UNIVERSITÀ
DEGLI STUDI DI TRIESTE

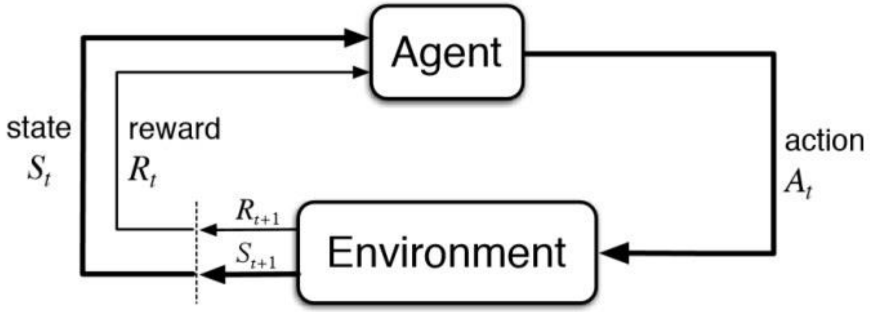
Learning reward function with Guided Cost Learning technique

Imola Fodor SM3500474

Final project – Reinforcement Learning course

DSSC, 2022-2023

The Reinforcement Learning system



- Markov Decision Process is a tuple <S, A, p, R, gamma> , policy π
- Goal/return – maximize discounted cumulative reward

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+(k+1)}$$

- Value function – gives long term value of state s

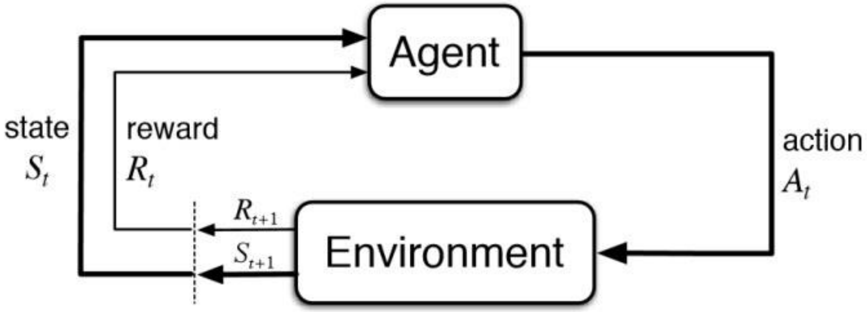
$$\mathrm{V}(s) = E[G_t | S_t = s]$$

- Overall Objective, Expected Reward: $argmax_{\pi} [E(G_t)]$
- Recursion relationship

$$\mathrm{V}_{\pi} (s) = E[R_{t+1} + \gamma V_{\pi}(S_{t+1}) | S_t = s]$$

- Reward function has a crucial role to quantify how good the agent is.

Reward function



- Reward function has a crucial role to quantify how good the agent is.
- Can be: hand-crafted and learnt

- Simple: win a game
- Hard: drive a car

Inverse RL

Forward RL

Given:

states s from S, actions a from A

(sometimes) transitions $p(s'|s,a)$

Reward function $r(s,a)$

Learn $\pi*\ (a|s)$

Inverse RL

Given:

expert state, action pairs

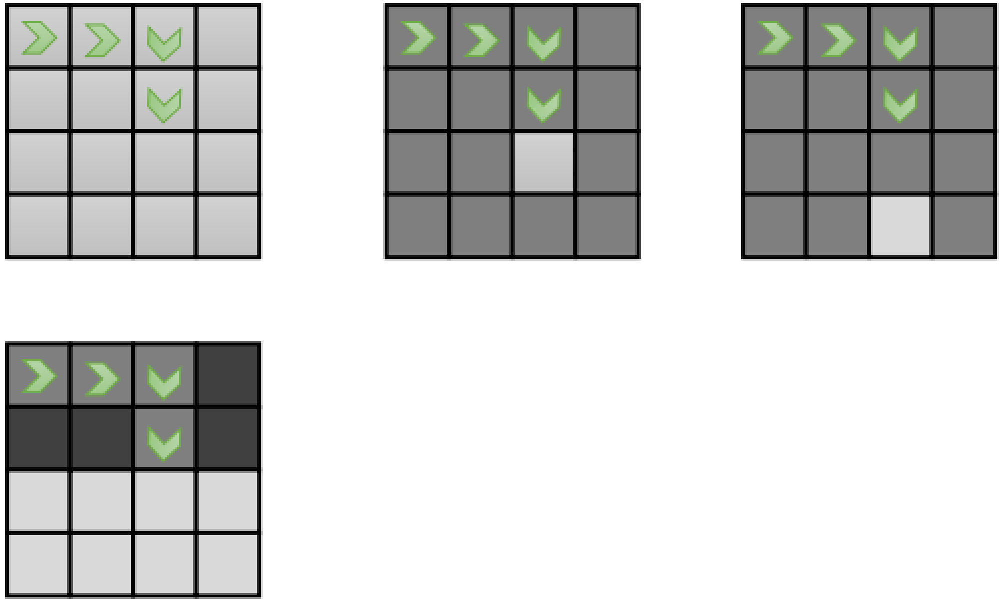
(sometimes) transitions $p(s'|s,a)$

Learn: $r_\phi(s,a)$

..and then learn $\pi*\ (a|s)$

Ambiguity

- IRL is essentially an ill-posed (unspecified) problem because multiple reward functions can explain the same expert's behavior



CS 285 at UC Berkeley
Deep Reinforcement Learning Course

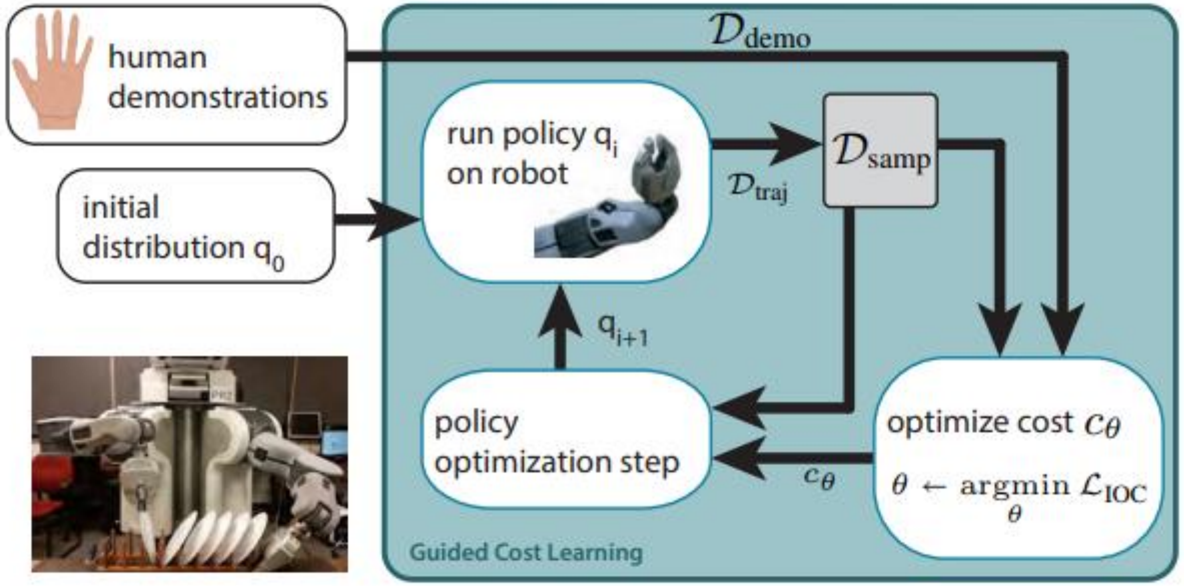
Learning a reward function -IRL

- No explicit reward function

Method	Pro	Con
Supervised Learning (Behavior Cloning)	Easy	Input far from training data distribution Reward function hidden
Linear reward function	Interpretable	Ambiguous Performs as well as the expert
Maximum margin	Less ambiguity	Requires perfect expert trajectories in entire state-space
Max (Relative) Entropy based	Probabilistic model, no ambiguity Good basis for other methods	Works for known dynamics, discrete state spaces
…		

Guided Cost Learning – High level

- Learn optimal policy and reward from demonstrations
- Is based on MaxEntropy, *Ziebart et al., 2008* algorithm, but addresses summing/integrating over all trajectories



Guided Cost Learning, Finn et al, 2016

MaxEntropy – for known dynamics

- Data distribution (Energy based model):

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

where $\tau\ =\ \{x_1, a_1, \ldots, x_T\ , a_T\ \}$ is a trajectory sample, $R(\tau\ ) =\ \sum_t r_\phi(s_t, a_t)$ is an unknown reward function parameterized by ϕ, and x_t and a_t are the state and action at time step t. Z partition function

- For inferring the reward function:

$$max_\phi\ L(\phi) = \sum_{\tau\in D} logp_{r\psi}(\tau) = \sum_{\tau\in D} r_\phi(\tau) - MlogZ\ \ = \sum_{\tau\in D} r_\phi(\tau) - Mlog\sum_{\tau}\exp\left(r_\phi(\tau)\right)$$

- To optimize, taking it’s derivative

MaxEntropy - for known dynamics

- Objective: $\sum_{\tau \in D} r_\phi(\tau) - M log \sum_\tau \exp\left(r_\phi(\tau)\right)$

- To optimize, taking it's derivative:

$$\nabla_\phi L = \frac{1}{N} \sum_{i=1}^{N} \nabla_\phi r_\phi(\tau_i) - \frac{1}{Z} \sum_\tau \exp\left(r_\phi(\tau)\right) \nabla_\phi r_\phi(\tau)$$

$$\nabla_\phi L = E_{\tau \sim \pi^*(\tau)}\left[\nabla_\phi r_\phi(\tau_i)\right] \boxed{- E_{\tau \sim p\,(\tau)}[\nabla_\phi r_\phi(\tau)]}$$

Estimate with expert samples

Under current policy

MaxEntropy - algorithm

0. Initialize ϕ, gather demonstrations D

1. Solve for optimal policy $\pi(\mathbf{a}|\mathbf{s})$ w.r.t. reward r_ϕ
2. Solve for state visitation frequencies $p(\boldsymbol{s}|\phi)$ - sample based updates
3. Compute gradient $\nabla_\phi L = \frac{1}{N}\sum_{i=1}^{N} \nabla_\phi r_\phi(\tau_i) - \sum_\tau p(s|\phi)\, \nabla_\phi r_\phi(s)$
4. Update ϕ with one gradient step using $\nabla_\phi L$

Guided Cost Learning GCL – Policy

- Not learning the optimal policy for given reward
- Not calculating Z
- Importance sampling ideally
- Adaptive sampling (from a created policy)

$$\frac{1}{\mathrm{N}}\sum_{\tau\in D} R_\phi(\tau) + \frac{1}{M} log \sum_{\tau_j} \frac{\exp\left(-R_\phi(\tau_j)\right)}{\pi(\tau_j)}$$

$$\nabla_\phi L \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\phi r_\phi(\tau_i) - \frac{1}{\sum_j w_j}\sum_{j=1}^{M} w_j \nabla_\phi r_\phi(\tau_j)$$

$$w_j = \frac{p(\tau)\exp(r_\phi(\tau_j))}{\pi(\tau_j)} = \frac{\exp(\sum_t r_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t))}{\prod_t \pi(\boldsymbol{a}_t|\boldsymbol{s}_t)}$$

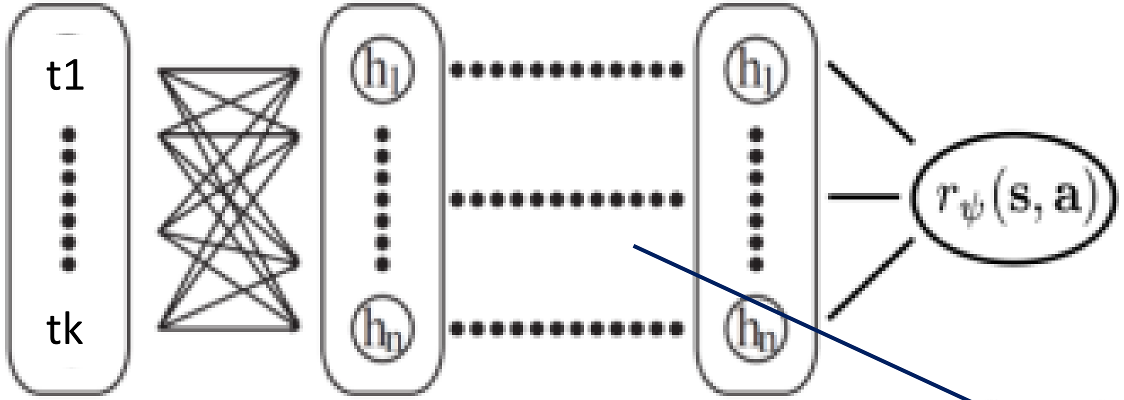
GCL – Cost

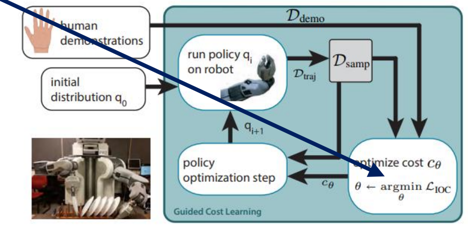
- The neural network for cost function as loss takes:

$$\frac{1}{\mathrm{N}}\sum_{\tau\in D} R_{\phi}(\tau) + \frac{1}{M} log \sum_{\tau_j} \frac{\exp\left(-R_{\phi}(\tau_j)\right)}{\pi(\tau_j)}$$

and performs the backward pass with gradient descent to improve

sample from
current policy
+
demo traj

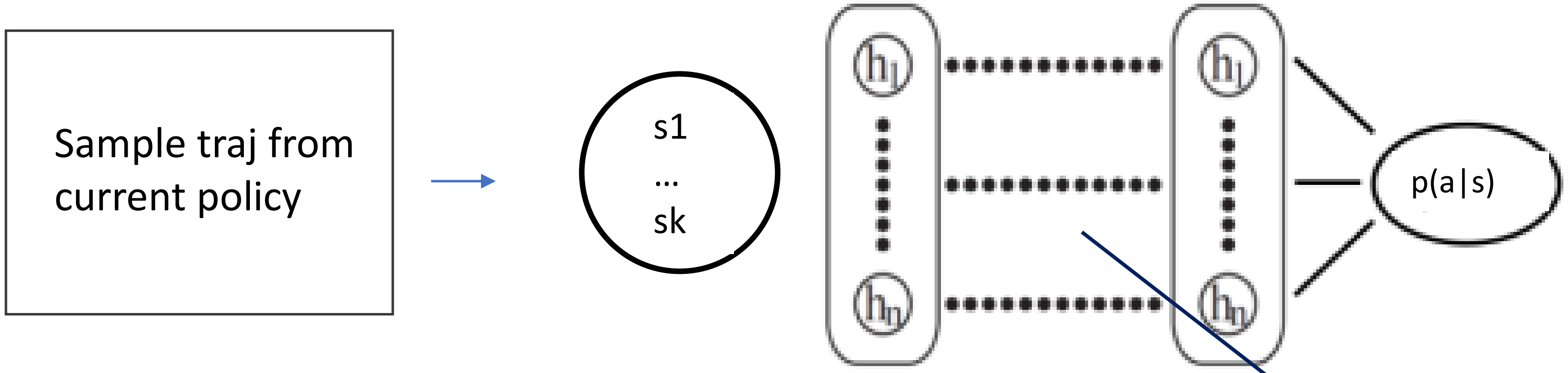




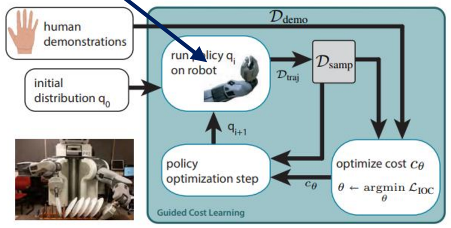
The policy gets updated w.r.t to reward

GCL – Policy w.r.t Cost

- At each improvement of the cost function, the policy performs a single Gradient Policy step (eg. REINFORCE algorithm)



REINFORCE Algorithm:
1. Run forward pass of policy network
2. Get cumulative rewards for each state, action based on current cost function
3. Calculate the expected reward $\mathrm{G} = \log(\mathrm{p}_{\pi}(\mathrm{a}|\mathrm{s}))*\gamma r$
3. Adjust weights by back-propagating the error $(-G)$



GCL – Algorithm

0. Input: demo trajectories, arbitrary (or from demos) policy q NN, arbitrary cost NN

1. Generate sample trajectory for current q
2. Forward pass of cost function with demo and q trajectory
3. Calculate c loss $\frac{1}{\mathrm{N}}\sum_{\tau\in D} R_{\phi}(\tau) + \frac{1}{M} log \sum_{\tau_j} \frac{\exp\left(-R_{\phi}(\tau_j)\right)}{q(\tau_j)}$ and backpropagate error
4. For q trajectory run current q, calculate cost f
5. Calculate q loss $\log(p_{\pi}(a|s)) * \gamma r$ and backpropagate error

GCL Implementations

- Implementation (**I1**) found online for *CartPole-v0* environment
  - Bugs fixed
- I1 adapted to *LunarLander-v2* environment, which has multiple actions on output, ie. four
  - collected expert trajectories from (yet another) implementation (**I2**) of LunarLander-v2, which trained successfully an agent for the problem

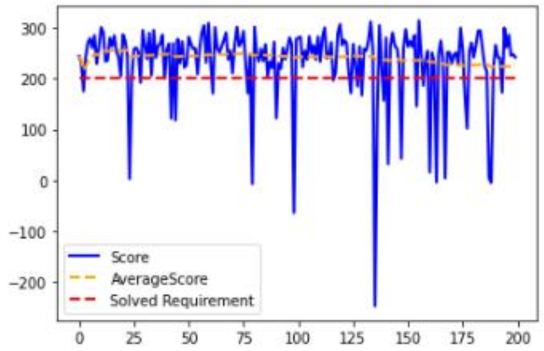
gym/gymnasium : Lunar Lander v2

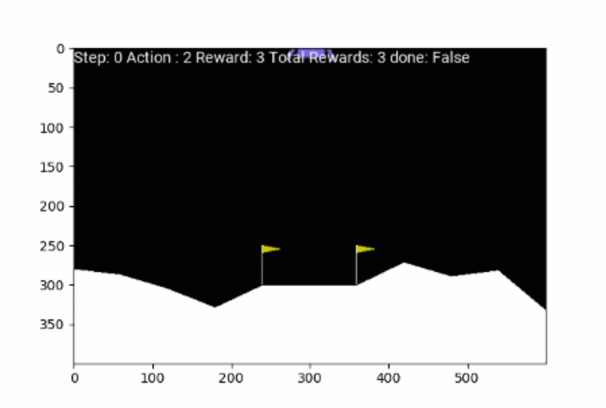
- v2 takes into consideration also the fuel consumption

Actions	States	Rewards Rule	Episode termination
0 - No action 1 - Fire left engine 2 - Fire main engine 3 - Fire right engine	0 - Lander horizontal coordinate 1 - Lander vertical coordinate 2 - Lander horizontal speed 3 - Lander vertical speed 4 - Lander angle 5 - Lander angular speed 6 - Bool: 1 if first leg has contact, else 0 7 - Bool: 1 if second leg has contact, else 0	Moving from the top of the screen to the landing pad gives a scalar reward between (100-140) Negative reward if the lander moves away from the landing pad If the lander crashes, a scalar reward of (-100) is given If the lander comes to rest, a scalar reward of (100) is given Each leg with ground contact corresponds to a scalar reward of (10) Firing the main engine corresponds to a scalar reward of (-0.3) per frame Firing the side engines corresponds to a scalar reward of (-0.3) per frame	Lander crashes Lander comes to rest Episode length > 400

GCL Results for LunarLander-v2

- I2: Perform usual RL, train a policy (DQN eg.), and test to collect expert trajectories (demos): deleted bad episodes ~30/200



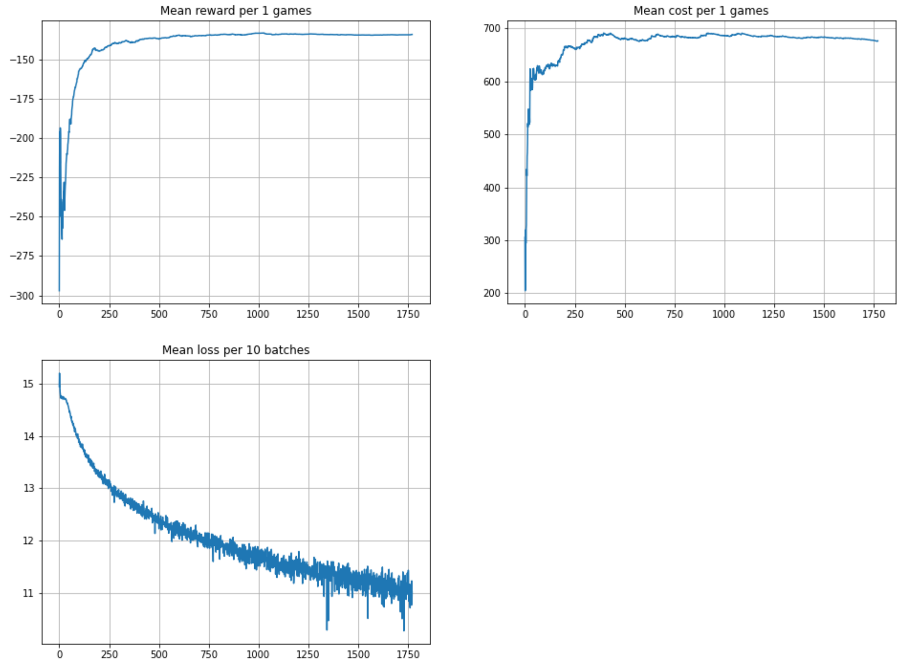


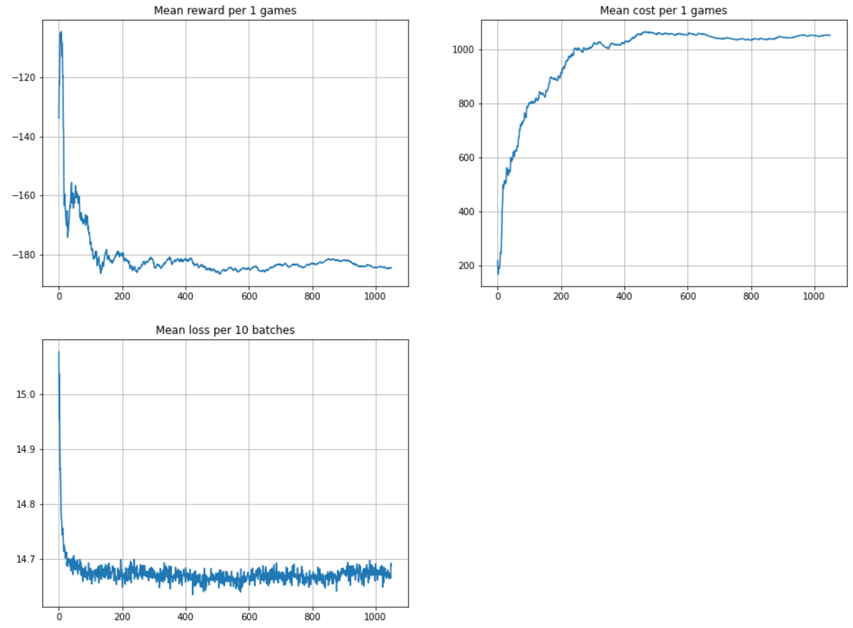
NOTE: this particular policy trained with reward rules & REINFORCE algo (not I2)

- I1: **Learn reward** and policy: converge to poor policy
- Next steps:
  - Better baseline
  - Actor-Critic
  - Use the policy from I2 as starting policy to improve in GCL
  - Not use env from gym, but generate steps (for trajectories) from learnt transition probabilities

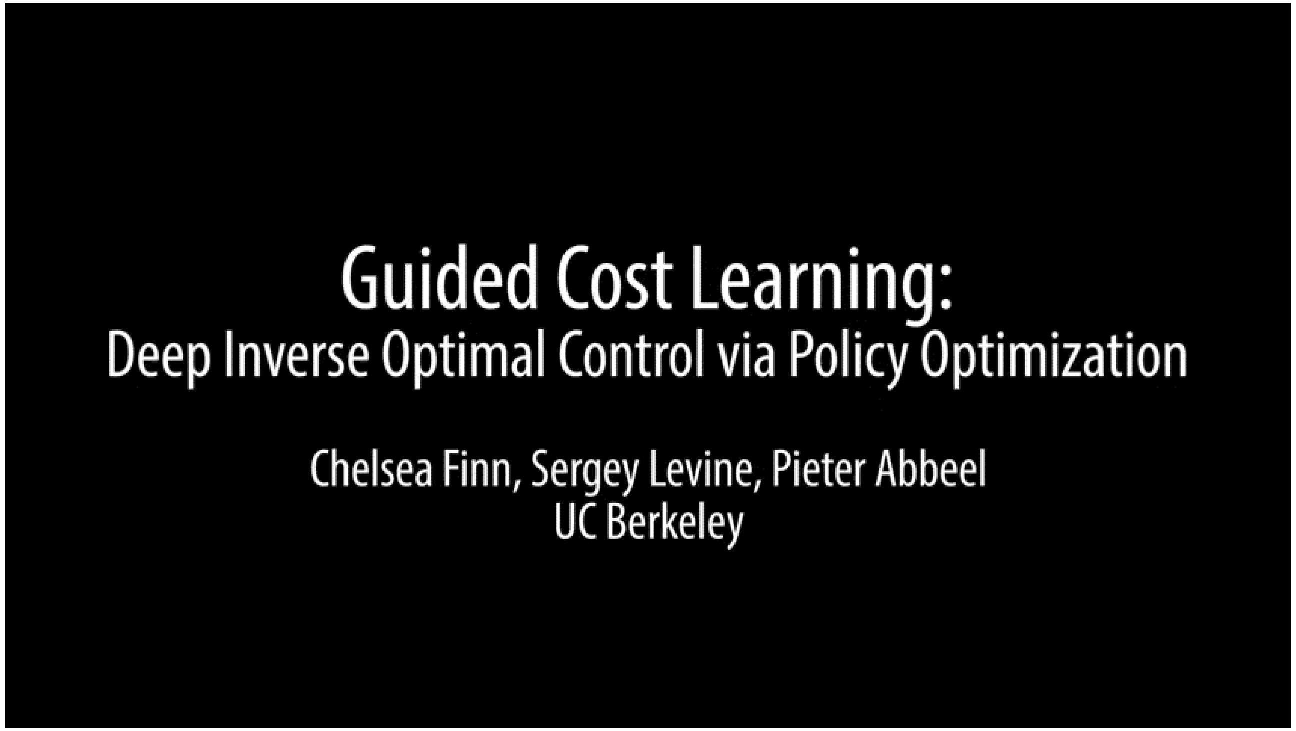
GCL Results for LunarLander-v2

- I1: **Learn reward** and policy
- Without weight decay it was diverging
- A. without baseline added, B. with whitening baseline





GCL Success Stories



Dish placement and pouring tasks. The robot learned
to place the plate gently into the correct slot, and to pour almonds, localizing the target cup using unsupervised visual features.

Bibliography

- Inverse RL, Andrew Ng et al, 2000
- MaxEntropy Ziebart et al., 2008
- Guided Cost Learning, Finn et al, 2016
- Generative Adversarial Imitation Learning, Jonathan Ho, Stefano Ermon, 2016

- A Survey of Inverse Reinforcement Learning: Challenges, Methods and Progress, Saurabh Arora et al, 2018