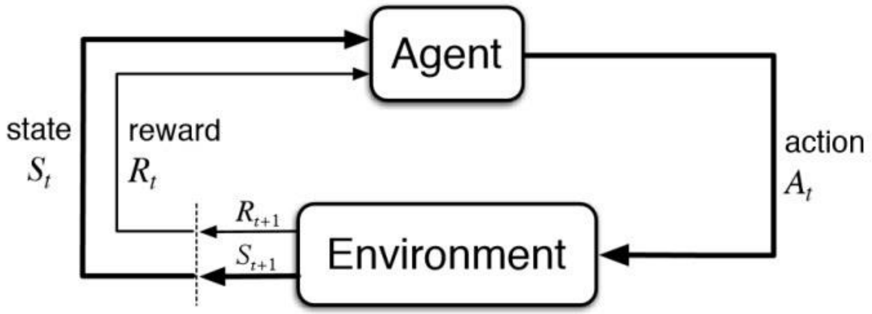
Inverse RL/Optimal Control w/ Energy based model (EBM)

Imola Fodor

- Reinforcement Learning (RL) high-level and challenges
- Inverse Reinforcement Learning (IRL) idea
- IRL: Maximum Entropy → Guided Cost Learning
- Electrolux context

The Reinforcement Learning system



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

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

- Value function – gives long term value of state s

$$V(s) = E[G_t | S_t = s]$$

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

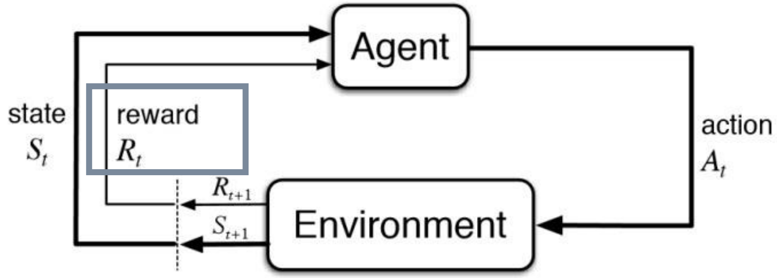
$$V_{\pi}(s) = E[R_{t+1} + \gamma V_{\pi}(S_{t+1}) | S_t = s]$$

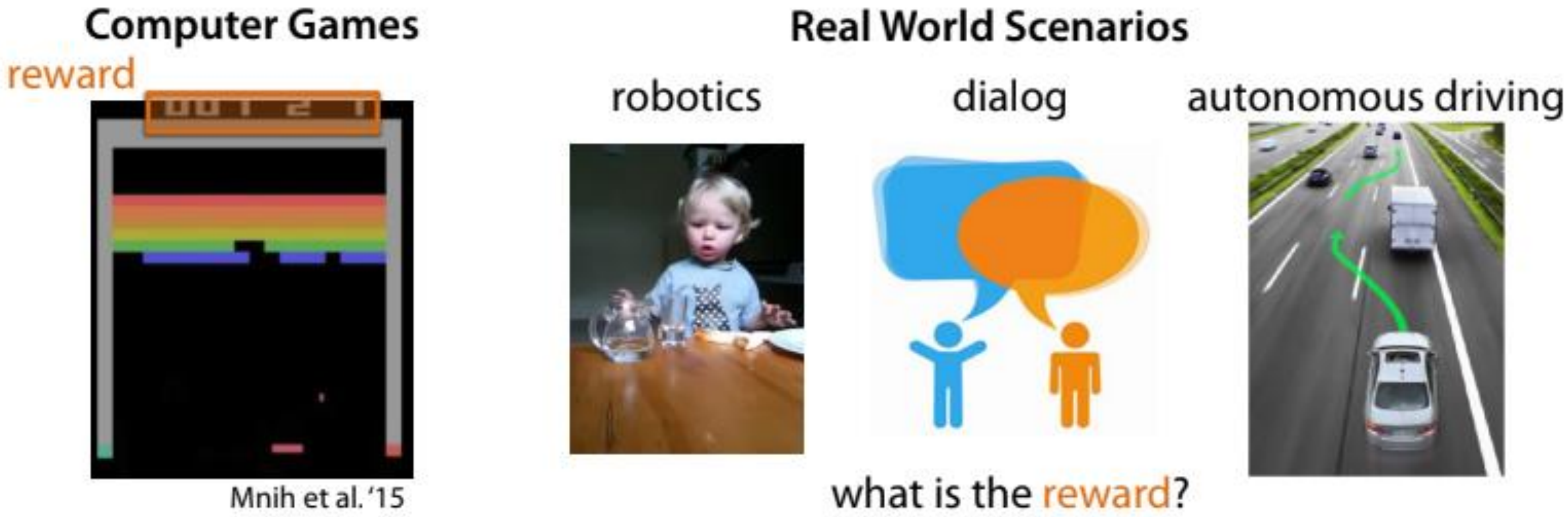
- Reward function has a crucial role to quantify how good the agent is.

Classified as Internal

Reward function

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





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 $p(s'|s, a)$

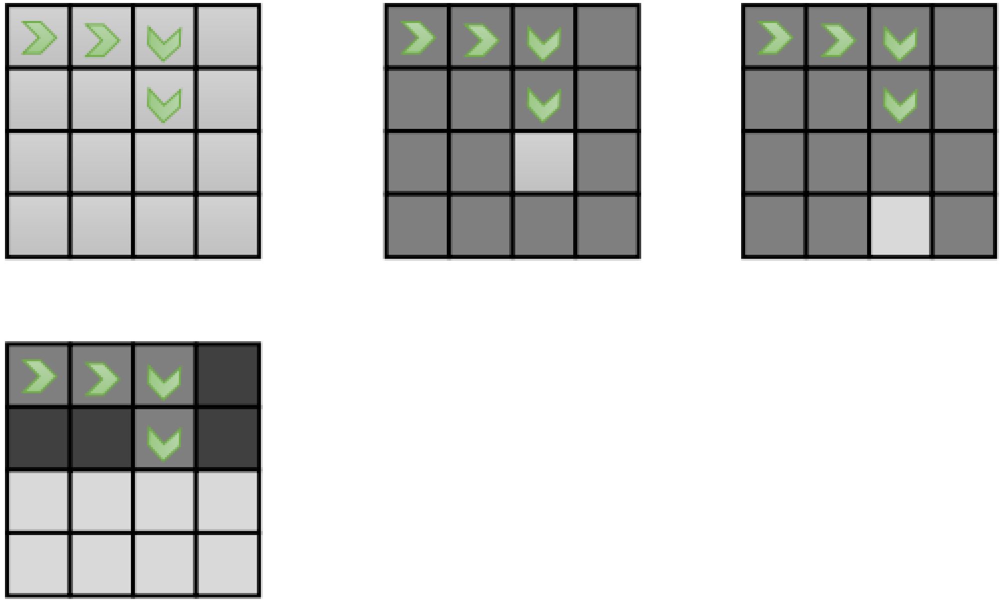
learn: $r_\phi(s, a)$

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

Classified as Internal

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

Classified as Internal

Energy Based Models

- Originally models the physical behaviour of interacting particles - physics
- Class of generative machine learning models
- Governed by an energy function that describes the probability of a certain state
- Guaranteed to sample a valid probability distribution (learn to sample)

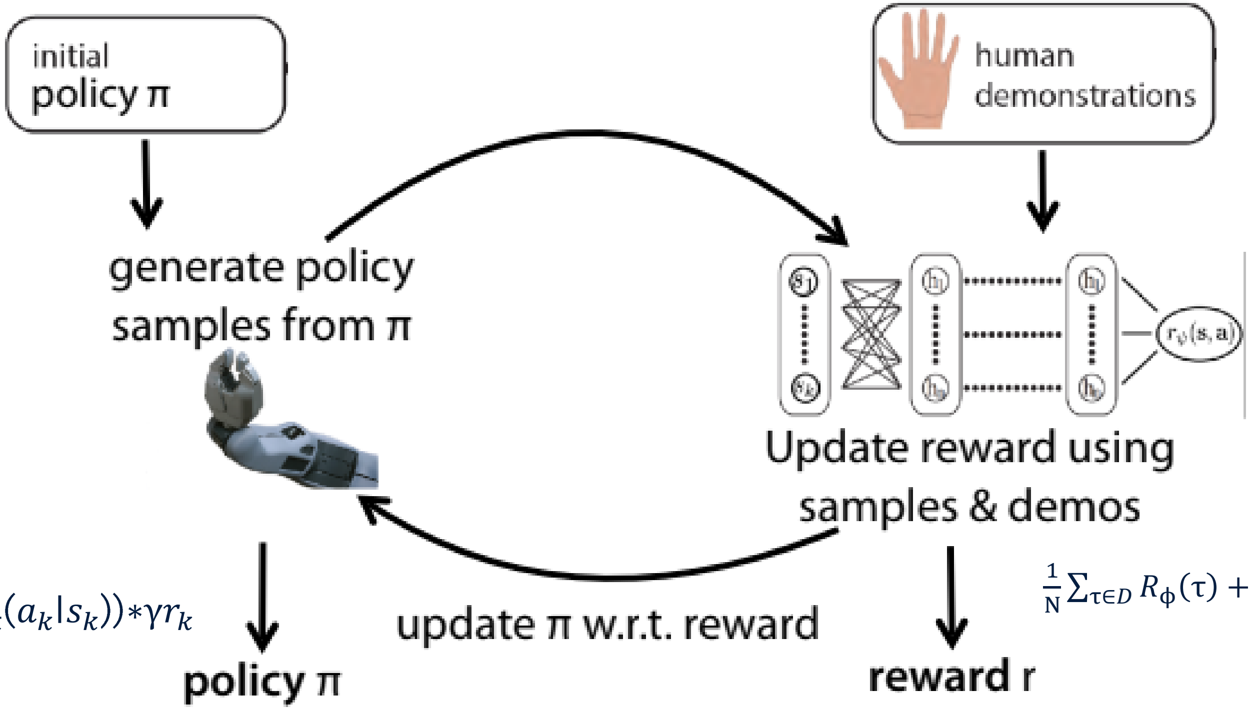
- What probability distribution $\mathrm{p}(x)$ can be assigned to a state x with energy $E(x)$?

- Boltzmann distribution
  - The distribution shows that states with lower energy will always have a higher probability of being occupied.
- Through maximum entropy..

Classified as Internal

guided cost learning algorithm

(Finn et al. ICML '16)



$-\sum \log(p_\pi(a_k|s_k)) * \gamma r_k$

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

Chelsea Finn
Deep RL Bootcamp

Maximum entropy

- The principle of maximum entropy states that the probability distribution which best represents the current state of knowledge about a system is the one with largest entropy, where the uncertainty is maximal.
- The formulas show the connection between energy, maximum entropy and Boltzmann distribution

$$\max_{P(x)} \sum_x -P(x) \log P(x)$$

$$\text{s.t.} \sum_x P(x) E(x) = \langle E \rangle,$$

giving [22]

$$\boxed{P(x) = \frac{1}{Z} \exp\left[-E(x)/T\right],}$$

- <u>The Boltzmann distribution establishes a concrete relationship between energy and probability: low-energy states are the most likely to be observed.</u>
- In low energy states are observed, all other in high temperatures are equally likely

Classified as Internal

IRL MaxEntropy – for known dynamics

- Boltzmann data distribution (Energy based model):

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

where $\tau = \{x_1, a_1, \dots, x_T, a_T\}$ 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 we need:

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

- To optimize, taking it's derivative

Classified as Internal

Guided Cost Learning GCL – Cost

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

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

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

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

Classified as Internal

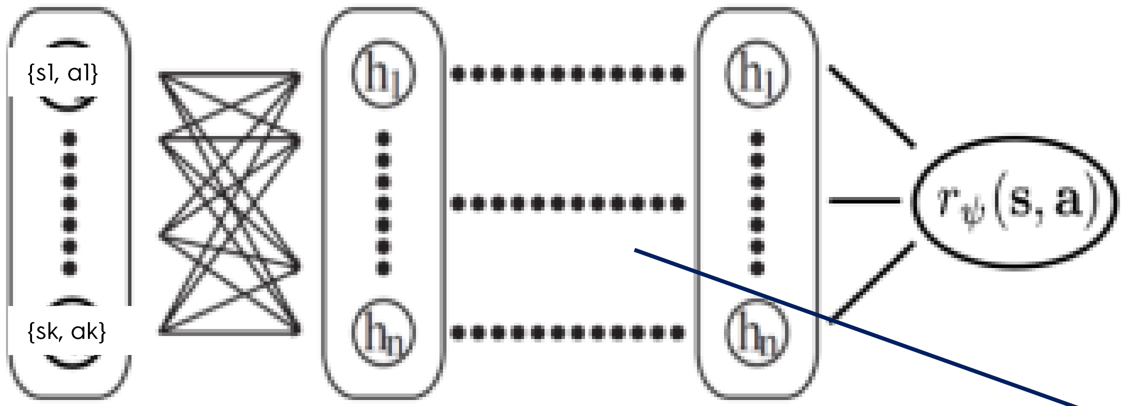
GCL – Reward

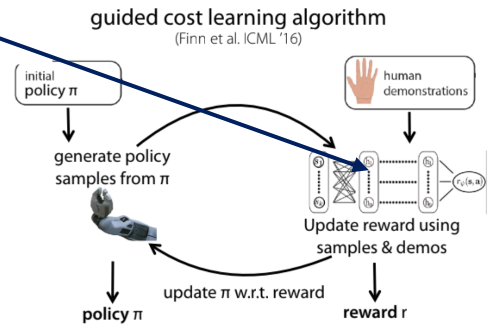
- The neural network for cost function as loss takes:

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

and performs the backward pass with gradient descent to improve

sample from
current policy
+
demo traj



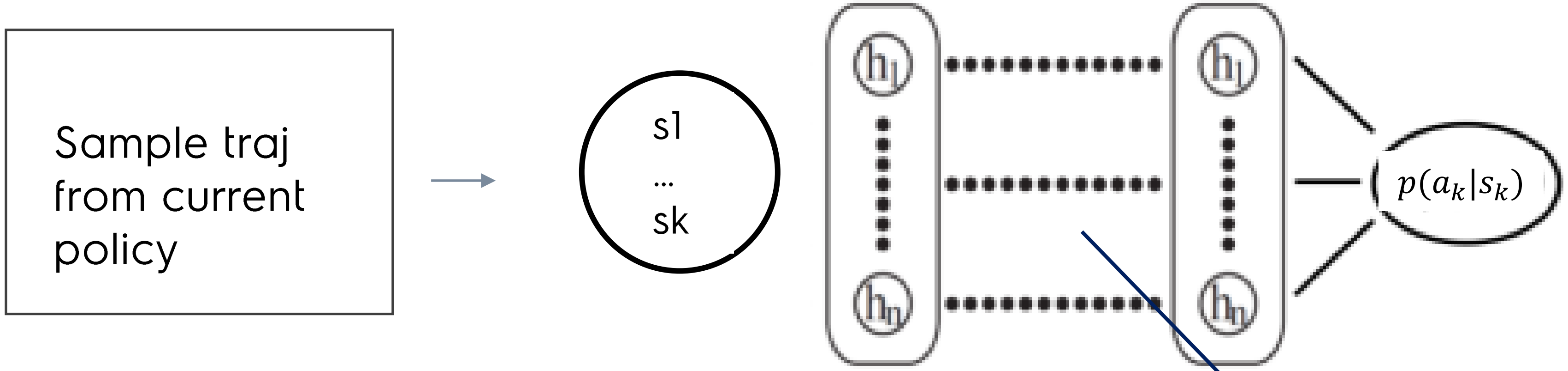


The policy gets updated w.r.t to reward

Classified as Internal

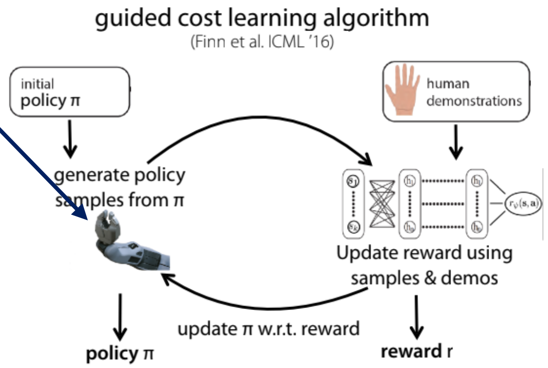
GCL – Policy

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



REINFORCE Algorithm:

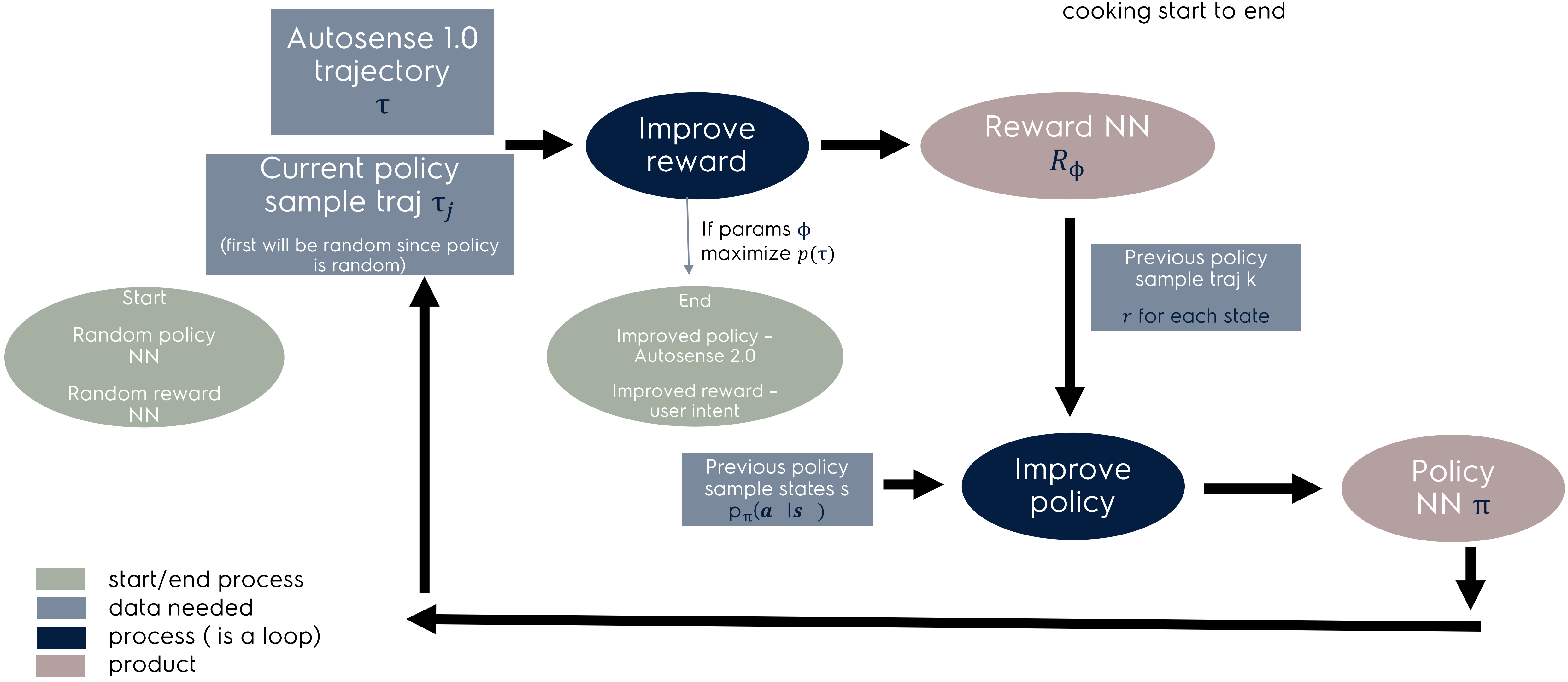
1. Run forward pass of policy network
2. Get cumulative reward $\boldsymbol{r}$ for each state, action based on current cost function
3. Calculate the expected reward $G = \log(p_\pi(\mathbf{a}|\mathbf{s})) * \gamma \boldsymbol{r}$
4. Adjust weights by back-propagating the error $(-\sum G)$



Classified as Internal

Autosense context

Policy – Control (state->action)
Reward – Cleaner air/power consumption
Trajectory – tvoc, speed pair from cooking start to end



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 $\frac{1}{N}\sum_{\tau \in D} R_\phi(\tau) - \frac{1}{M} log \sum_{\tau_j} \frac{\exp\left(R_\phi(\tau_j)\right)}{q(\tau_j)}$ 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 – (sometimes) environment

In the examples an environment is used, eg. CartPole/LunarLander-v2 with implemented physics to facilitate the simulation of real behavior of the system.

In our internal cases there would be a **third** model that is able to map state + action to new_state.

+	−
Simulation of **system model**	Run **system** itself

Classified as Internal

gym/gymnasium : Lunar Lander v2

- v2 takes into consideration also the fuel consumption

Actions	States	Rewards Rule	Episode terminatio n
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

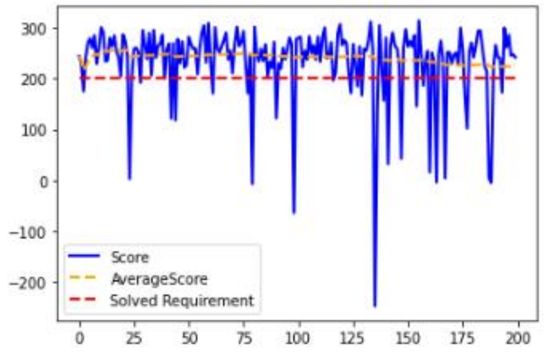
Classified as Internal

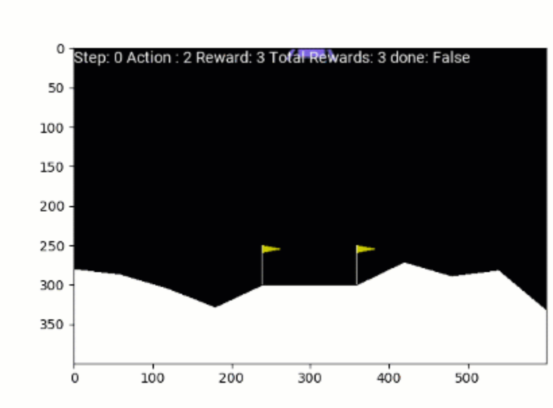
GCL Implementations

- 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

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



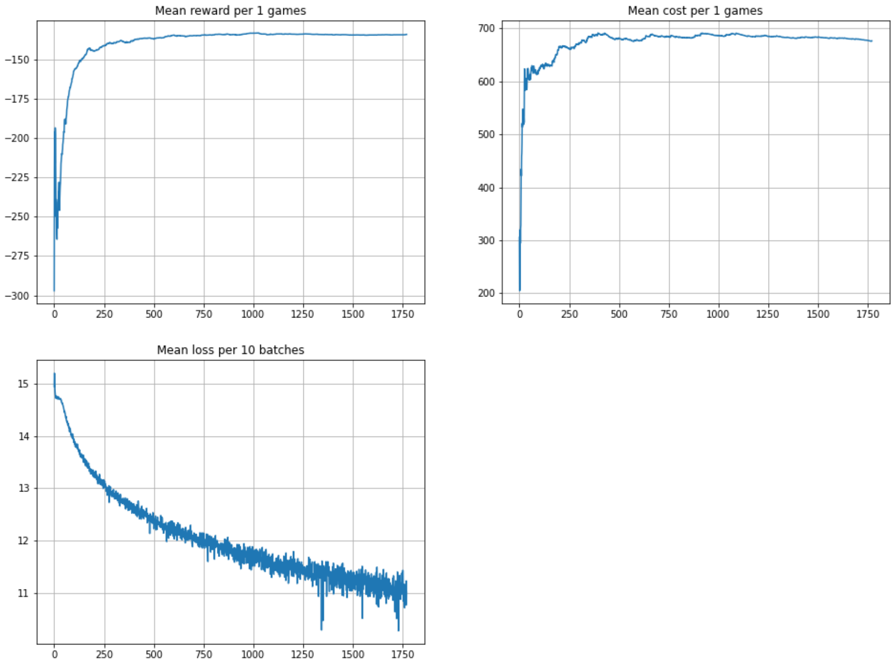


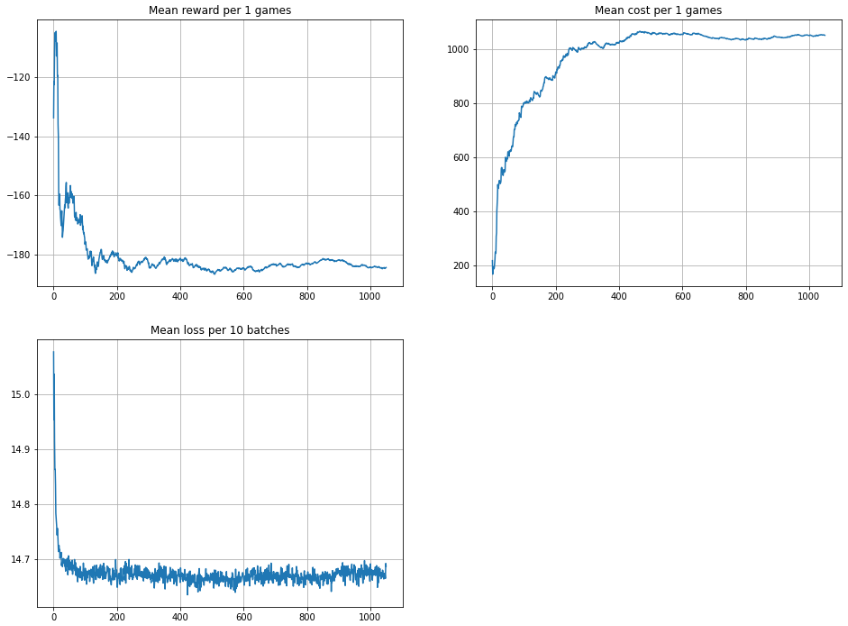
NOTE: this particular policy trained with reward rules & REINFORCE algo (not I2)

- 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

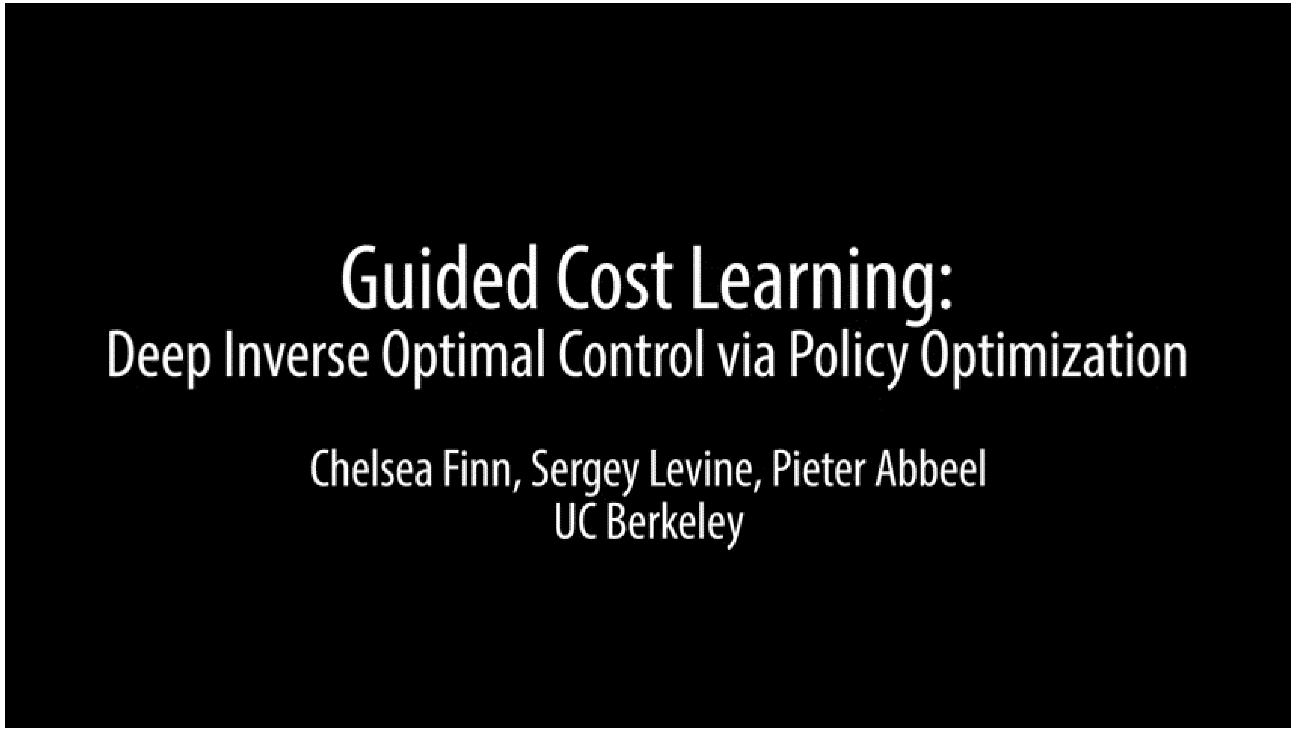
GCL Results for LunarLander-v2

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





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.

Literature

- A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models, C. Finn, P. Christiano, P. Abbeel, S. Levine

- 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

Classified as Internal

Electrolux