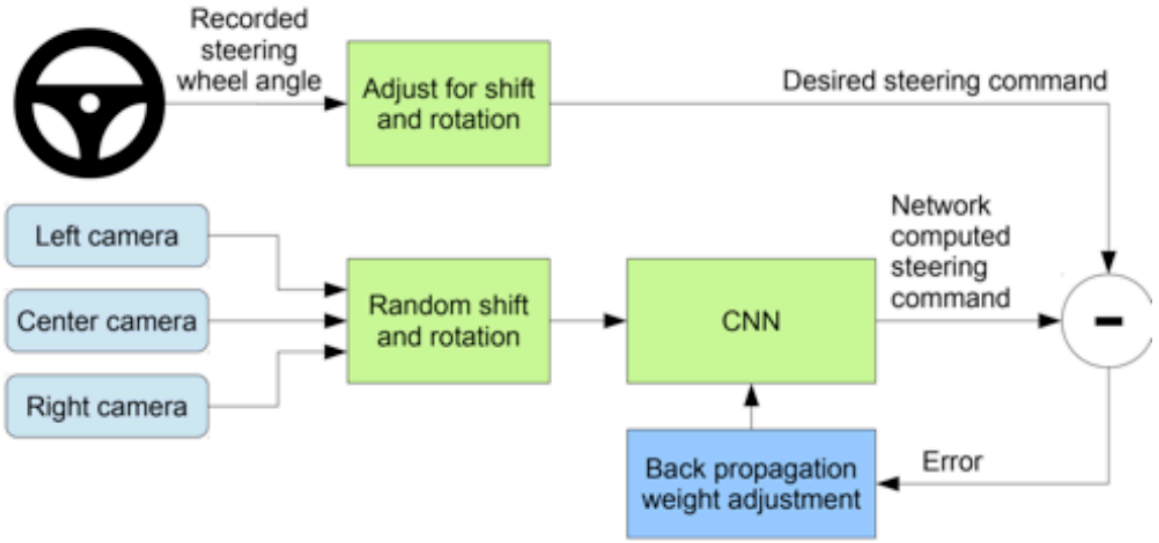
Training data:

$(s_1, \hat{a}_1)$
$(s_2, \hat{a}_2)$
$(s_3, \hat{a}_3)$
……



Case Study: End to End Learning for Self-Driving Cars

- **72 hours of driving data as training data**



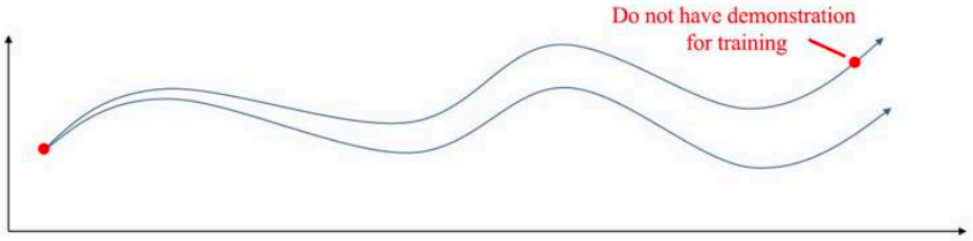
End-to-End Deep Learning for Self-Driving Cars
Mariusz Bojarski, Ben Firner, Beat Flepp, Larry Jackel, Urs Muller, Karol Zieba and Davide Del Testa | August 17, 2016

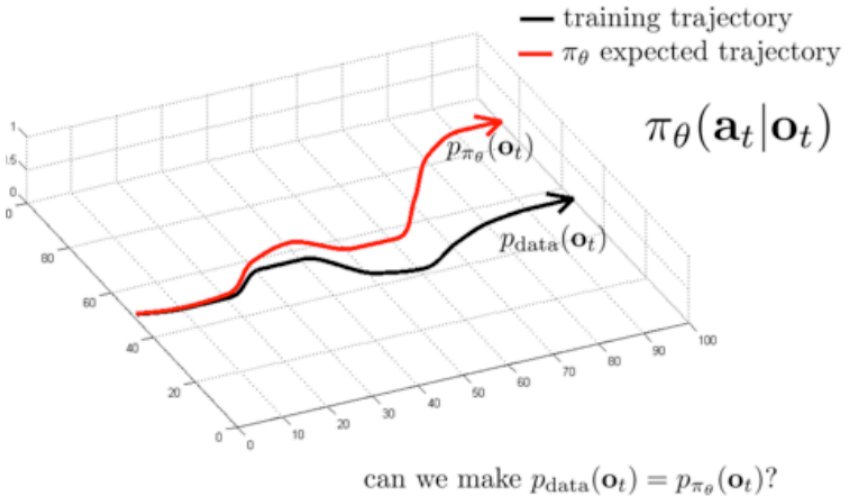
Limitations of Supervised Imitation Learning

- The trained policy run into the off-course situations
- Mistakes grow quadratically in T: don't learn how to recover from failure

$$\mathbf{J}(\hat{\pi}_{\mathbf{sup}}) \leq \mathbf{T}^2 \varepsilon$$

[Ross 2012]





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

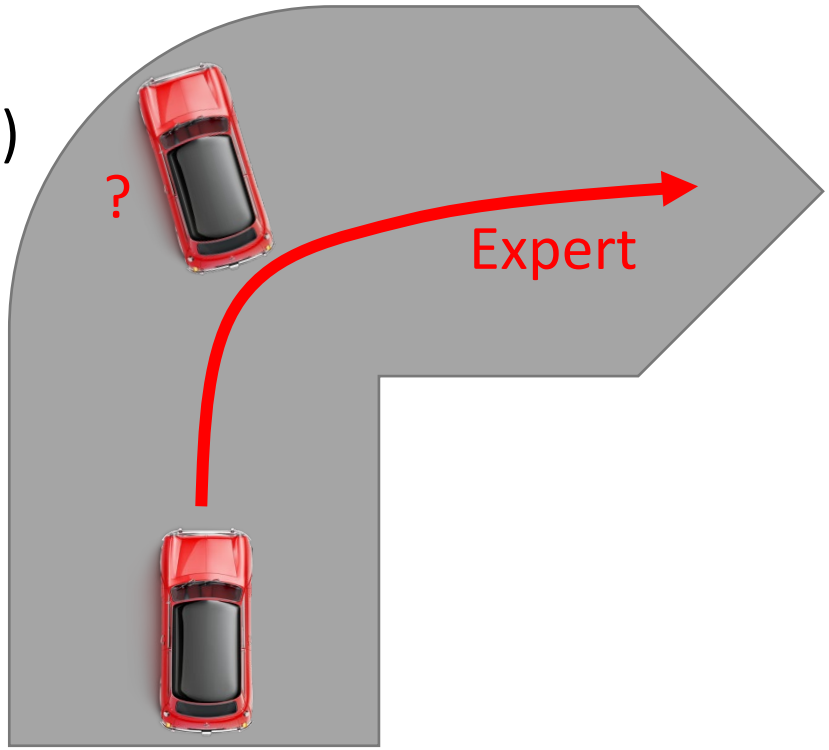
Behavior Cloning

- Problem

  Expert only samples limited observation (states)

  Let the expert in the states seem by machine

  Dataset Aggregation



Behavior Cloning

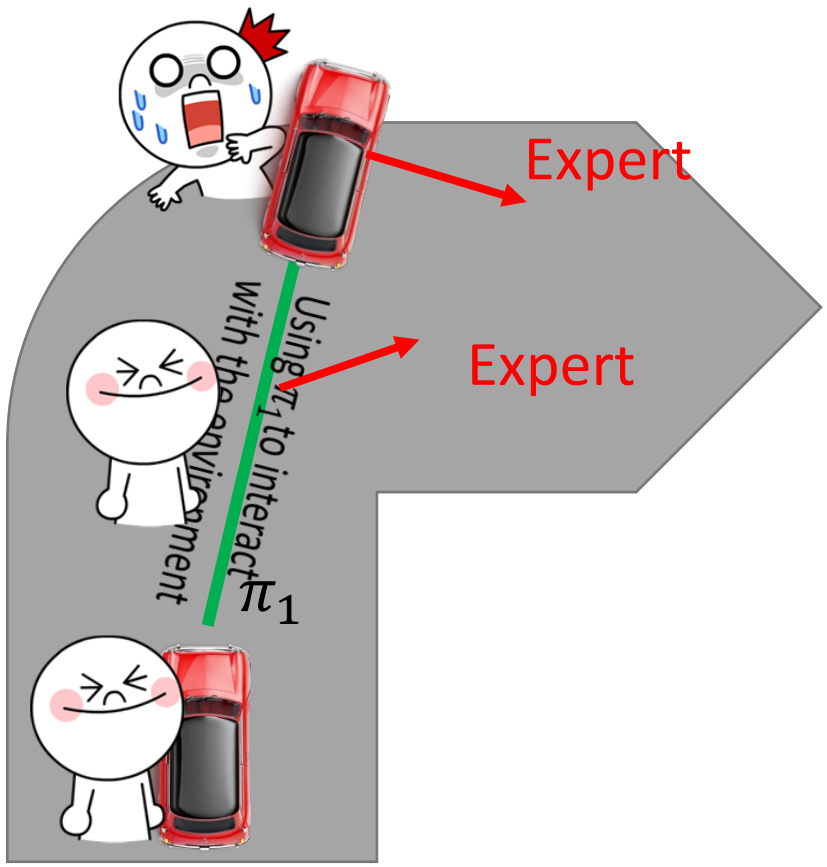
- Dataset Aggregation

  Get actor π_1 by behavior cloning

  Using π_1 to interact with the environment

    Ask the expert to label the observation of π_1

  Using new data to train π_2



DAgger: Dataset Aggregation

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: Dataset Aggregation

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

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

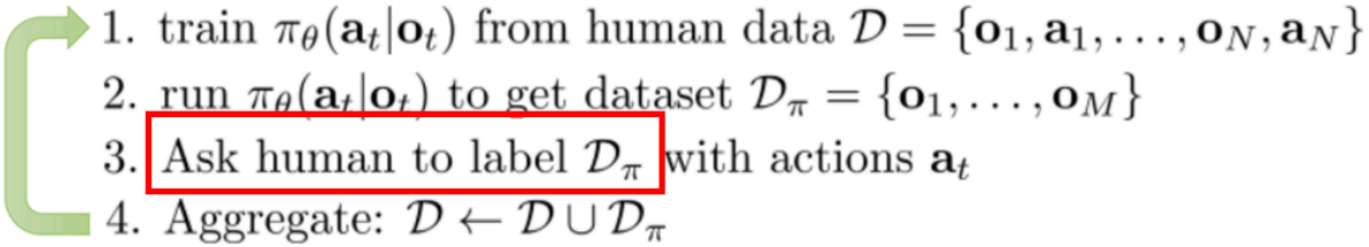
but need labels $\mathbf{a}_t$!

1. train $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \ldots, \mathbf{o}_N, \mathbf{a}_N\}$
2. run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
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, Gordon, and Bagnell 2011

Limitation with DAgger

- Need to query expert many times during the training
- Very expansive if it is human expert



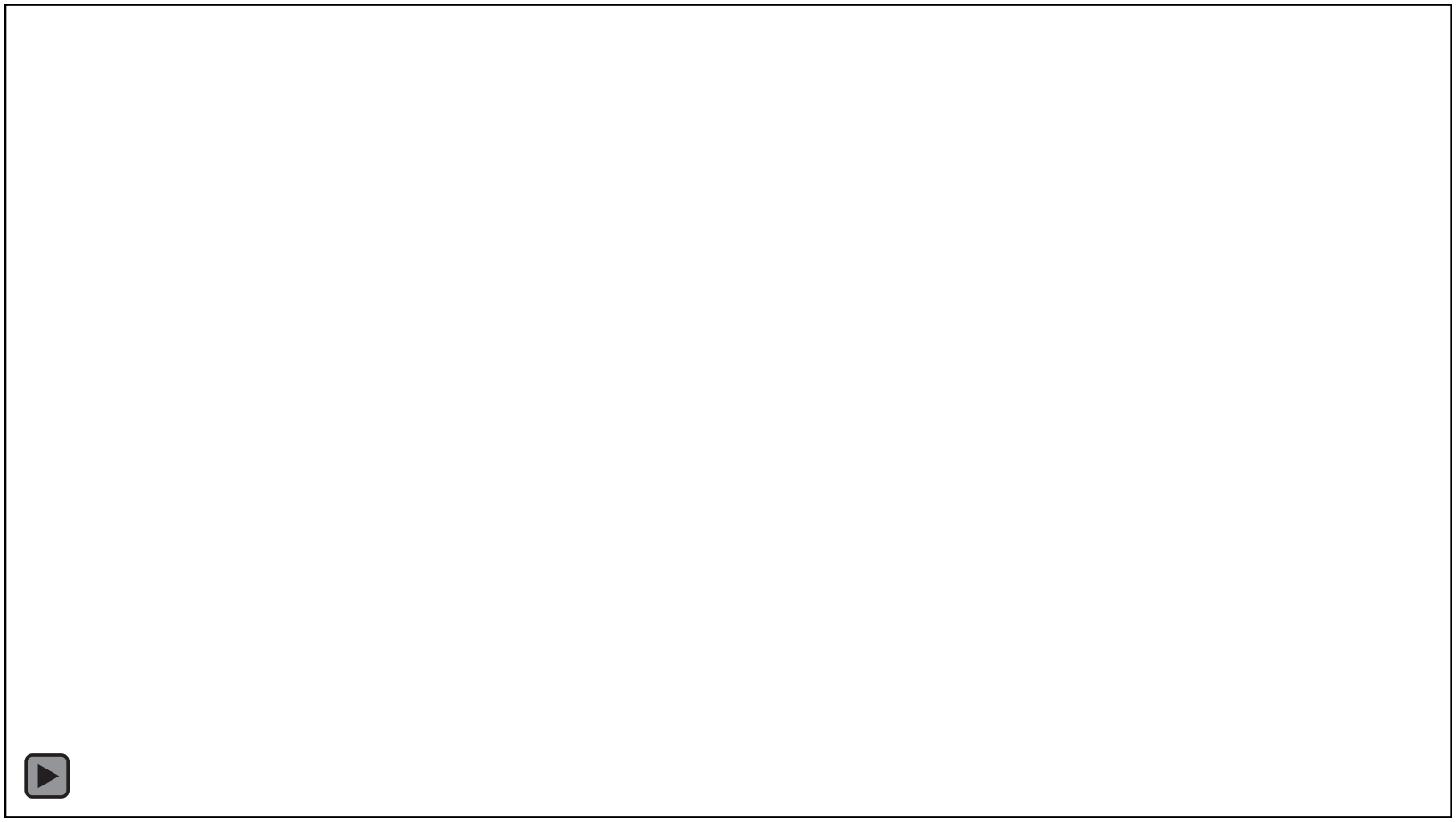
DAgger with Other Expert Algorithms

- Learn (quickly) to imitate other slow algorithms
- Game playing: we can compute suboptimal expert behavior with brute-force search offline, then use it during training time for a policy function to approximate
- Discrete optimizers: Many discrete optimization problems such as shortest path are very slow. Then use imitation learning to try to build shortest path optimizers that will run sufficiently efficiently in real time.

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

Behavior Cloning

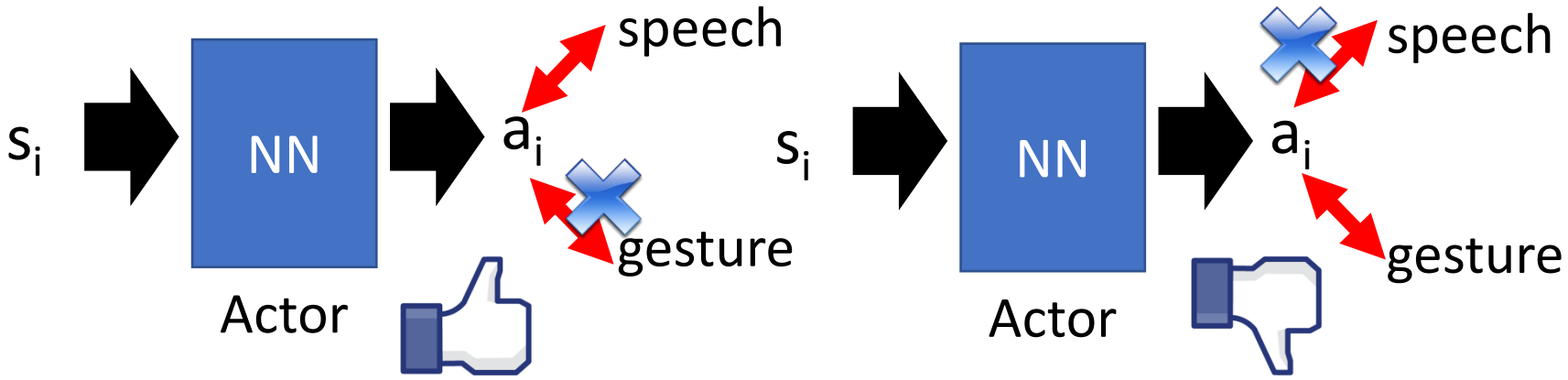
The agent will copy every behavior, even irrelevant actions.



https://www.youtube.com/watch?v=j2FSB3bseek

Behavior Cloning

- Major problem: if machine has limited capacity, it may choose the wrong behavior to copy.



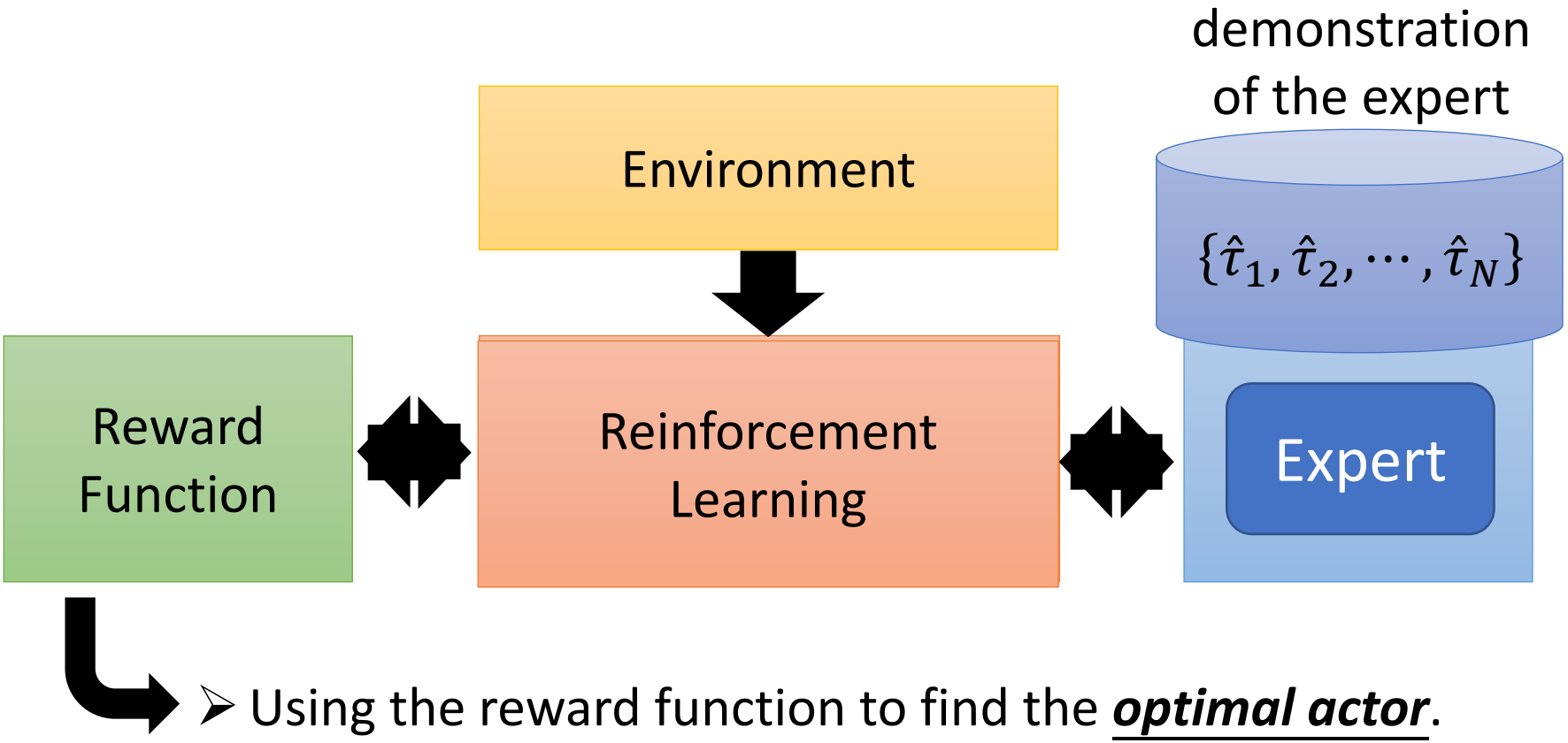
- Some behavior must copy, but some can be ignored.
  - Supervised learning takes all errors equally

Mismatch

- In supervised learning, we expect training and testing data have the same distribution.
- In behavior cloning:
  - Training: $(s, a) \sim \hat{\pi}$ (expert)
    - ***Action a taken by actor influences the distribution of s***
  - Testing: $(s', a') \sim \pi^*$ (actor cloning expert)
    - If $\hat{\pi} = \pi^*$, (s, a) and (s', a') from the same distribution
    - If $\hat{\pi}$ and π^* have difference, the distribution of s and s' can be very different.

Inverse Reinforcement Learning (IRL)

Inverse Reinforcement Learning

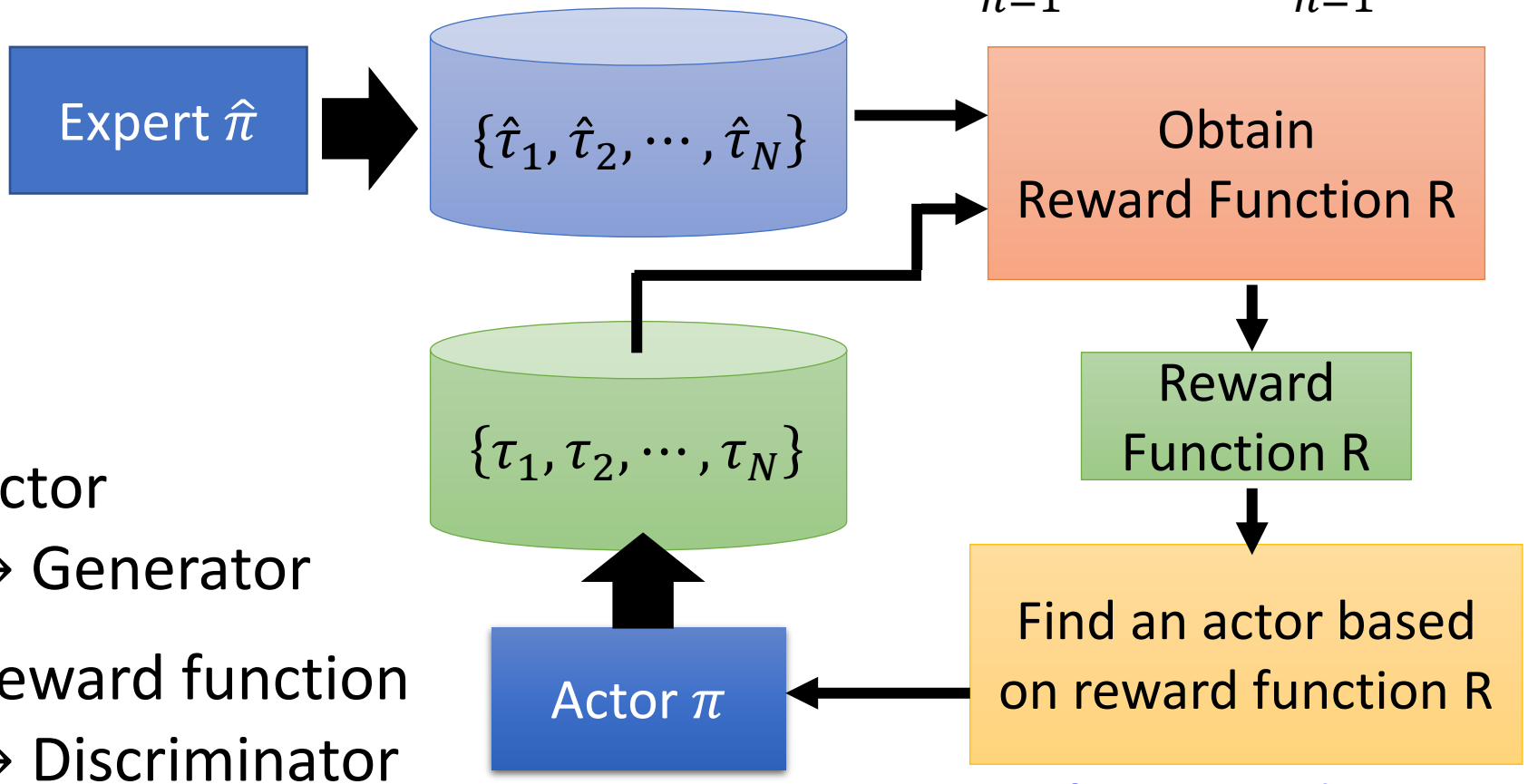


- Using the reward function to find the ***optimal actor***.
- Modeling reward can be easier. Simple reward function can lead to complex policy.

Framework of IRL

The expert is always the best.

$$\sum_{n=1}^{N} R(\hat{\tau}_n) > \sum_{n=1}^{N} R(\tau)$$

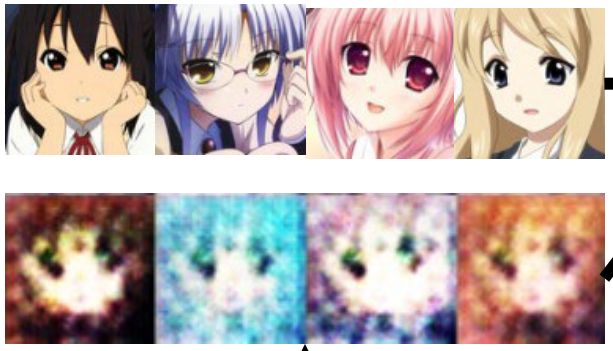


Actor
→ Generator

Reward function
→ Discriminator

By Reinforcement learning

GAN



High score for real,
low score for generated

D

Find a G whose output
obtains large score from D

G

IRL

Expert

$\{\hat{\tau}_1, \hat{\tau}_2, \cdots, \hat{\tau}_N\}$

$\{\tau_1, \tau_2, \cdots, \tau_N\}$

Larger reward for $\hat{\tau}_n$,
Lower reward for τ

Reward
Function

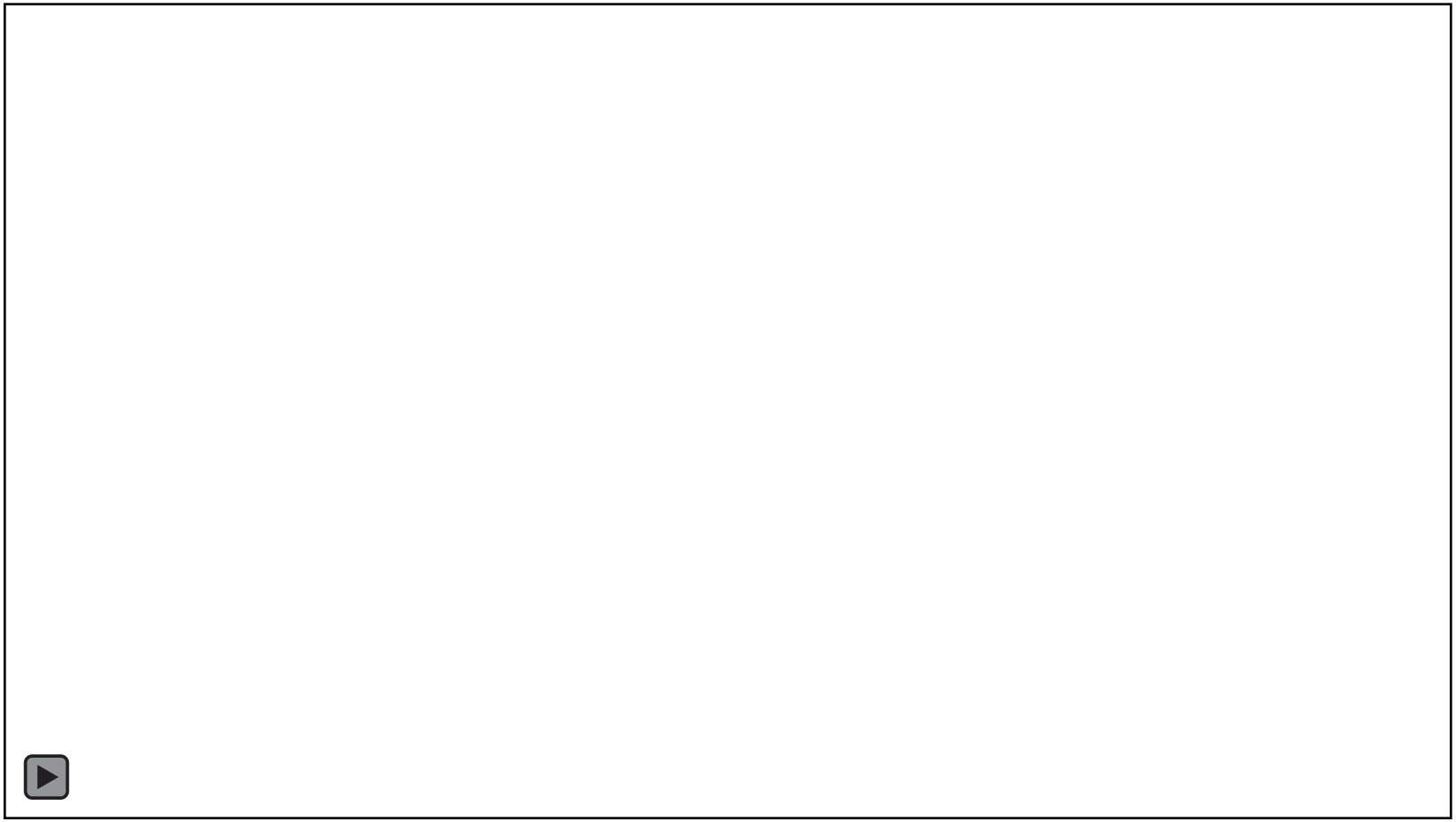
Find a Actor obtains
large reward

Actor

Robot

Chelsea Finn, Sergey Levine, Pieter Abbeel, ” Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization”, ICML 2016

http://rll.berkeley.edu/gcl/



Problems with Imitation Learning

- Humans need to provide data, which is typically in a limited amount
- Humans are not good at providing some kinds of actions
- Human can learn autonomously (unsupervised learning)




Imitation Learning vs. Reinforcement Learning

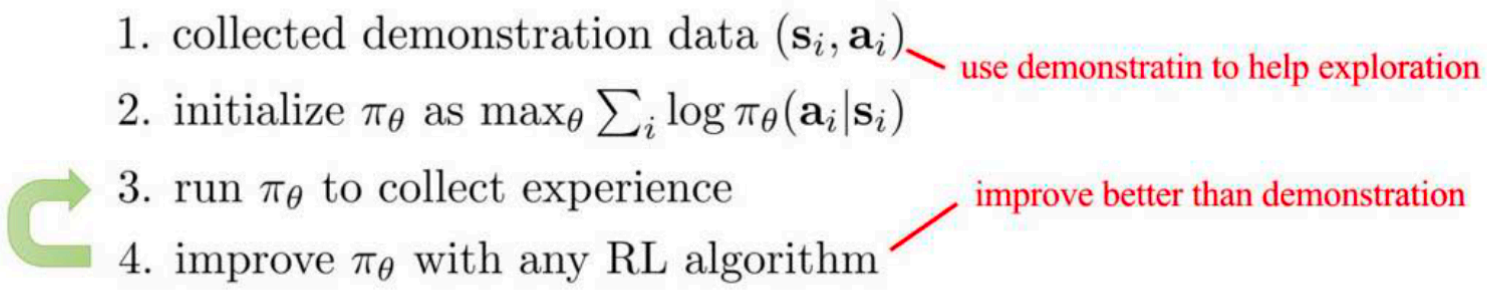
Imitation Learning
- Requires demonstrations
- Issue of distributional shift
- Simple stable supervised learning
- Only as good as the demo

Reinforcement Learning
- Requires reward function
- Must address exploration
- Potentially non-convergent
- Can become superhuman good

Can we get the best of both worlds?
What if we can have both demonstrations and rewards

Simplest Combination: Pretrain & Finetune

- Use the expert demonstration to initialize a policy (overcome exploration)
- Then apply RL to improve the policy and learn to deal with those off-course scenarios and go beyond performance of the demonstrator



Simplest Combination: Pretrain & Finetune

Pretrain & finetune

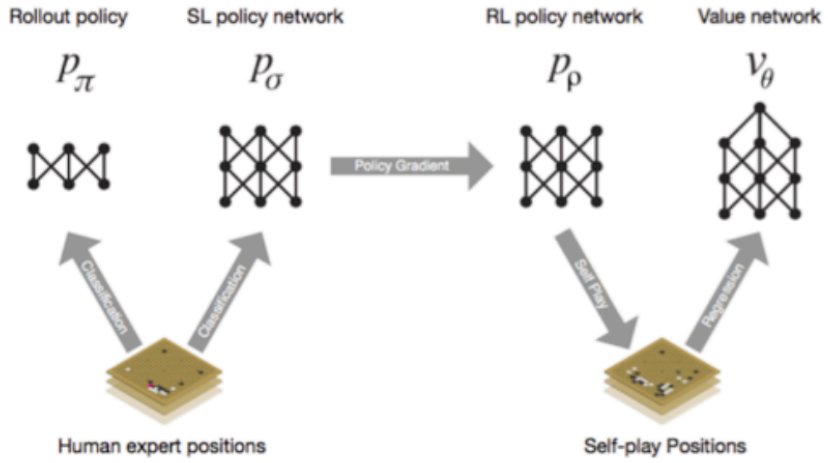
1. collected demonstration data $(\mathbf{s}_i, \mathbf{a}_i)$
2. initialize π_θ as $\max_\theta \sum_i \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)$
3. run π_θ to collect experience
4. improve π_θ with any RL algorithm

vs. DAgger

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

Pretrain & Finetune for AlphaGo

- SL Policy Network: Train the policy network on 30 million moves from games played by human experts (could predict the human move 57% of the time)
- RL Policy Network: Then finetune the weights with policy gradients



David Silver, et al. Mastering the ancient game of Go with Machine Learning. Nature 2016

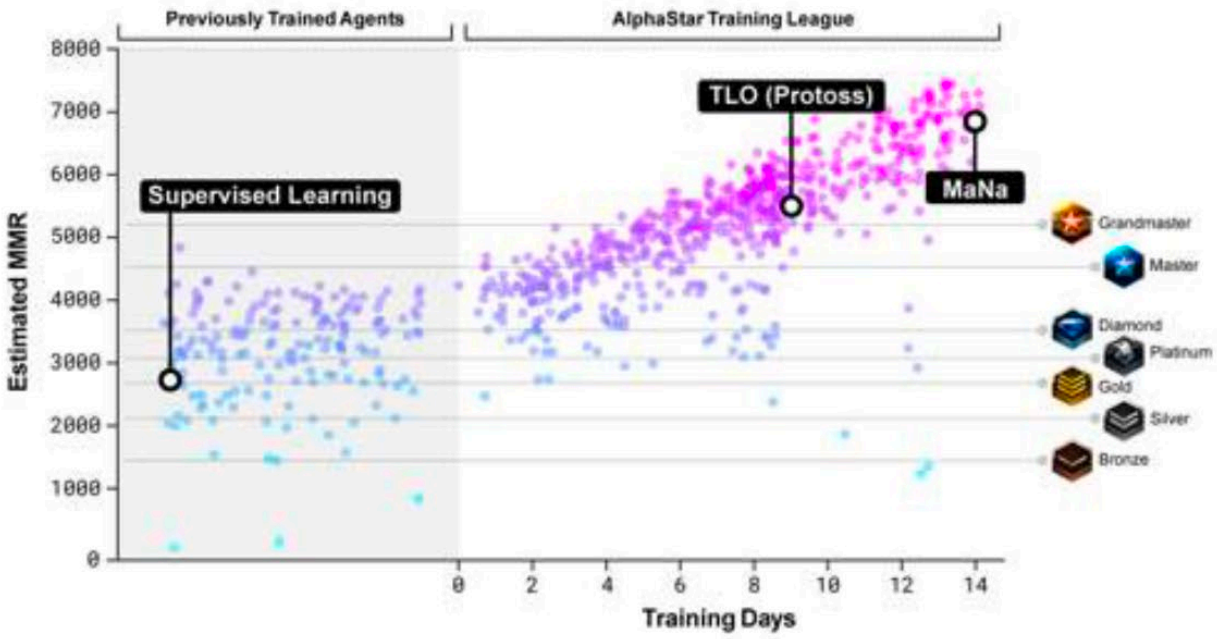
Pretrain & Finetune for Starcraft2

- Supervised learning from anonymised human games released by Blizzard.
- This allowed AlphaStar to learn, by imitation, the basic micro and macro-strategies used by players on the StarCraft ladder.
- This initial agent defeated the built-in “Elite” level AI - around gold level for a human player - in 95% of games.

https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/

Pretrain & Finetune for Starcraft2

- Population-based and multi-agent reinforcement learning



https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/

Problem with the Pretrain & Finetune

1. collected demonstration data $(\mathbf{s}_i, \mathbf{a}_i)$
2. initialize π_θ as $\max_\theta \sum_i \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)$
3. run π_θ to collect experience ← can be very bad (due to distribution shift)
4. improve π_θ with any RL algorithm ← first batch of (very) bad data can destroy initialization

Can we avoid forgetting the demonstrations?

Solution: Off-policy Reinforcement Learning

- Off-policy RL can use any data
- We can use demonstrations as off-policy samples
  - Since demonstrations are provided as data in every iteration, they are never forgotten
  - Policy can still become better than the demos, since it is not forced to mimic them
- Off-policy policy gradient (with importance sampling)
- Off-policy Q-learning

Policy gradient with demonstrations

- Importance sampling on policy gradient

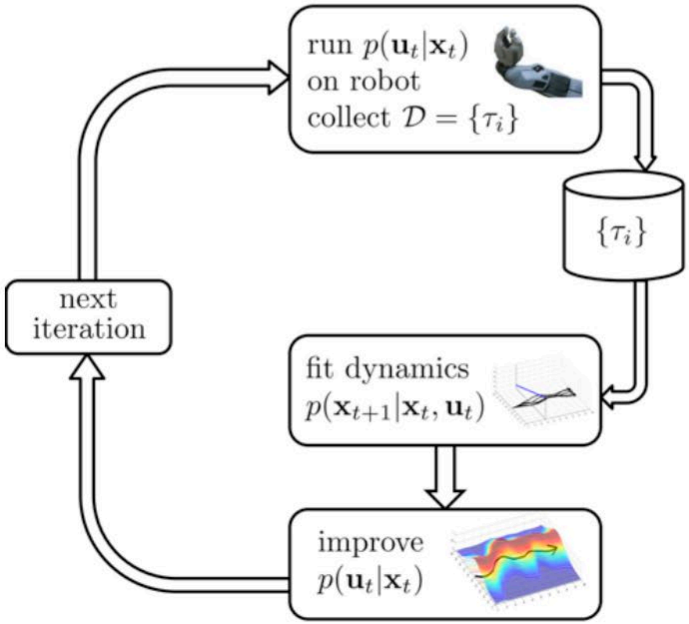
$$\nabla_\theta J(\theta) = \sum_{\tau \in \mathcal{D}} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \left(\prod_{t'=1}^{t} \frac{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{q(\mathbf{a}_{t'}|\mathbf{s}_{t'})} \right) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})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

from experience and demonstration

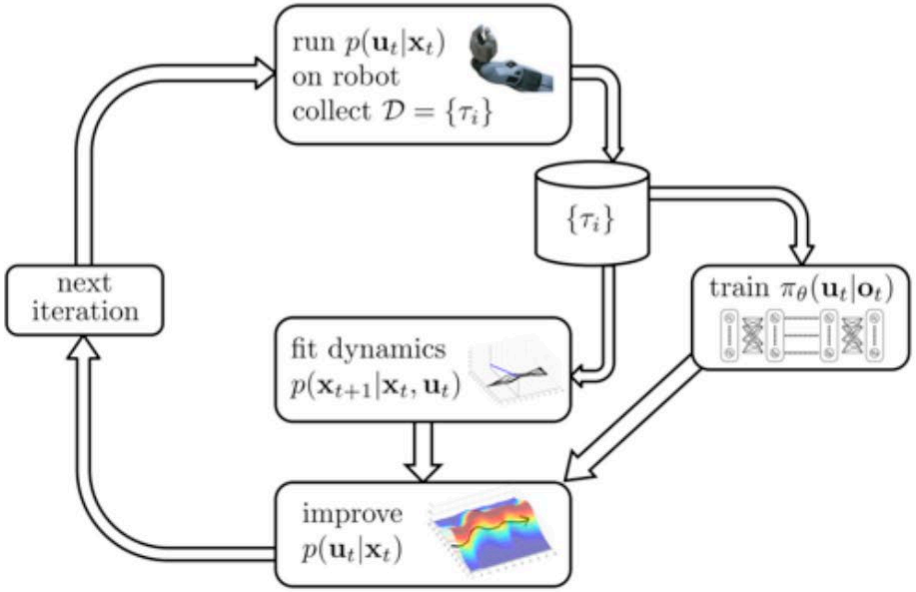
- Problem: which distribution did the demonstrations come from?
- Option 1: use supervised behavior cloning to approximate π_{demo}
- Option 2: assume Diract delta: $\pi_{demo}(\tau) = \frac{1}{N}\delta(\tau \in \mathcal{D})$

Guided Policy Search

Optimal control for trajectory optimization



Guide the training of policy network

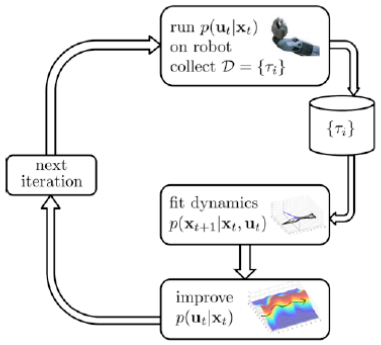


Review on model-based RL

- Optimal control for trajectory optimization (f is given)

$$\min_{u_1,\ldots,u_t} \sum_{t=1}^{T} c(x_t, u_t) \text{ s.t. } x_t = f(x_{t-1}, u_{t-1})$$

- If f is not given, we learn locally linear models and use these models to solve for a time-varying linear Gaussian controller

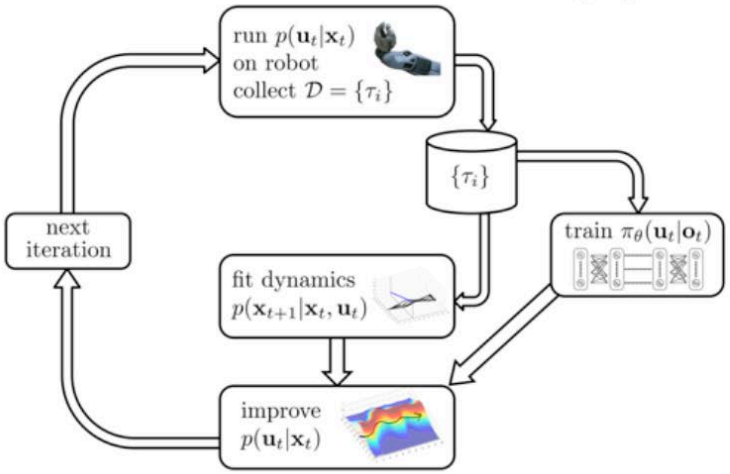


$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f(\mathbf{x}_t, \mathbf{u}_t))$$
$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{A}_t\mathbf{x}_t + \mathbf{B}_t\mathbf{u}_t$$

Guided policy search (GPS) formulation

$$\min_{u_1,\ldots,u_t} \sum_{t=1}^{T} c(x_t, u_t) \text{ s.t. } x_t = f(x_{t-1}, u_{t-1}) \text{ s.t. } u_t = \pi_\theta(x_t)$$

$$\min_{p(\tau)} \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{u}_t^T \lambda_t + \rho_t D_{KL}(p(\mathbf{u}_t|\mathbf{x}_t) \| \pi_\theta(\mathbf{u}_t|\mathbf{x}_t))]$$



Dual gradient descend

1. Start with some initial choice of λ (by λ, we include λ_t corresponding to each time step)
2. Assign $\tau \leftarrow \arg\min_\tau \mathcal{L}(\tau, \theta, \lambda)$.
3. Assign $\theta \leftarrow \arg\min_\theta \mathcal{L}(\tau, \theta, \lambda)$.
4. Compute $\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(\tau, \theta, \lambda)$. Take a gradient step $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$
5. Repeat steps 2-4.

Q-learning with demonstrations

- Q-learning is already off-policy, no need to bother with importance weighting
- Simple solution: throw demonstrations into the replay buffer