Algorithmic Human-Robot Interaction

(Shallow/Deep/Bottomless) Reinforcement Learning
Inverse Reinforcement Learning

CSCI 7000

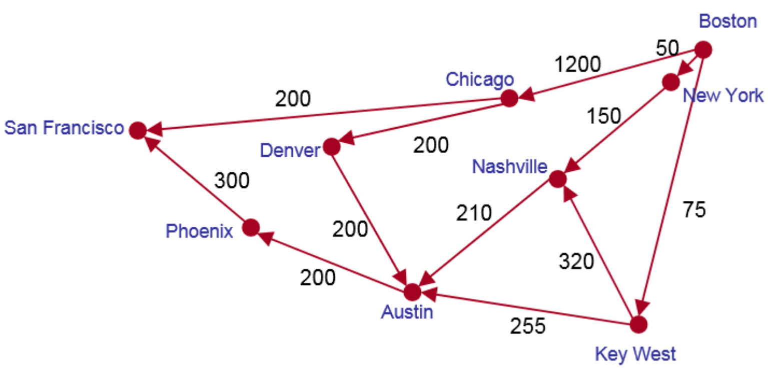
Prof. Brad Hayes

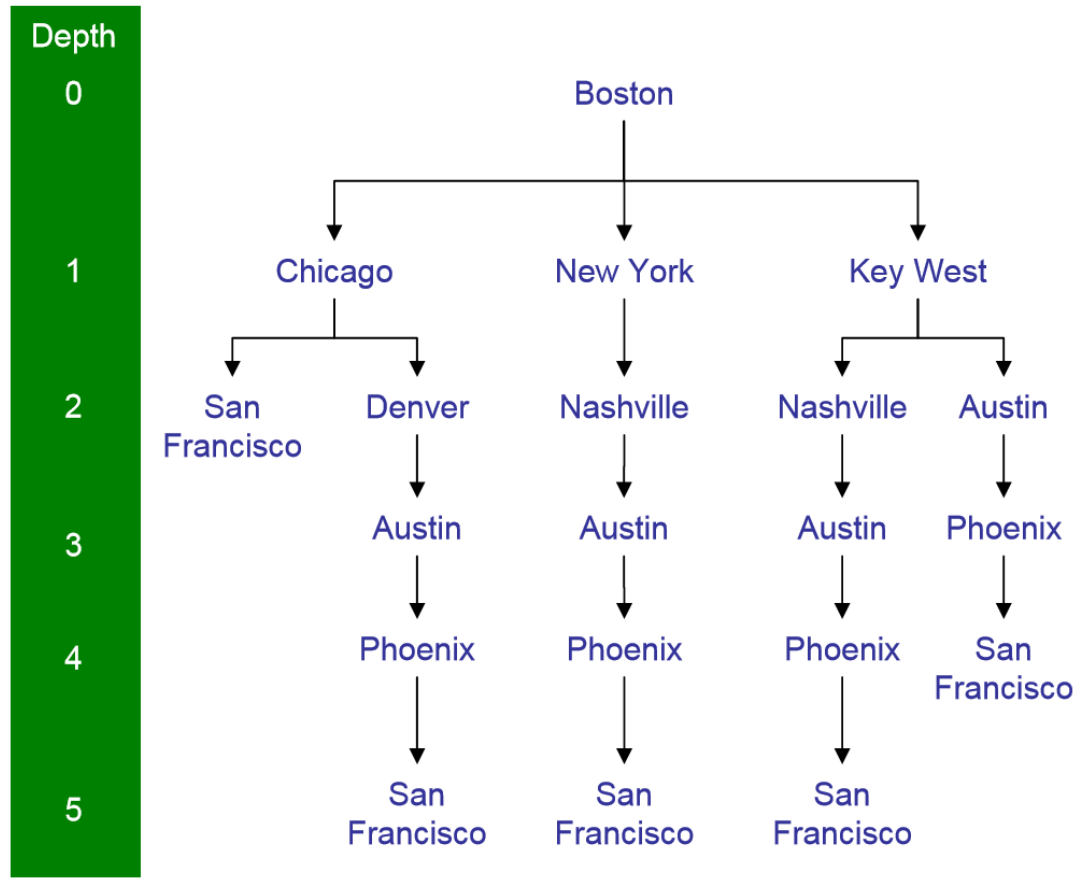
Computer Science Department

University of Colorado Boulder

Planning Recap

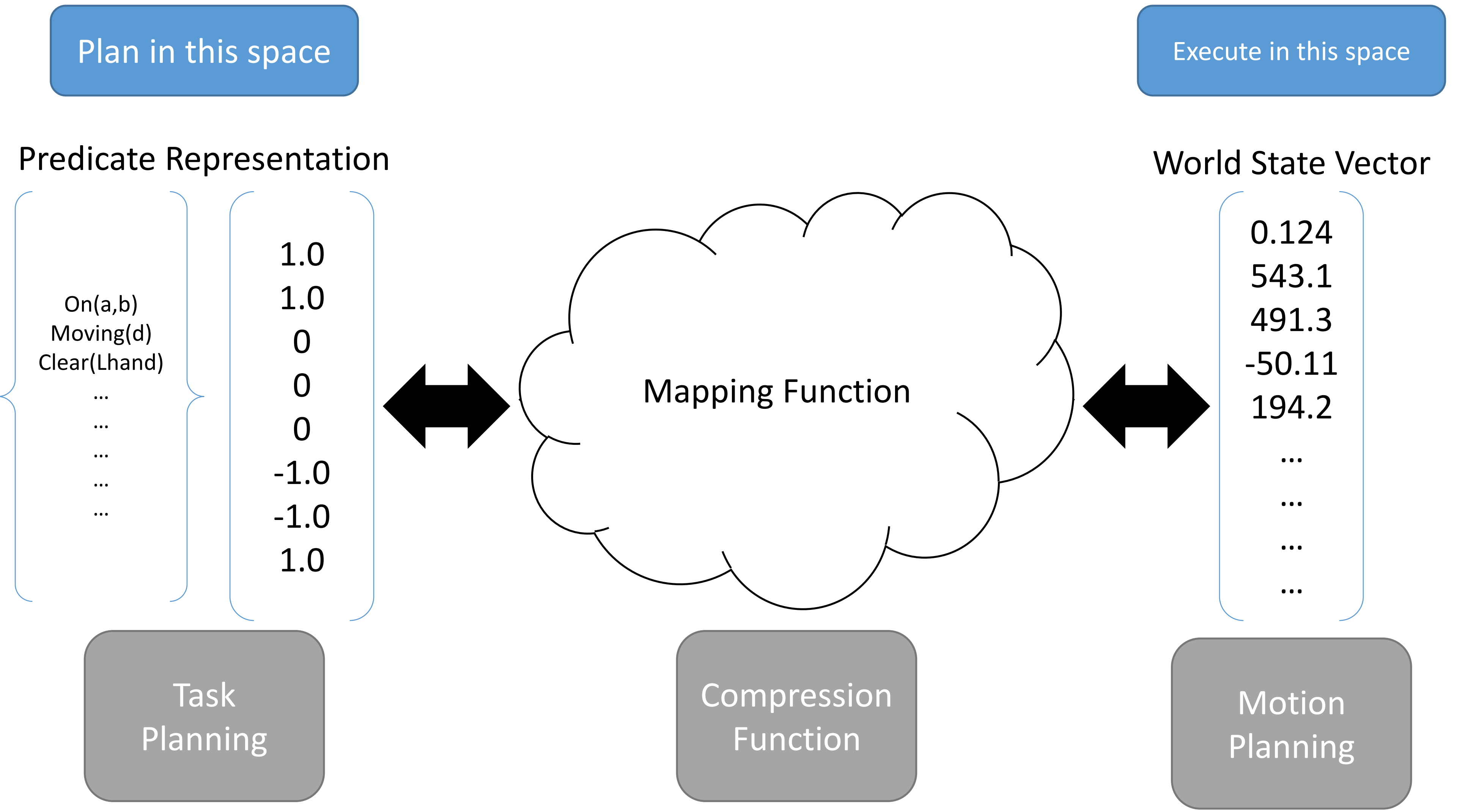
Search as a Problem-Solving Technique





Branching Factor b=3

Lifted vs. Grounded Planning



Strategies

- Breadth-first search with sampled action parameterizations
- Pre-determined action parameterizations
  - e.g., Place(object,x,y,z)
- Plan for abstract 'areas'
  - Lazy Grounding:
    Sample parameterizations and add that action to plan if a valid parameterization is found
  - Greedy Grounding:
    Sample parameterizations and add that specific parameterization to the plan once a valid one is found

Planning Heuristics

- FastForward Heuristic
  - Remove deletions from all operators and run planner
  - Resulting plan gives a heuristic for # steps from current state to goal
  - FF Plan can provide ordering constraints on actions, even within partially specified plans
- Learned Cost Function
  - Learning Real-Time A* (LRTA*)
  - Update heuristic dynamically: Value function V(s)
  - Connection back to reinforcement learning: **V(s) and Q(s,a)** for search heuristic

Admissible Heuristics and Pruning Strategies

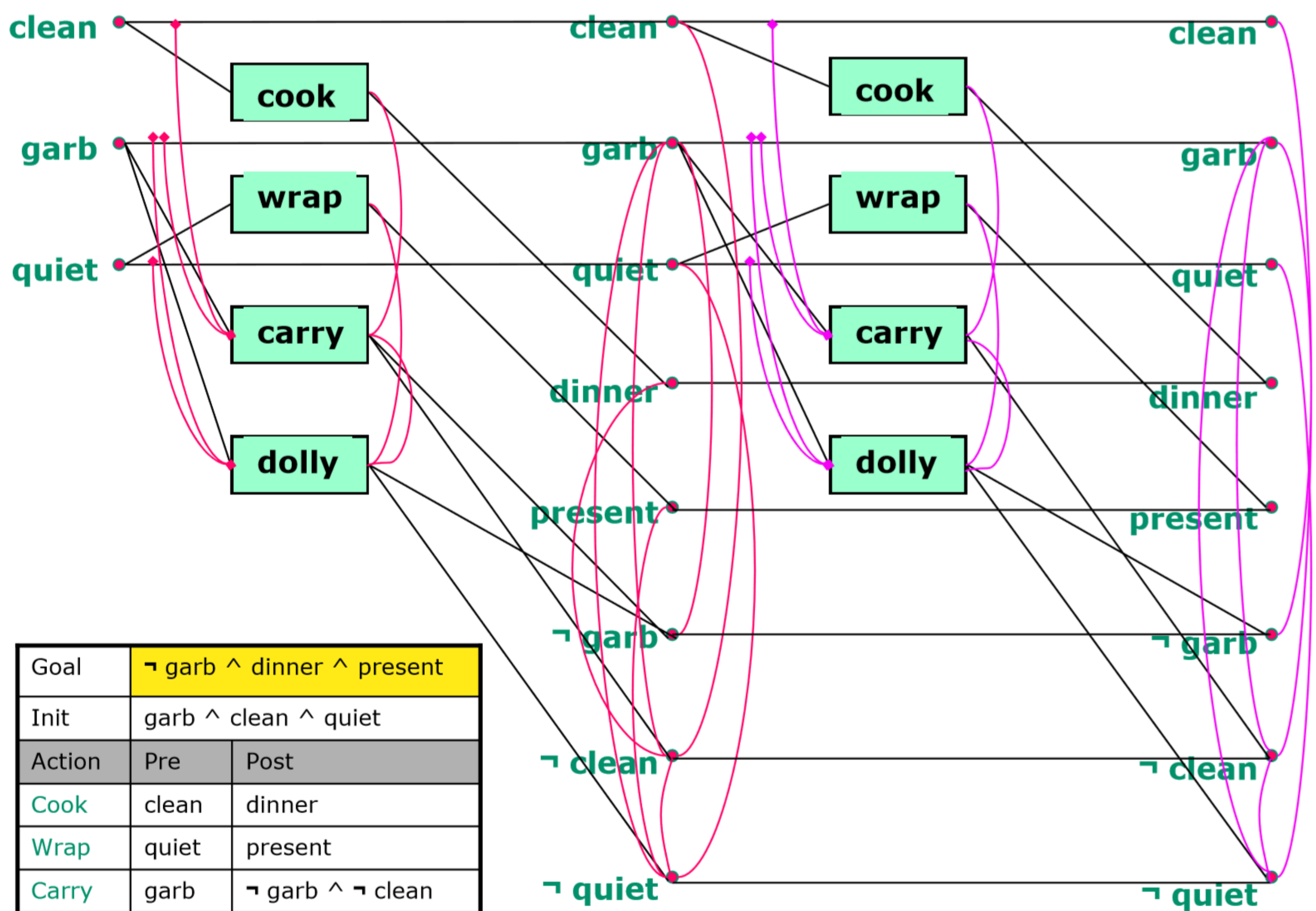
Admissible heuristics from delete-relaxation:

- h_{max} = i iff goal g is reachable in i steps from s
  - Admissible because of GRAPHPLAN Algorithm
- h_{FF} = number of **different** actions in $\pi(g)$

Search Pruning Strategies:

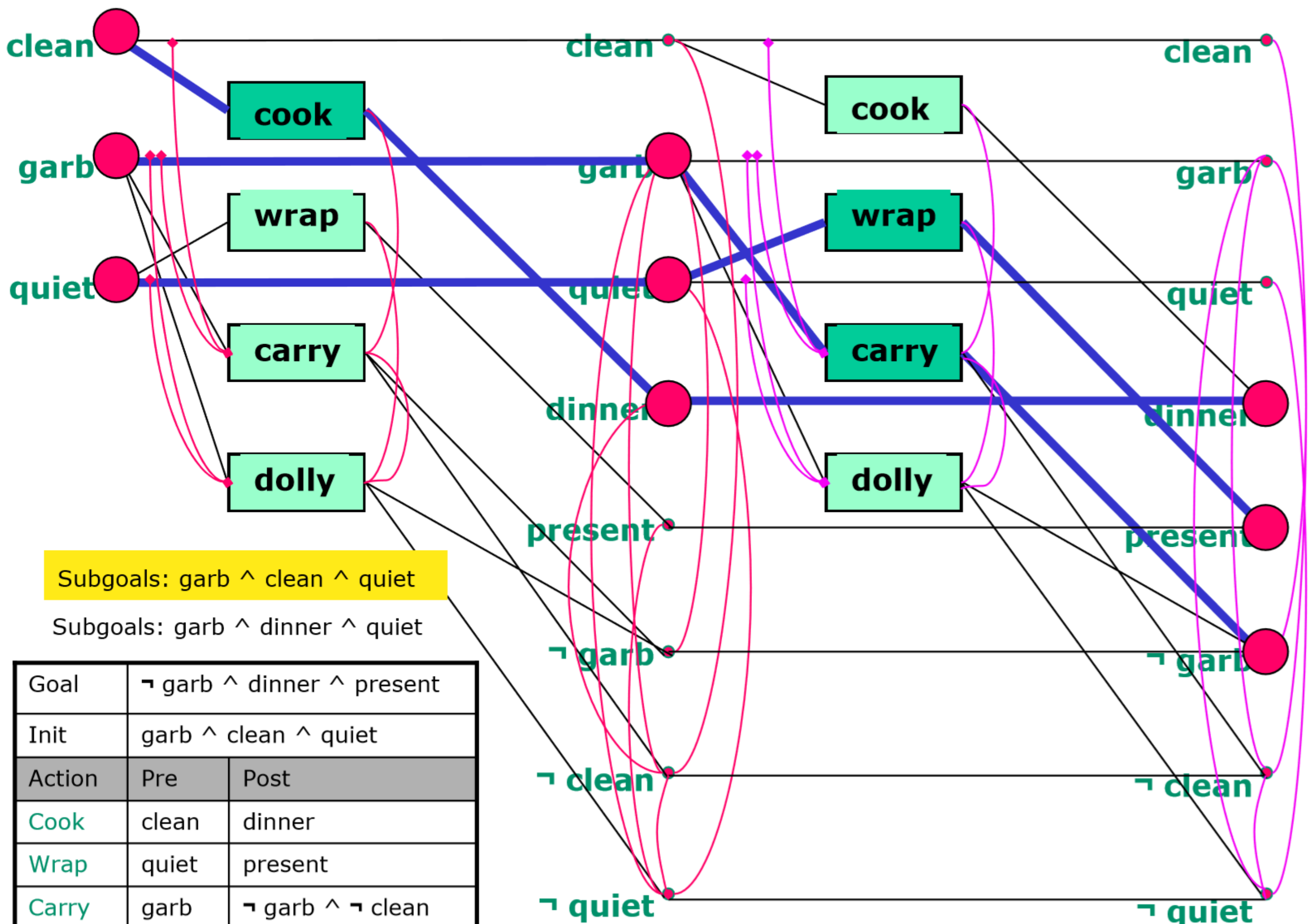
- Novelty: Assign each state $novelty(s) = i$ if i predicates take on a value never seen in path from s_0 to s
- IW-search (Iterative Width): Ignore any nodes at the search frontier with $novelty(s) > i$.
  - Call planner for increasing i until problem solved or i exceeds number of variables in the problem.

GRAPHPLAN Graph



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

GRAPHPLAN Graph Solution



Subgoals: garb ^ clean ^ quiet

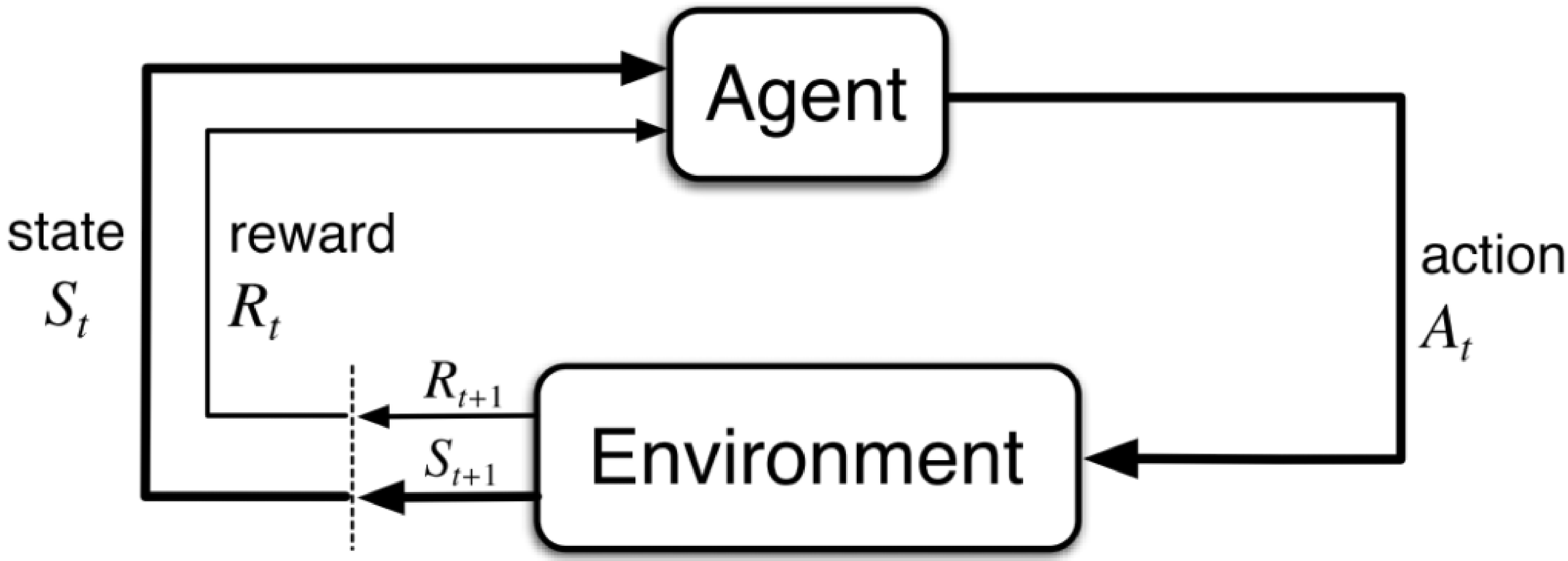
Subgoals: garb ^ dinner ^ quiet

Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Inverse Reinforcement Learning

Further Reading: ICML 2017 Deep RL Tutorial

Reinforcement Learning



Reinforcement Learning

Function: $O \rightarrow A$

$\mathbf{o}$ $\quad$ $\pi_\theta(\mathbf{a}|\mathbf{o})$ $\quad$ $\mathbf{a}$

$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy (fully observed)

$\mathbf{o}_t$ – observation

$\mathbf{s}_t$ – state

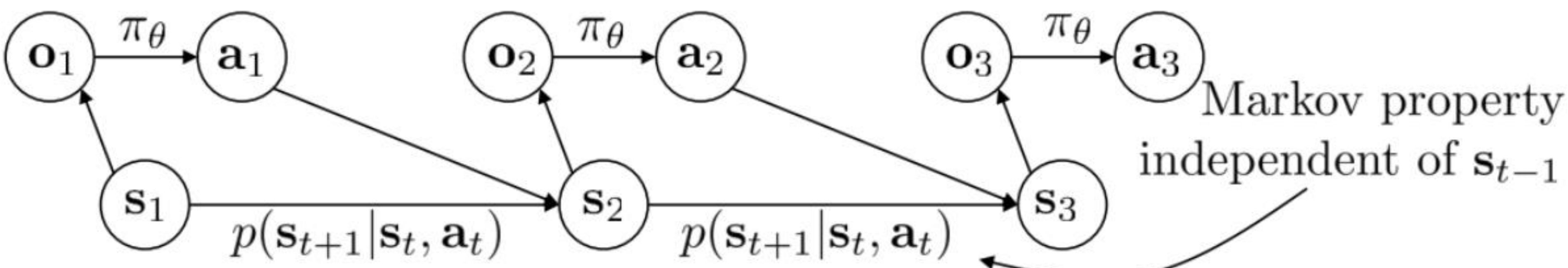
Reinforcement Learning

Function: $O \rightarrow A$

$\mathbf{o}$ $\quad\quad$ $\pi_\theta(\mathbf{a}|\mathbf{o})$ $\quad\quad$ $\mathbf{a}$

$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy (fully observed)



Reward Functions

$\mathbf{o}_t$

Function: $O \to A$

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

$\mathbf{a}_t$

which action is better or worse?

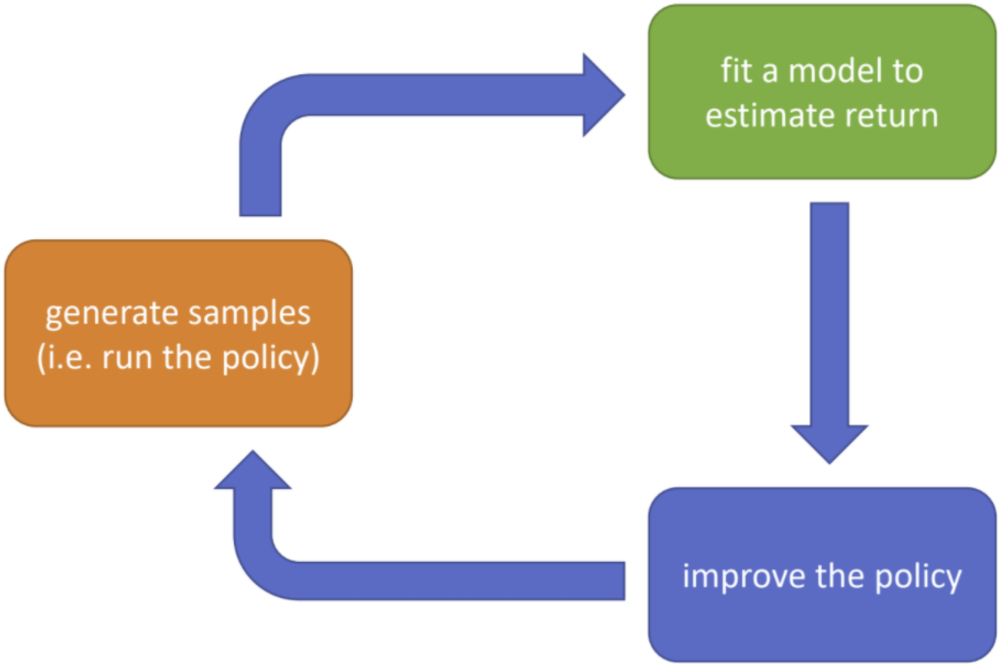
$r(\mathbf{s}, \mathbf{a})$: reward function

tells us which states and actions are better

$\mathbf{s}$, $\mathbf{a}$, $r(\mathbf{s}, \mathbf{a})$, and $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ define Markov decision process

high reward

low reward



compute $\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ (MC policy gradient)

fit $Q_\phi(\mathbf{s}, \mathbf{a})$ (actor-critic, Q-learning)

estimate $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ (model-based)

$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$ (policy gradient)

$\pi(\mathbf{s}) = \arg\max Q_\phi(\mathbf{s}, \mathbf{a})$ (Q-learning)

optimize $\pi_\theta(\mathbf{a}|\mathbf{s})$ (model-based)

Anatomy of an RL Algorithm

Policy Differentiation

Trajectory: τ
Trajectory length: T
Policy: π
Reward: r
Parameters: θ
Gradient: ∇
J: Expected reward

$$\theta^\star = \underbrace{\text{Best } \theta \text{ based on expected reward over the } T\text{-length trajectory}}_{J(\theta)}$$

$$\underbrace{\pi_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = \text{Probability of given trajectory using policy } \pi_\theta$$

$J(\theta) =$ Expected reward value given trajectory (τ) sampled from π_θ

Gradient of policy w.r.t. θ is equal to policy * gradient of log policy

$\nabla_\theta J(\theta) =$ Take gradient of $J(\theta)$ $=$ Use convenient identity to get rid of non-log policy $=$ Sample from policy instead of enumerating all possible trajectories

Expand policy(trajectory) and take log gradient

Rewrite substituting in worked-out gradient

Policy Differentiation

Trajectory: τ
Trajectory length: T
Policy: π
Reward: r
Parameters: θ
Gradient: ∇
J: Expected reward

$$\theta^\star = \arg\max_\theta \underbrace{\sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)]}_{J(\theta)}$$

$$\underbrace{\pi_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T)}_{\pi_\theta(\tau)} = \underline{p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)] = \int \pi_\theta(\tau)r(\tau)d\tau$$

$$\underline{\pi_\theta(\tau)\nabla_\theta \log\pi_\theta(\tau)} = \pi_\theta(\tau)\frac{\nabla_\theta\pi_\theta(\tau)}{\pi_\theta(\tau)} = \underline{\nabla_\theta\pi_\theta(\tau)}$$

$$\nabla_\theta J(\theta) = \int \underline{\nabla_\theta\pi_\theta(\tau)}r(\tau)d\tau = \int \underline{\pi_\theta(\tau)\nabla_\theta\log\pi_\theta(\tau)}r(\tau)d\tau = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta\log\underline{\pi_\theta(\tau)}r(\tau)]$$

$$\boxed{\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^{T}r(\mathbf{s}_t,\mathbf{a}_t)\right)\right]}$$

$$\nabla_\theta\overbrace{\left[\log \cancel{p(\mathbf{s}_1)} + \sum_{t=1}^{T}\log\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \cancel{\log p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}\right]}$$

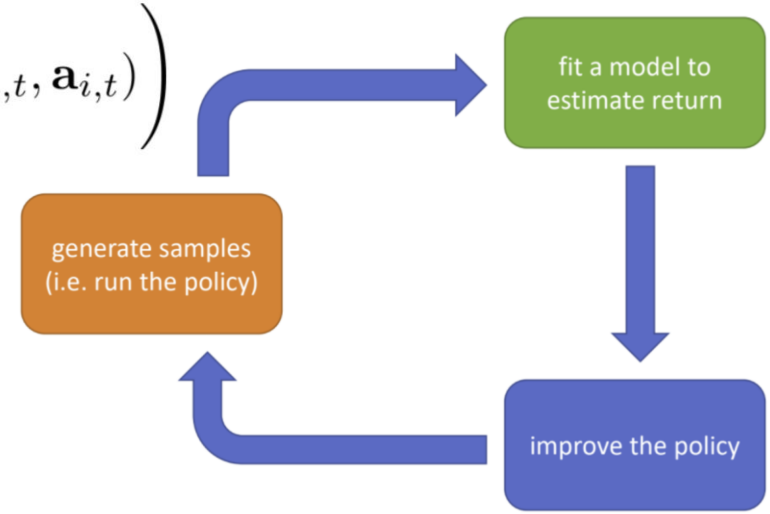
Using the Policy Gradient

$$\nabla_\theta J(\theta) = \boxed{E_{\tau \sim \pi_\theta(\tau)}} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\boxed{\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)} \right) \right]$$

$$\boxed{\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)}$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau) r(\tau)$$



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

Where does reward come from?

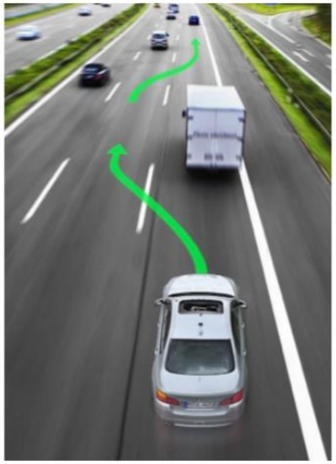
Computer Games

Mnih et al. '15

robotics

dialogue

autonomous driving



What's the reward here?
Usually we use a proxy.

It's generally easier to provide expert data than to specify a useful reward function.
Consider two cases:

- Autonomous helicopter tricks?
- Growing a plant?

Inverse Optimal Control / Inverse RL

Given:

- State and action space
- Roll-outs from π^*
- Dynamics Model (sometimes)

Goal:

- Recover reward function
- Use reward function to derive $\pi \approx \pi^*$

Challenges:

- Problem is underspecified
- Hard to evaluate your learned reward
- Demonstrations might be good, but not optimal

Maximum Entropy IRL
(Ziebart et al. 2008)

Notation:

$\tau = \{s_1, a_1, ..., s_t, a_t, ..., s_T\}$

trajectory

$R_\psi(\tau) = \sum_t r_\psi(s_t, a_t)$

learned reward

$\mathcal{D} : \{\tau_i\} \sim \pi^*$

expert demonstrations

$$p(\tau) = \frac{1}{\boxed{Z}} \exp(R_\psi(\tau))$$

Probability is exponential in the reward (good trajectories are more likely)

$$\max_\psi \sum_{\tau \in D} \log p_{r_\psi}(\tau)$$

Maximize likelihood of a given reward function parameterization

$$\boxed{Z = \int \exp\left(R_\psi(\tau)\right) d\tau}$$

Integral over all possible trajectories ☹

MaxEnt IRL

1. Initialize ψ (reward function params), gather demonstrations D
2. Solve for optimal policy $\pi(a|s)$ with reward r_ψ
3. Solve for state visitation frequencies $p(s|\psi)$
4. Compute gradient: $\nabla_\psi L = -\frac{1}{|D|}\sum_{\tau_d \in D} \frac{dr_\psi}{d\psi}(\tau_d) - \sum_s p(s|\psi)\frac{dr_\psi}{d\psi}(s)$
5. Update ψ with one gradient step using $\nabla_\psi L$
6. GOTO 2

Must solve the whole MDP in the inner loop of finding the reward function!

Making IRL work for complex problems

Must handle:

(1) Unknown dynamics
(2) Solving MDP in an inner loop

$$p(\tau) = \frac{1}{Z}\exp(R_\psi(\tau))$$

$$\max_{\psi} \sum_{\tau \in D} \log p_{r_\psi}(\tau)$$

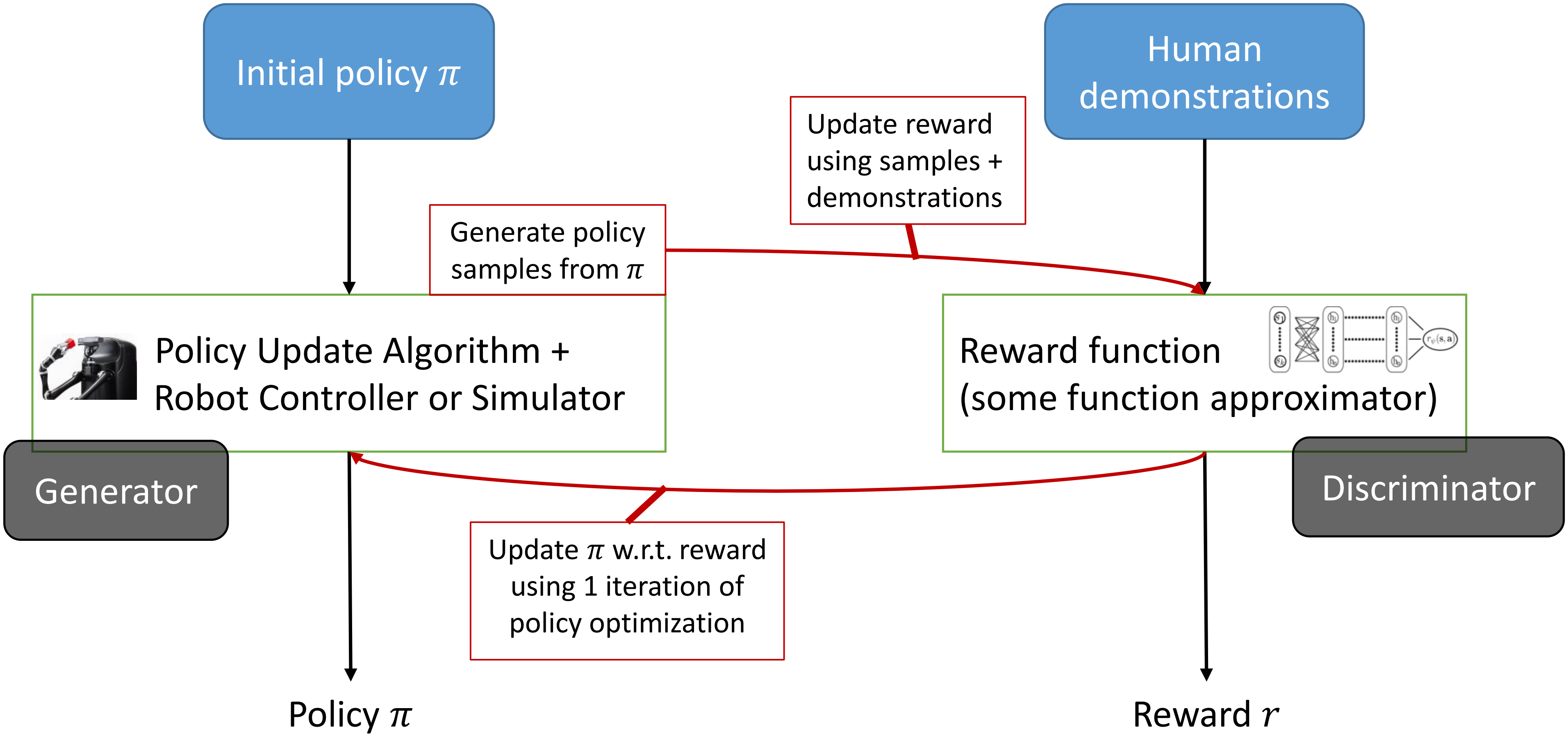
$$\boxed{Z = \int \exp\left(R_\psi(\tau)\right) d\tau}$$

Adaptively sample increasingly close to ψ by constructing a policy to do it for us

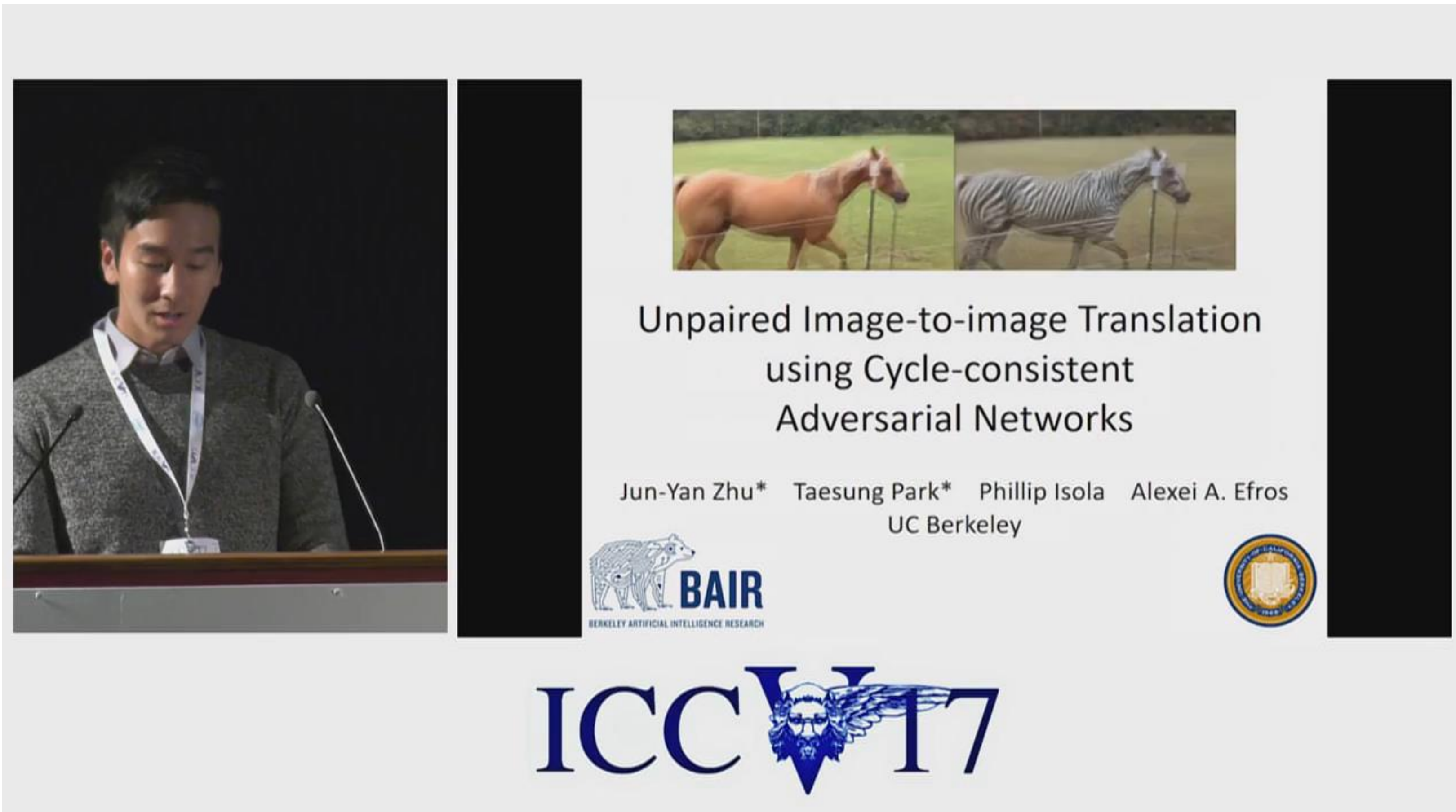
Sample from what?
Can't sample from policies near ψ because we're solving for ψ!

Can sample to approximate Z!

Guided Cost Learning
(Generative Adversarial Imitation)



Brief Aside: Generative Adversarial Networks

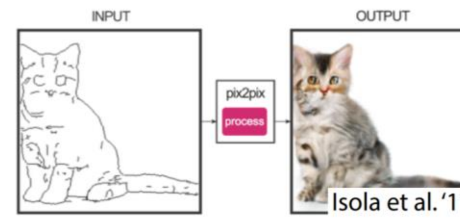


Inverse RL <=> GANs

Similar to inverse RL, **GANs** learn an objective for generative modeling.


Zhu et al.'17


Arjovsky et al.'17



Isola et al.'17

	trajectory τ	$\longleftrightarrow$	**sample x**	
Inverse RL	**policy** $\pi \sim q(\tau)$	$\longleftrightarrow$	**generator** G	GANs
	reward r	$\longrightarrow$	**discriminator** D	

Discriminators and Loss Functions

Inverse RL			GANs
	trajectory τ	⟷ **sample x**	
	policy π~q(τ)	⟷ **generator** G	
	reward r	⟶ **discriminator** D	

data distribution p

Guided Cost Learning:

$$D^*(\tau) = \frac{p(\tau)}{p(\tau) + q(\tau)}$$

$$D_\psi(\tau) = \frac{\frac{1}{Z}\exp(R_\psi)}{\frac{1}{Z}\exp\left(R_\psi\right) + q(\tau)}$$

Guided Adversarial Imitation Learning:

$D_\psi(\tau) =$ some classifier

$$\mathcal{L}_{\mathrm{discriminator}}(\psi) = \mathbb{E}_{\tau\sim p}[-\log D_\psi(\tau)] + \mathbb{E}_{\tau\sim q}[-\log(1 - D_\psi(\tau))]$$

Generators and Loss Functions

Inverse RL				GANs
	trajectory τ	⟷	**sample x**	
	policy π~q(τ)	⟷	**generator** G	
	reward r	⟶	**discriminator** D	

data distribution p

$$\mathcal{L}_{\text{generator}}(\theta) = \mathbb{E}_{\tau\sim q}[\log(1 - D_\psi(\tau)) - \log D_\psi(\tau)]$$
$$= \mathbb{E}_{\tau\sim q}[\log q(\tau) + \log Z - R_\psi(\tau)]$$

Entropy-regularized RL

Must train generator/policy with RL because we don't know the dynamics of the environment!

(otherwise we could just train using the discriminator signal straight through to the policy)

IRL Recap

Goal: Infer reward function underlying expert demonstrations

Evaluating the partition function (Z):

- Initial approaches solve the MDP in the inner loop of IRL (or assume known dynamics).
- Can estimate Z using sampling!

Connection to Generative Adversarial Networks:

- Sampling-based MaxEnt IRL is a GAN with a special form of discriminator, using RL to optimize the generator.

Further Reading

Classic Papers

- **Abbeel & Ng ICML '04.** *Apprenticeship Learning via Inverse Reinforcement Learning.* Good introduction to inverse reinforcement learning
- **Ziebart et al. AAAI '08**. *Maximum Entropy Inverse Reinforcement Learning.* Introduction of probabilistic method for inverse reinforcement learning

Modern Papers

- **Wulfmeier et al. arXiv '16**. *Deep Maximum Entropy Inverse Reinforcement Learning.* MaxEnt IRL using deep reward functions
- **Finn et al. ICML '16**. *Guided Cost Learning*. Sampling-based method for MaxEnt IRL that handles unknown dynamics and deep reward functions
- **Ho & Ermon NIPS '16**. *Generative Adversarial Imitation Learning*. IRL method building on Abbeel & Ng '04 using generative adversarial networks

Designing Your Evaluation

Evaluation Design:

What are your hypotheses about your system?

How will you test them?

What are you trying to prove with this work?

Experiment Design:

Do you need human subjects?

Are your conditions likely to test your hypotheses?

Within-subjects or between-subjects?

Protocol design:

Someone not on your project should be able to run your experiment with this script!