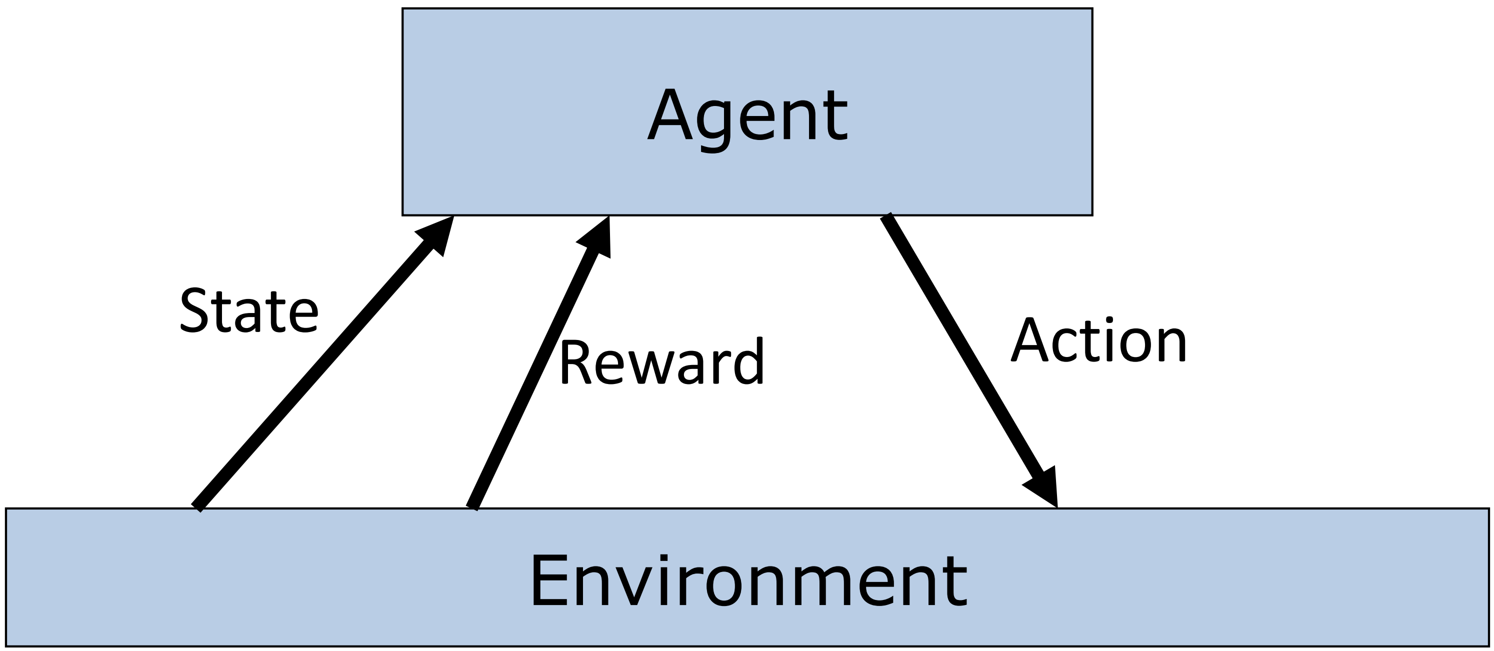
Inverse Reinforcement Learning
CS885 Reinforcement Learning
Module 6: November 9, 2021

Ziebart, B. D., Bagnell, J. A., & Dey, A. K. (2010). Modeling interaction via the principle of maximum causal entropy. In ICML.

Finn, C., Levine, S., & Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *ICML* (pp. 49-58).

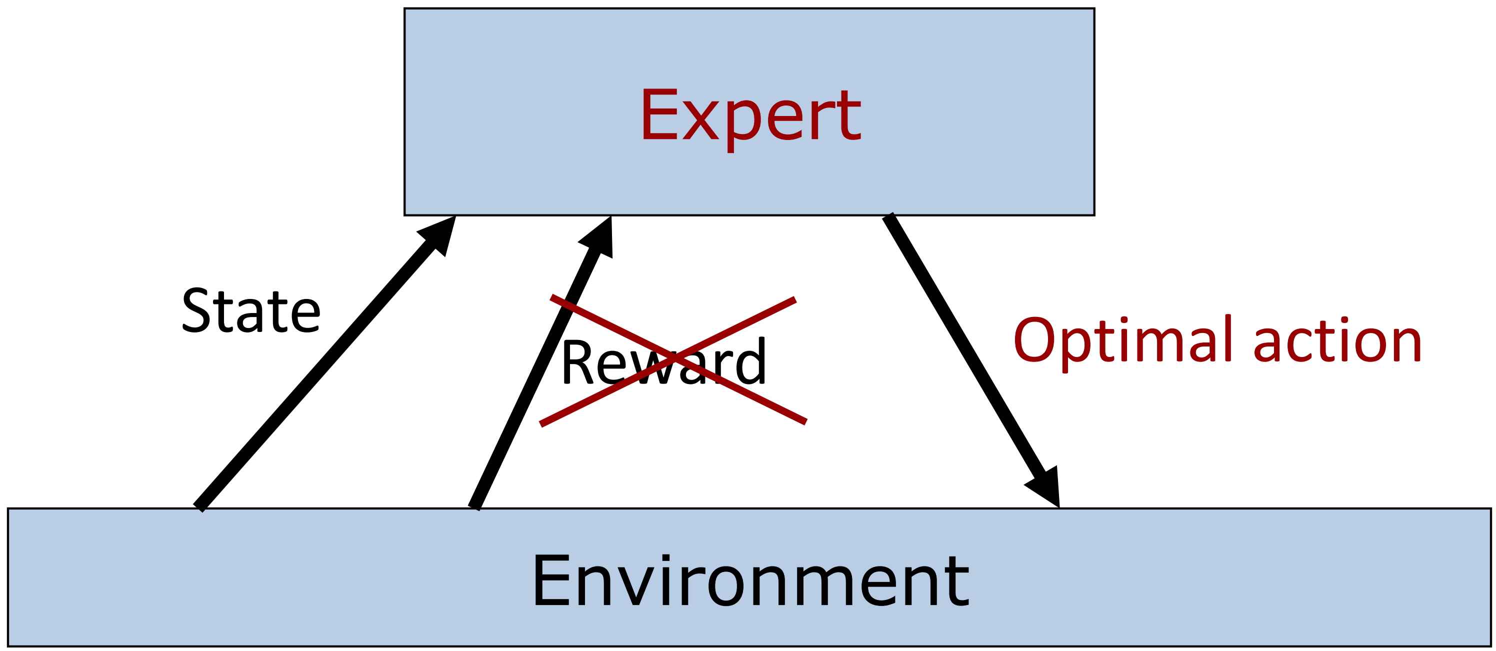
Reinforcement Learning Problem



Data: $(s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_h, a_h, r_h)$

Goal: Learn to choose actions that maximize rewards

Imitation Learning



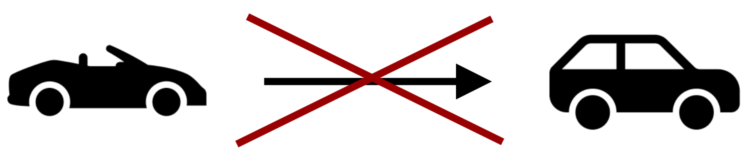
Data: $(s_0, a_0^*, s_1, a_1^*, \ldots, s_h, a_h^*)$
Goal: Learn to choose actions by imitating expert actions

Problems

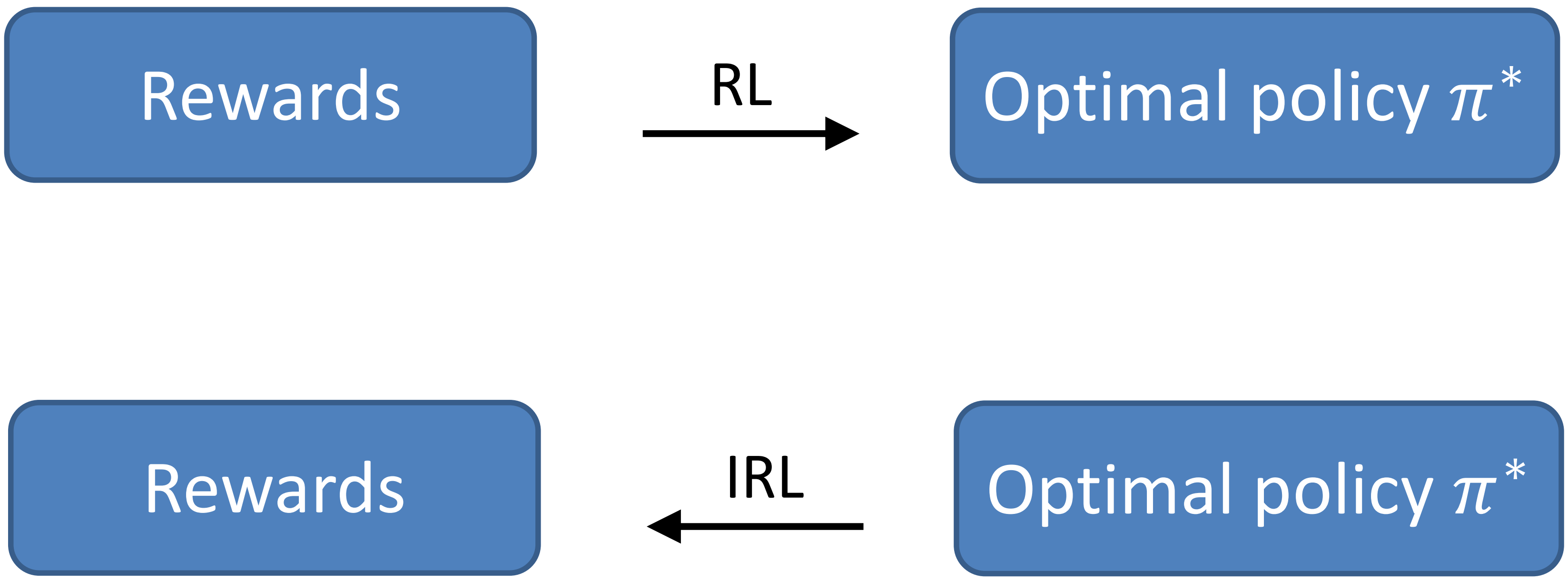
- Imitation learning: supervised learning formulation
  - **Issue #1**: Assumption that state-action pairs are identically and independently distributed (i.i.d.) is false

$$(s_0, a_0^*) \rightarrow (s_1, a_1^*) \rightarrow \cdots \rightarrow (s_h, a_h^*)$$

  - **Issue #2**: Can't easily transfer learnt policy to environments with different dynamics



Inverse Reinforcement Learning (IRL)



Benefit: can easily transfer reward function to new environment where we can learn an optimal policy

Formal Definition

Reinforcement Learning (RL)

Definition

- States: $s \in S$
- Actions: $a \in A$
- Transition: $\Pr(s_t | s_{t-1}, a_{t-1})$
- Rewards: $r \in \mathbb{R}$
- Reward model: $\Pr(r_t | s_t, a_t)$
- Discount factor: $0 \leq \gamma \leq 1$
- Horizon (i.e., # of time steps): h

Data: $(s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_h, a_h, r_h)$

Goal: find optimal policy π^*

Inverse Reinforcement Learning (IRL)

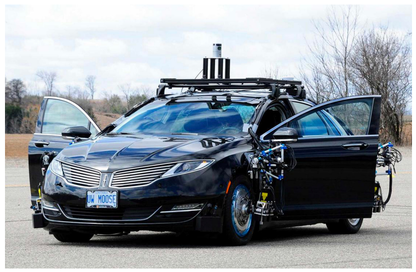
Definition

- States: $s \in S$
- Optimal actions: $a^* \in A$
- Transition: $\Pr(s_t | s_{t-1}, a_{t-1})$
- Rewards: $r \in \mathbb{R}$
- Reward model: $\Pr(r_t | s_t, a_t)$
- Discount factor: $0 \leq \gamma \leq 1$
- Horizon (i.e., # of time steps): h

Data: $(s_0, a_0^*, s_1, a_1^*, \dots, s_h, a_h^*)$

Goal: find $\Pr(r_t | s_t, a_t)$ for which expert actions a^* are optimal

IRL Applications



autonomous driving

robotics

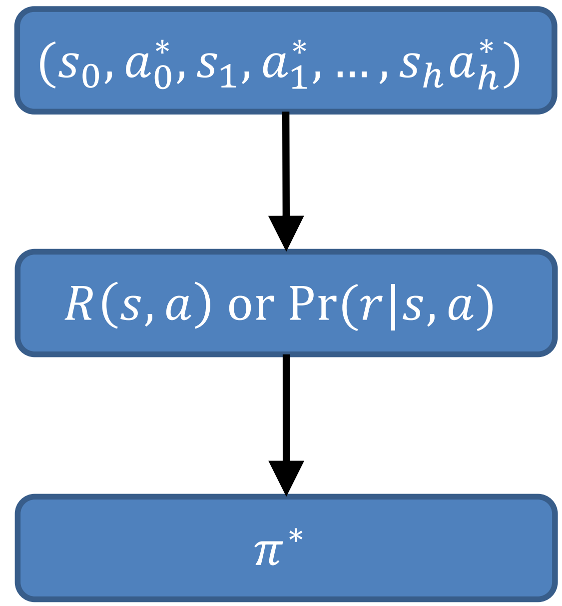
Advantages

- No assumption that state-action pairs are i.i.d.
- Transfer reward function to new environments/tasks

IRL Techniques

General approach:

1. Find reward function for which expert actions are optimal
2. Use reward function to optimize policy in same or new environments



Broad categories of IRL techniques

- Feature matching
- Maximum margin IRL
- Maximum entropy IRL
- Bayesian IRL

Feature Expectation Matching

- Normally: find R such that π^* chooses the same actions a^* as expert
- Problem: we may not have enough data for some states (especially continuous states) to properly estimate transitions and rewards

- Note: rewards typically depend on features $\phi_i(s,a)$

$$\text{e.g., } R(s,a) = \sum_i w_i \phi_i(s,a) = \boldsymbol{w}^T \boldsymbol{\phi}(s,a)$$

- Idea: Compute feature expectations and match them

Feature Expectation Matching

Let $\boldsymbol{\mu}^e(s_0) = \frac{1}{N}\sum_{n=1}^{N}\sum_t \gamma^t \boldsymbol{\phi}(s_t^{(n)}, a_t^{(n)})$
be the average feature count of expert e
(where n indexes trajectories)

Let $\boldsymbol{\mu}^\pi(s_0)$ be the expected feature count of policy π

Claim: If $\boldsymbol{\mu}^\pi(s) = \boldsymbol{\mu}^e(s)\ \forall s$ then $V^\pi(s) = V^e(s)\ \forall s$

Proof

Features: $\boldsymbol{\phi}(s,a) = (\phi_1(s,a), \phi_2(s,a), \phi_3(s,a), \dots)^T$
Linear reward function: $R_{\boldsymbol{w}}(s,a) = \sum_i w_i \phi_i(s,a) = \boldsymbol{w}^T \boldsymbol{\phi}(s,a)$

Discounted state visitation frequency:

$$\psi_{s_0}^{\pi}(s') = \delta(s', s_0) + \gamma \sum_s \psi_{s_0}^{\pi}(s) \Pr(s'|s, \pi(s))$$

Value function:

$$\begin{aligned} V^{\pi}(s) &= \sum_{s'} \psi_s^{\pi}(s') R_{\boldsymbol{w}}(s', \pi(s')) \\ &= \sum_{s'} \psi_s^{\pi}(s') \boldsymbol{w}^T \boldsymbol{\phi}(s', \pi(s')) \\ &= \boldsymbol{w}^T \sum_{s'} \psi_s^{\pi}(s') \boldsymbol{\phi}(s', \pi(s')) \\ &= \boldsymbol{w}^T \ \boldsymbol{\mu}^{\pi}(s) \end{aligned}$$

Hence: $\boldsymbol{\mu}^{\pi}(s) = \boldsymbol{\mu}^{e}(s)$

→ $\boldsymbol{w}^T \boldsymbol{\mu}^{\pi}(s) = \boldsymbol{w}^T \boldsymbol{\mu}^{e}(s)$

→ $V^{\pi}(s) = V^{e}(s)$

Indeterminacy of Rewards

- Learning $R_{\boldsymbol{w}}(s,a) = \boldsymbol{w}^T \boldsymbol{\phi}(s,a)$ amounts to learning $\boldsymbol{w}$

- When $\boldsymbol{\mu}^{\pi}(s) = \boldsymbol{\mu}^{e}(s)$, then $V^{\pi}(s) = V^{e}(s)$,
  but $\boldsymbol{w}$ can be anything since

$$\boldsymbol{\mu}^{\pi}(s) = \boldsymbol{\mu}^{e}(s) \;\rightarrow\; \boldsymbol{w}^T\boldsymbol{\mu}^{\pi}(s) = \boldsymbol{w}^T\boldsymbol{\mu}^{e}(s)\ \forall \boldsymbol{w}$$

- We need a bias to determine $\boldsymbol{w}$
- Ideas:
  - Maximize the margin
  - Maximize entropy

Maximum Margin IRL

- **Idea:** select reward function that yields the greatest minimum difference (margin) between the Q-values of the expert actions and other actions

$$margin = \min_{s}\left[Q(s,a^*) - \max_{a\neq a^*} Q(s,a)\right]$$

Maximum Margin IRL

Let $\boldsymbol{\mu}^{\pi}(s,a) = \boldsymbol{\phi}(s,a) + \gamma \sum_{s'} \Pr(s'|s,a)\, \boldsymbol{\mu}^{\pi}(s')$

Then $Q^{\pi}(s,a) = \boldsymbol{w}^T \boldsymbol{\mu}^{\pi}(s,a)$

Find $\boldsymbol{w}^*$ that maximizes margin:

$$\boldsymbol{w}^* = argmax_{\boldsymbol{w}} \min_{s} \left[\boldsymbol{w}^T \boldsymbol{\mu}^{\pi}(s,a^*) - \max_{a \neq a^*} \boldsymbol{w}^T \boldsymbol{\mu}^{\pi}(s,a)\right]$$

$$\text{s.t. } \boldsymbol{\mu}^{\pi}(s,a) = \boldsymbol{\mu}^{e}(s,a)\ \forall s,a$$

Problem: maximizing margin is somewhat arbitrary since it doesn't allow suboptimal actions to have values that are close to optimal

Maximum Entropy

Idea: Among models that match the expert's average features, select the model with maximum entropy

$$\max_{P(\tau)} H(P(\tau))$$
$$\text{s.t. } \frac{1}{|\text{data}|}\sum_{\tau\in data} \boldsymbol{\phi}(\tau) = E[\boldsymbol{\phi}(\tau)]$$

Trajectory: $\tau = (s_0^\tau, a_0^\tau, s_1^\tau, a_1^\tau, \dots, s_h^\tau, a_h^\tau)$
Trajectory feature vector: $\boldsymbol{\phi}(\tau) = \sum_t \gamma^t \boldsymbol{\phi}(s_t^\tau, a_t^\tau)$
Trajectory cumulative reward: $R(\tau) = \boldsymbol{w}^T \boldsymbol{\phi}(\tau) = \sum_t \gamma^t \boldsymbol{w}^T \boldsymbol{\phi}(s_t^\tau, a_t^\tau)$

Probability of a trajectory: $P_{\boldsymbol{w}}(\tau) = \frac{e^{R(\tau)}}{\sum_\tau e^{R(\tau)}} = \frac{e^{\boldsymbol{w}^T\boldsymbol{\phi}(\tau)}}{\sum_{\tau'} e^{\boldsymbol{w}^T\boldsymbol{\phi}(\tau')}}$

Entropy: $H(P(\tau)) = -\sum_\tau P(\tau) \log P(\tau)$

Maximum Likelihood

Maximum Entropy

$$\max_{P(\tau)} H(P(\tau))$$
$$\text{s.t. } \frac{1}{|\text{data}|}\sum_{\tau\in data}\phi(\tau) = E[\phi(\tau)]$$

Dual objective: This is equivalent to maximizing the log likelihood of the trajectories under the constraint that $P(\tau)$ takes an exponential form:

$$\max_{\boldsymbol{w}} \sum_{\tau\in data} \log P_{\boldsymbol{w}}(\tau)$$
$$\text{s.t. } P_{\boldsymbol{w}}(\tau) \propto e^{\boldsymbol{w}^T\boldsymbol{\phi}(\tau)}$$

Maximum Log Likelihood (LL)

$$\boldsymbol{w}^* = argmax_{\boldsymbol{w}} \frac{1}{|data|} \sum_{\tau \in data} \log P_{\boldsymbol{w}}(\tau)$$

$$= argmax_{\boldsymbol{w}} \frac{1}{|data|} \sum_{\tau \in data} \log \frac{e^{\boldsymbol{w}^T \boldsymbol{\phi}(\tau)}}{\sum_{\tau'} e^{\boldsymbol{w}^T \boldsymbol{\phi}(\tau')}}$$

$$= argmax_{\boldsymbol{w}} \frac{1}{|data|} \sum_{\tau \in data} \boldsymbol{w}^T \boldsymbol{\phi}(\tau) - \log \sum_{\tau'} e^{\boldsymbol{w}^T \boldsymbol{\phi}(\tau')}$$

Gradient: $\nabla_{\boldsymbol{w}} LL = \frac{1}{|data|} \sum_{\tau \in data} \boldsymbol{\phi}(\tau) - \sum_{\tau''} \frac{e^{\boldsymbol{w}^T \boldsymbol{\phi}(\tau'')}}{\sum_{\tau'} e^{\boldsymbol{w}^T \boldsymbol{\phi}(\tau')}} \boldsymbol{\phi}(\tau'')$

$$= \frac{1}{|data|} \sum_{\tau \in data} \boldsymbol{\phi}(\tau) - \sum_{\tau''} \frac{e^{\boldsymbol{w}^T \boldsymbol{\phi}(\tau'')}}{\sum_{\tau'} e^{\boldsymbol{w}^T \boldsymbol{\phi}(\tau')}} \boldsymbol{\phi}(\tau'')$$

$$= \frac{1}{|data|} \sum_{\tau \in data} \boldsymbol{\phi}(\tau) - \sum_{\tau''} P_{\boldsymbol{w}}(\tau'')\, \boldsymbol{\phi}(\tau'')$$

$$= E_{data}[\boldsymbol{\phi}(\tau)] - E_{\boldsymbol{w}}[\boldsymbol{\phi}(\tau)]$$

Gradient estimation

Computing $E_{\boldsymbol{w}}[\phi(\tau)]$ exactly is intractable due to exponential number of trajectories. Instead, approximate by sampling.

$$E_{\boldsymbol{w}}[\phi(\tau)] \approx \frac{1}{n} \sum_{\tau \sim P_{\boldsymbol{w}}(\tau)} \phi(\tau)$$

Importance sampling: Since we don't have a simple way of sampling τ from $P_{\boldsymbol{w}}(\tau)$, sample τ from a base distribution $q(\tau)$ and then reweight τ by $P_{\boldsymbol{w}}(\tau)/q(\tau)$:

$$E_{\boldsymbol{w}}[\phi(\tau)] \approx \frac{1}{n} \sum_{\tau \sim q(\tau)} \frac{P_{\boldsymbol{w}}(\tau)}{q(\tau)} \phi(\tau)$$

We can choose $q(\tau)$ to be a) uniform, b) close to demonstration distribution, or c) close to $P_{\boldsymbol{w}}(\tau)$

Maximum Entropy IRL Pseudocode

Assumption: Linear rewards $R_{\boldsymbol{w}}(s,a) = \boldsymbol{w}^T \boldsymbol{\phi}(s,a)$

Input: expert trajectories $\tau_e \sim \pi_{expert}$ where $\tau_e = (s_1, a_1, s_2, a_2, \ldots)$
Initialize weights $\boldsymbol{w}$ at random
Repeat until stopping criterion

Expert feature expectation: $E_{\pi_{expert}}[\boldsymbol{\phi}(\tau)] = \frac{1}{|data|} \sum_{\tau_e \in data} \boldsymbol{\phi}(\tau_e)$

Model feature expectation:

Sample n trajectories: $\tau \sim q(\tau)$

$E_{\boldsymbol{w}}[\phi(\tau)] = \frac{1}{n} \sum_{\tau} \frac{P_{\boldsymbol{w}}(\tau)}{q(\tau)} \boldsymbol{\phi}(\tau)$

Gradient: $\nabla_{\boldsymbol{w}} LL = E_{\pi_{expert}}[\boldsymbol{\phi}(\tau)] - E_{\boldsymbol{w}}[\boldsymbol{\phi}(\tau)]$

Update model: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \nabla_{\boldsymbol{w}} LL$

Return $\boldsymbol{w}$

Non-Linear Rewards

Suppose rewards are non-linear in $\boldsymbol{w}$

e.g., $R_{\boldsymbol{w}}(s, a) = NeuralNet_{\boldsymbol{w}}(s, a)$

Then $R_{\boldsymbol{w}}(\tau) = \sum_t \gamma^t R_{\boldsymbol{w}}(s_t^\tau, a_t^\tau)$

Likelihood: $LL(\boldsymbol{w}) = \frac{1}{|data|} \sum_{\tau \in data} R_{\boldsymbol{w}}(\tau) - \log \sum_{\tau'} e^{R_{\boldsymbol{w}}(\tau')}$

Gradient: $\nabla_{\boldsymbol{w}} LL = E_{data}[\nabla_{\boldsymbol{w}} R_{\boldsymbol{w}}(\tau)] - E_{\boldsymbol{w}}[\nabla_{\boldsymbol{w}} R_{\boldsymbol{w}}(\tau)]$

Maximum Entropy IRL Pseudocode

General case: Non-linear rewards $R_w(s,a)$

Input: expert trajectories $\tau_e \sim \pi_{expert}$ where $\tau_e = (s_1, a_1, s_2, a_2, \dots)$
Initialize weights w at random
Repeat until stopping criterion

Expert feature expectation: $E_{\pi_{expert}}[\nabla_{\boldsymbol{w}} R_{\boldsymbol{w}}(\tau_e)] = \frac{1}{|data|} \sum_{\tau_e \in data} \nabla_{\boldsymbol{w}} R_{\boldsymbol{w}}(\tau_e)$

Model feature expectation:

Sample n trajectories: $\tau \sim q(\tau)$

$E_{\boldsymbol{w}}[\nabla_{\boldsymbol{w}} R_{\boldsymbol{w}}(\tau)] = \frac{1}{n} \sum_{\tau} \frac{P_{\boldsymbol{w}}(\tau)}{q(\tau)} \nabla_{\boldsymbol{w}} R_{\boldsymbol{w}}(\tau)$

Gradient: $\nabla_{\boldsymbol{w}} LL = E_{\pi_{expert}}[\nabla_{\boldsymbol{w}} R_{\boldsymbol{w}}(\tau)] - E_{\boldsymbol{w}}[\nabla_{\boldsymbol{w}} R_{\boldsymbol{w}}(\tau)]$

Update model: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \nabla_{\boldsymbol{w}} LL$

Return $\boldsymbol{w}$

Policy Computation

Two choices:
1) Optimize policy based on $R_{\boldsymbol{w}}(s, a)$ with favorite RL algorithm
2) Compute policy induced by $P_{\boldsymbol{w}}(\tau)$.

Induced policy: probability of choosing a after s in trajectories
Let (s, a, τ) be a trajectory that starts with s, a and then continues with the state action-pairs of τ

$$\pi_{\boldsymbol{w}}(a|s) = P_{\boldsymbol{w}}(a|s)$$
$$= \frac{\sum_{\tau} P_{\boldsymbol{w}}(s,a,\tau)}{\sum_{a',\tau'} P_{\boldsymbol{w}}(s,a',\tau')}$$
$$= \frac{\sum_{\tau} e^{R_{\boldsymbol{w}}(s,a)+\gamma R_{\boldsymbol{w}}(\tau)}}{\sum_{a',\tau'} e^{R_{\boldsymbol{w}}(s,a')+\gamma R_{\boldsymbol{w}}(\tau')}}$$

Demo: Maximum Entropy IRL

Finn, C., Levine, S., & Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. *ICML*.