Inverse RL and Reward Learning from Preferences

Instructor: Daniel Brown

[Some slides adapted from Sergey Levine (CS 285) and Alina Vereshchaka (CSE4/510)]
Course feedback is open

• Extra credit if class response rate is 70% or higher
  • Sliding scale if we reach 70%:
    • Extra credit points = \( \frac{\text{response\_rate\_percentage}}{10} \)
Reward Learning (Inverse Reinforcement Learning)

Why? What is the human’s reward function?
Why not just imitate behavior? (Behavioral Cloning)

What would the human do?

Policy $\pi$

Action

Observation

Observation

Action
Human Intent Inference
Inverse Reinforcement Learning

- Given
  - MDP without a reward function
  - Demonstrations from an optimal policy $\pi^*$

- Recover the reward function $R$ that makes $\pi^*$ optimal
Imitation Learning

**Behavioral Cloning**

- Answers the “How?” question
- Mimic the demonstrator
- Learn mapping from states to actions
- Computationally efficient
- Compounding errors

\[ \Rightarrow \pi \]

**Inverse Reinforcement Learning**

- Answers the “Why?” question
- Explain the demonstrator’s behavior
- Learn a reward function capturing the demonstrator’s intent
- Can require lots of data and compute
- Better generalization. Can recover from arbitrary states

\[ \Rightarrow R \Rightarrow \pi \]
IRL Example: Teaching a robot to navigate through demonstrations
Toy version
What is the reward?

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R( ) = ?
R( ) = ?
R( ) = ?
R( ) = ?
R( ) = ?
What is the reward?

R(□)=?
R(□)=?
R(□)=?
R(□)=?
R(□)=?
What is the reward?

\[ R(\text{blue}) = ? \]
\[ R(\text{yellow}) = ? \]
\[ R(\text{green}) = ? \]
\[ R(\text{red}) = ? \]
\[ R(\text{white}) = ? \]
What is the reward?

\[
\begin{align*}
R(\text{blue}) &= +1 \\
R(\text{yellow}) &= 0 \\
R(\text{green}) &= 0 \\
R(\text{red}) &= -1 \\
R(\text{white}) &= 0
\end{align*}
\]
What is the reward?

- $R(\square) = +10$
- $R(\square) = 0$
- $R(\square) = 0$
- $R(\square) = -10$
- $R(\square) = 0$
What is the reward?

\[
\begin{align*}
R(\text{blue }) &= +10 \\
R(\text{yellow }) &= -1 \\
R(\text{green }) &= -1 \\
R(\text{red }) &= -10 \\
R(\text{white }) &= -1
\end{align*}
\]
What is the reward?

\[ R(\text{blue}) = 0 \]
\[ R(\text{yellow}) = 0 \]
\[ R(\text{green}) = 0 \]
\[ R(\text{red}) = 0 \]
\[ R(\text{white}) = 0 \]
What is the reward?

\[ R(\text{blue}) = c \]
\[ R(\text{yellow}) = c \]
\[ R(\text{green}) = c \]
\[ R(\text{red}) = c \]
\[ R(\text{white}) = c \]
Inverse Reinforcement Learning Formalism

- Given
  - MDP without a reward function
  - Demonstrations from an optimal policy \( \pi^* \)

- Recover the reward function \( R \) that makes \( \pi^* \) optimal

- Ill-Posed Problem
  - Infinite number of reward functions that can make \( \pi^* \) optimal
    - Trivial all zero reward
    - Constant reward
    - \( aR + c \) (positive scaling \( a > 0 \), and affine shifts)
How would you do this?

- maximize likelihood of $D$ given $R$
Basic IRL Algorithm

- Start with demonstrations, $D$
- Guess initial reward function $R_0$
- $\hat{R} = R_0$
- Loop:
  - Solve for optimal policy $\pi^*_\hat{R}$
  - Compare $D$ and $\pi^*_\hat{R}$
  - Update $\hat{R}$ to try and make $D$ and $\pi^*_\hat{R}$ more similar
Flashback: Approximate Q-Learning

\[ Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \ldots + w_n f_n(s, a) \]

- Q-learning with linear Q-functions:

  transition \( = (s, a, r, s') \)

  difference \( = [r + \gamma \max_{a'} Q(s', a')] - Q(s, a) \)

  \[ Q(s, a) \leftarrow Q(s, a) + \alpha \text{ [difference]} \]

  \[ w_i \leftarrow w_i + \alpha \text{ [difference]} f_i(s, a) \]

  \( \text{Exact Q's} \)

  \( \text{Approximate Q's} \)

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state’s features

- Formal justification: online least squares
Feature count matching

• Assume the reward function is a linear combination of features:

\[ R(s) = w^T \phi(s) = w_1 \phi_1(s) + w_2 \phi_2(s) + \ldots \]

• Value function becomes linear combination of (discounted) feature expectations:

\[
V_{\pi_R} = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) \right]
\]

Feature count matching

• Assume the reward function is a linear combination of features:

$$R(s) = w^T \phi(s)$$

• Value function becomes linear combination of (discounted) feature expectations:

$$V^\pi_R = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t w^t \phi(s_t) \right]$$

Feature count matching

• Assume the reward function is a linear combination of features:

\[ R(s) = w^T \phi(s) \]

• Value function becomes linear combination of (discounted) feature expectations:

\[ V_R^\pi = w^T \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t \phi(s_t) \right] = w^T \mu_\pi \]

Inverse reinforcement learning: feature matching
(Abbeel and Ng 2004, Syed and Schapire 2007)

• If $\|\mathbf{w}\|_1 \leq 1$, then

$$V_{R*}^{\pi} - V_{R}^{\pi_{robot}} = \mathbf{w}^T (\mu_{\pi*} - \mu_{\pi_{robot}}) \leq \|\mu_{\pi*} - \mu_{\pi_{robot}}\|_\infty$$

If feature expectations match, then expected returns are identical.

Idea: Can we update the reward guess $\hat{R}$ so the feature counts get closer?
Problem: Many different policies can lead to same expected feature counts
Maximum Entropy IRL (Ziebart et al. 2008)

- Collect $M$ demonstrations $D = \{\tau_1, ..., \tau_M\}$
- Initialize reward weights $w$
- Loop
  - Solve for (soft) optimal policy $\pi(a|s)$ via Value Iteration
  - Solve for expected feature counts of $\pi(a|s)$
  - Compute weight update $w \leftarrow w + \alpha(\mu_D - \mu_\pi)$

\[
P(\tau) = \frac{e^{Rw(\tau)}}{Z}
\]

\[
R(s) = w^T \phi(s)
\]
Soft Value Iteration

\[ \pi_{\Theta}(A_t | S_t) = e^{Q_{\pi_{\Theta}}^{\text{soft}}(A_t, S_t)} - V_{\pi_{\Theta}}^{\text{soft}}(S_t) \]

\[ V_{\pi_{\Theta}}^{\text{soft}}(S_t) = \log \sum_{A_t \in A} e^{Q_{\pi_{\Theta}}^{\text{soft}}(A_t, S_t)} \]

Policy is a softmax policy.

\[ Q(A, S) - \log \frac{e^{Q(b, S)}}{b} \]

\[ e^{Q(A, S)} - \log \frac{e^{Q(b, S)}}{b} \]

\[ e^{Q(A, S)} - \log \frac{e^{Q(b, S)}}{b} \]

\[ = e^{Q(A, S)} \]

\[ \frac{1}{\sum_{b} e^{Q(b, S)}} \]
Soft Maximum

• Let $a > b$

\[ \log(e^a + e^b) = \log(e^a + e^b) + \log e^{-b} + \log e^b \]

\[ = \log((e^a + e^b) e^{-b}) + b \]

\[ = \log(e^{a-b} + 1) + b \]

\[ = \log(2e^a) \]

\[ = \log(2) + \log e^a = \log(2) + a \]

• If $a = b$

\[ \log(e^a + e^b) = \log(2e^a) \]

\[ = \log(2) + \log e^a = \log(2) + a \]

• In general $\max\{x_1, x_2, \ldots, x_n\} \leq \log \sum \frac{e^{x_i}}{x_i} \leq \max\{x_1, \ldots, x_n\} + \log n$
Soft Value Iteration

\[ \pi_{\Theta} (A_t | S_t) = e^{Q_{\pi_{\Theta}}^{\text{soft}}(A_t, S_t)} - V_{\pi_{\Theta}}^{\text{soft}}(S_t) \]

\[ V_{\pi_{\Theta}}^{\text{soft}}(S_t) = \log \sum_{A_t \in A} e^{Q_{\pi_{\Theta}}^{\text{soft}}(A_t, S_t)} \]

\[ Q_{\pi_{\Theta}}^{\text{soft}}(A_t, S_t) = R_{\Theta}(S_t, A_t) + \sum_{S' \in S} P_T(S' | A_t, S_t) V_{\pi_{\Theta}}^{\text{soft}}(S') \]

- Initialize value of terminal states to 0 and other values to \(-\infty\)
- Repeat:
  - Solve for Q
  - Solve for V
Watch This: Scalable Cost-Function Learning for Path Planning in Urban Environments

Markus Wulfmeier¹, Dominic Zeng Wang¹ and Ingmar Posner¹

Fig. 1: Schema for training neural networks in the Maximum Entropy paradigm for IRL.
Another way to look at MaxEnt IRL

\[ P(\tau) = \frac{e^{R_\theta(\tau)}}{Z} \]

- Maximum Likelihood Estimation
- Find reward function that maximizes the log likelihood of the demonstration trajectories:

\[
\max_{\theta} \frac{1}{N} \sum_{\tau \in D} R_\theta(\tau) - \log Z
\]
How to avoid fully solving MDP

\[
\max_{\theta} \frac{1}{N} \sum_{\tau \in D} R_\theta(\tau) - \log Z \quad Z = \int e^{R_\theta(\tau)} d\tau
\]

- Estimate \( Z \) with a finite set of trajectories \( Z_\tau \).
- Loop:
  - Update parameters \( \theta \) so demonstrations have higher reward than trajectories in \( Z_\tau \).
  - Update \( Z_\tau \)
How to make this more tractable

Relative Entropy Inverse Reinforcement Learning

Abdeslam Boularias  Jens Kober  Jan Peters
Max-Planck Institute for Intelligent Systems 72076 Tübingen, Germany
{abdeslam.boularias,jens.kober,jan.peters}@tuebingen.mpg.de

Learning Objective Functions for Manipulation

Mrinal Kalakrishnan*, Peter Pastor*, Ludovic Righetti†, and Stefan Schaal†
kalakris@usc.edu, pastorsa@usc.edu, ludovic.righetti@i3.epfl.ch, sschaal@usc.edu
*CLMC Lab, University of Southern California, Los Angeles CA 90089
†Max Planck Institute for Intelligent Systems, Tübingen, Germany 72076

Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization

Chelsea Finn  Sergey Levine  Pieter Abbeel
University of California, Berkeley, Berkeley, CA 94709 USA

\[ P(\tau) = \frac{e^{R_\theta(\tau)}}{Z} \]

Uniform sampling to approximate Z.

Noisy perturbations of demonstrations to approximate Z

Use current policy to approximate Z.

Alternate between a few steps of reward updates and a few steps of policy updates.
Finn et al. “Guided Cost Learning.” 2016
GANs (Generative Adversarial Networks)
GAIL (Generative Adversarial Imitation Learning)

Ho and Ermon, 2016
What if we don’t want just a single reward estimate?

• Can we get a samples from the full Bayesian posterior?

\[ P(R|D) \propto P(D|R)P(R) \]
Markov Chain Monte Carlo (MCMC)

Markov chain:

\[ P(X_1) \quad P(X_t|X_{t-1}) \]

Stationary Distribution:

\[ P_\infty(X) = P_{\infty+1}(X) = \sum_x P(X|x)P_\infty(x) \]

MCMC is a sampling approach for Bayesian inference where we construct a Markov chain such that the stationary distribution is the posterior distribution we care about.
MCMC (Metropolis Hastings Algorithm)

- We want to sample from $P(R|D)$
- Start with random sample $r_0$
- Loop
  - Sample $r' \sim q(R_{t+1}|r_t)$
  - With probability $\min\left\{1, \frac{P(r'|D)}{P(r_t|D)}\right\}$ set $x_{t+1} = x'$
  - Else set $r_{t+1} = r_t$

Assume $q$ is symmetric. For example, a Gaussian distribution with mean $x_t$ and standard deviation $\sigma$

Accept!

Reject!

Normalizing constant cancels in the ratio!
Bayesian Inverse Reinforcement Learning
(Ramachandran and Amir 2007)

• Assume demonstrator is Boltzman rational
  • Demonstrator follows a softmax policy with inverse temperature $c$

$$P(D|R) = \prod_{(s,a) \in D} \frac{e^{cQ^*(s,a,R)}}{\sum_{b \in A} e^{cQ^*(s,b,R)}}$$

$Q^*(s,a,R) = \text{How much reward will I expect to see if I take action a in state s and act optimally thereafter.}$
Bayesian Inverse Reinforcement Learning
(Ramachandran and Amir 2007)

• Assume demonstrator is Boltzmann rational
  • Demonstrator follows a softmax policy with inverse temperature $c$

$$P(D|R) = \prod_{(s,a) \in D} \frac{e^{cQ^*(s,a,R)}}{\sum_{b \in A} e^{cQ^*(s,b,R)}}$$

$$P((s,\leftarrow)|R) = \frac{e^{Q^*(s,\leftarrow,R)}}{e^{Q^*(s,\leftarrow,R)} + e^{Q^*(s,\rightarrow,R)}}$$
Bayesian Inverse Reinforcement Learning
(Ramachandran and Amir 2007)

• Assume demonstrator is Boltzman rational
  • Demonstrator follows a softmax policy with inverse temperature $c$

\[
P(D|R) = \prod_{(s,a) \in D} \frac{e^{cQ^*(s,a,R)}}{\sum_{b \in A} e^{cQ^*(s,b,R)}}
\]

• Perform Bayesian inference (MCMC) to sample from posterior distribution

\[
P(R|D) \propto P(D|R)P(R)
\]
Applications of Bayesian IRL

• Active Learning
• Uncertainty Estimation
• Demonstration Sufficiency
Autonomous Assessment of Demonstration Sufficiency via Bayesian Inverse Reinforcement Learning

Demos

Am I able to learn a policy that performs within 5% of the expert with high confidence?

YES.

Learned policy
RL from Human Feedback (RLHF)
RL from Human Preferences

https://arxiv.org/abs/1706.03741
Why would you want to learn a reward from ranked examples?
Inverse Reinforcement Learning

Prior approaches ...

1. Typically couldn’t do much better than the demonstrator.
2. Were hard to scale to complex problems.

Inverse Reinforcement Learning

Prior approaches ...

1. Typically couldn’t do much better than the demonstrator. Find a reward function that explains the ranking, allowing for extrapolation.
2. Were hard to scale to complex problems.

Inverse Reinforcement Learning

Prior approaches ...

1. Typically couldn’t do much better than the demonstrator.
Find a reward function that explains the ranking, allowing for extrapolation.
2. Were hard to scale to complex problems.
Reward learning becomes a supervised learning problem.

Trajectory-ranked Reward Extrapolation (T-REX)

Pre-ranked demonstrations

Trajectory-ranked Reward Extrapolation (T-REX)

Reward Function

\[ R_\theta : S \to \mathbb{R} \]

Examples of S:

- Current Robot Joint Angles and Velocities
  - \( \rightarrow 0.5 \)
  - \( \rightarrow -0.7 \)
Reward Function

\[ R_\theta : S \to \mathbb{R} \]

Examples of S:

Current Robot Joint Angles and Velocities

- \( \rightarrow 0.5 \)
- \( \rightarrow 0.7 \)

Short Sequence of Images

- \( \rightarrow 0.9 \)
- \( \rightarrow -1.2 \)
Binary Classification and the Cross Entropy Loss

https://www.v7labs.com/blog/cross-entropy-loss-guide
Flashback: How should we parameterize our policy?

- We need to be able to do two things:
  - Sample actions $a_t \sim \pi_\theta (\cdot | s_t)$
  - Compute log probabilities $\log \pi_\theta (a_t | s_t)$
- Categorical (classifier over discrete actions)
  - Typically, you output a value $x_i$ for each action (class) and then the probability is given by a softmax equation

$$
\pi_\theta (a_i | s) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}
$$
Cross Entropy

\[ H(p, q) = - \sum_{x \in \text{classes}} p(x) \log q(x) \]

True probability distribution (one-shot)

Your model's predicted probability distribution

\[
S \xrightarrow{\text{Softmax}} f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \quad CE = - \sum_i^C t_i \log(f(s)_i)
\]
Example: Image Classification

Input image

NN Layers

Logits (L)

Softmax

Output probabilities (P)

Classes

Dog
Cat
Horse
Cheetah

Cross-Entropy

$D(S, L) = -\sum_i L_i \log(S_i)$
Trajectory-ranked Reward Extrapolation (T-REX)

\[ \mathcal{L}(\theta) = - \sum_{\tau_i \prec \tau_j} \frac{\exp \sum_{s \in \tau_i} R_\theta(s)}{\exp \sum_{s \in \tau_i} R_\theta(s) + \exp \sum_{s \in \tau_j} R_\theta(s)} \]
Trajectory-Ranked Reward Extrapolation (T-REX)

\[ f(s)_i = \frac{e^{s_i}}{\sum_j e^{s_j}} \]

\[ CE = -\sum_i t_i \log(f(s)_i) \]

Minimize cross-entropy loss

\[ \mathcal{L}(\theta) = -\sum_{\tau_i < \tau_j} \frac{\exp \sum_{s \in \tau_i} R_\theta(s)}{\exp \sum_{s \in \tau_i} R_\theta(s) + \exp \sum_{s \in \tau_j} R_\theta(s)} \]

\[ \sum_{s \in \tau_1} R_\theta(s) < \sum_{s \in \tau_2} R_\theta(s) \]
Trajectory-ranked Reward Extrapolation (T-REX)

Given pre-ranked demos, reward learning can be formulated as a standard supervised learning task.

Minimize cross-entropy loss

\[
\mathcal{L}(\theta) = - \sum_{\tau_i < \tau_j} \frac{\text{exp} \sum_{s \in \tau_i} R_\theta(s)}{\text{exp} \sum_{s \in \tau_j} R_\theta(s)} + \frac{\text{exp} \sum_{s \in \tau_j} R_\theta(s)}{\text{exp} \sum_{s \in \tau_i} R_\theta(s)}
\]
T-REX can extrapolate beyond the performance of the best demo

“Autonomous Driving” in Atari

Best demo (Score = 84)  

T-REX (Score = 520)

Uses only 12 ranked demonstrations
Atari Breakout

Best of 12 demos

Behavioral Cloning

GAIL (Ho and Ermon 2016)

T-REX
What if you don’t have explicit preference labels?

Learning from a learner [ICML’19]

自动偏好标签生成 [CoRL’20]
Automatic Rankings via Noise Injection

• Assumption: Demonstrator is significantly better than a purely random policy.

• Provides automatic rankings as noise increases.

• Generates a large diverse set of ranked demonstrations.

Disturbance-based Reward Extrapolation (D-REX)

Behavioral Cloning

$\pi_{BC}$

Disturbance-based Reward Extrapolation (D-REX)

Behavioral Cloning \( \xrightarrow{\downarrow} \) Automatic Rankings via Noise Injection

\[ \pi_{BC} \]

\[ \epsilon = 1.0 \quad \epsilon = 0.2 \quad \epsilon = 0.01 \]
Disturbance-based Reward Extrapolation (D-REX)

Behavioral Cloning $\implies$ Automatic Rankings via Noise Injection $\implies$ T-REX

$\pi_{BC} \downarrow \Upsilon \leftarrow \epsilon = 1.0 \Upsilon \leftarrow \epsilon = 0.2 \Upsilon \leftarrow \epsilon = 0.01$
Disturbance-based Reward Extrapolation (D-REX)

Behavioral Cloning $\rightarrow$ Automatic Rankings via Noise Injection $\rightarrow$ T-REX $\rightarrow$ Policy Optimization

$\pi_{BC}$ $\rightarrow$ $\epsilon = 1.0$ $\prec$ $\epsilon = 0.2$ $\prec$ $\epsilon = 0.01$ $\rightarrow$ D-REX Policy
Experiments

D-REX consistently outperforms the best demonstration as well as outperforming BC and GAIL.

AI systems can efficiently infer human intent from suboptimal demonstrations.
T-REX only learns a maximum likelihood estimate of the reward function.
Reward Hacking

• Overfit to spurious correlations
• No consideration of alternative hypotheses
$P(R|D)$

$\pi$
Next time: LLMs and ChatGPT