Learning From Observing Behavior

CSCI 2951-F
Ron Parr
Brown University

Algorithms for Inverse RL
(Ng & Russell, ICML 00)

Differences From RL

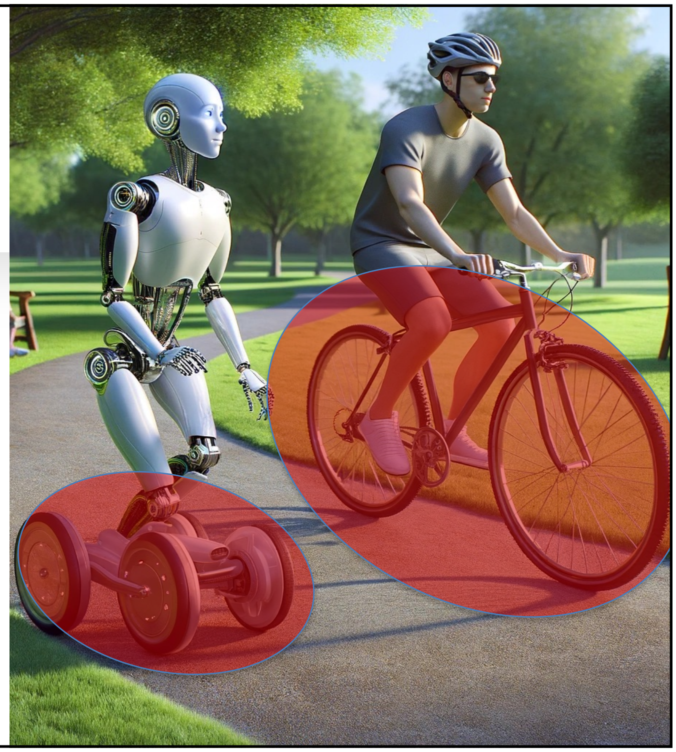
- RL: Learn optimal behavior from known/observed reward in unknown domain
- Learning from observed behavior:
  – Watch an agent acting in the domain
  – Reward function typically not known
  – Transition model may be known
  – Assume expert is acting (near) optimally
  – Goal: Produce similarly optimal behavior

When Does This Happen?

- Sometimes easier to demonstrate something than it specify it
- What are you optimizing when you ride a bicycle?
- When goal of RL is a **behavior**, rather than an **optimization**
  – Not obviously sensible for finance or other applications with costs/rewards grounded in real things
  – Possibly sensible in robotics where rewards often are not grounded, but are viewed as means to an end

Behavioral Cloning

- Just copy what the teacher does
- Turn things into a supervised learning problem
- Pro: Simple
- Cons:
  - Lack of robustness (limited training states)
  - (compounds with) lack of robustness to changes model/configuration/assumptions
- (Not our focus)



Inverse RL

- Unknown expert reward function
- Agent watches expert perform task under presumption of expert optimality
- Agent tries to infer reward function and produce an optimal policy for it

IRL vs. Utility Elicitation

- Utility elicitation:
  - Seeks to discover an agent's utility function
  - Useful for:
    - Advising people on good choices
    - Allocating resources in an equitable/desirable manner (population scale)
- IRL:
  - Seeks to discover an agent's reward function
  - Useful for:
    - Optimizing the same reward function on different hardware
    - Understanding underlying causes for how/why people act
- Reward function may be ***sparser*** than the utility function

What we know

- P_a = transition matrix with all actions = a
- $P_{\pi*}$ = transition matrix for π^*
- For all a:

$$P_a V^* - P_{\pi^*} V^* \leq 0$$

$$(P_a - P_{\pi^*})(I - \gamma P_{\pi^*})^{-1} R \leq 0$$

How To Use This

(Ng & Russell ICML 00)

- Set of linear constraints on R
- Could write an LP (with no objective function)
- Problem:
  - Under-constrained
  - R=0 is a solution for any $\pi*$

Workarounds

- WLOG, renumber actions so that $P_{\pi*} = P_{a1}$
- Assume m actions
- Create objective function to maximize difference between $\pi*$ and other actions:

$$\sum_{i=1}^{n}\sum_{j=2}^{m}\sum_{k=1}^{n}\left(P(S_k|S_i,a_1)-P\left(S_k\middle|S_i,a_j\right)\right)\left(\left(I-P_{a_1}\right)^{-1}R\right)[S_k]$$

Details

- Problem: R not bounded
- Solution: Add constraint $R(s) \leq R_{max}$

- Problem: May want a sparse R
- Solution: Add term to objective $-\lambda \|R\|_1$

Continuous State

- Problem: Can't write down all constraints

- Solution:
  – Sample constraints
  – Assume reward function linear combination features
  – Modify objective function to be a soft penalty
    (in case optimality of $\pi*$ isn't feasible)
  – Add bound on the norm of the weights on features

No Model Case

- Estimate value of policies by Monte Carlo

- Proposed algorithm in original Ng+Russell paper was not very practical/efficient but still important first step

Apprenticeship Learning via Inverse RL (Abbeel & Ng, ICML 04)

Assumptions

- We know P and γ well enough to solve the MDP if we have R
- Reward function is a linear combination of features
- We observe expert trajectories sampled from optimal policy
- Know start state or distribution

- Want to recover:
  - Expert reward function
  - Policy as good as expert's policy

Feature Expectations

- Suppose: $R = \sum_{i=1}^{K} w_i \phi_i$
- Expert policy gets:

$$\mathrm{E}\left[\sum_{j=0}^{t} \gamma^j r_j\right] = \mathrm{E}\left[\sum_{j=0}^{t} \sum_{i=1}^{K} \gamma^j w_i \phi_i(s_j)\right] = w^T E[\gamma \phi] = w^T \mu_E$$

Expected discounted sum of features expert gets

Matching Feature Expectations

- Policy w/same (discounted) feature expectations has same value of starting initial state distribution
- Find weights and a policy optimal WRT weights to match expert's feature expectations

The Hard Way

- Suppose:
  - We could enumerate all policies 1…n
  - Can compute feature expectations of each policy
- Set of constraints on w:

$$\forall i: w^T \mu_i \leq w^T \mu_E$$

Constraint Generation

- Problem: Exponential number of constraints
- Solution: Sequentially generate weight vectors that maximize the difference between expert return and return of policies we have generated so far (constraint generation)

Algorithm

- Assume we have generated $w_1 \ldots w_j$ so far
- $\mu_1 \ldots \mu_j$ are the feature expectations for the optimal policies given $w_1 \ldots w_j$
- Compute w_{j+1} using the following QP:

Maximize t, w_{j+1} subject to

$\forall i: w_{j+1}^T \mu_E \geq w_{j+1}^T \mu_i + t$

$\left\| w_{j+1} \right\|_2$

Idea: Propose a new set of reward that is maximally difference under expert feature expectations vs. current weight/policy pairs

Repeat until small t → near optimality

Properties

- In general constraint generation for math programs has no guarantees
- For k features, Abbeel and Ng guarantee ε-optimal performance in:
  - Iterations proportional to: k log(k)
  - Iterations proportional to: $1/\varepsilon^2 \log(1/\varepsilon)$

Maximum Entropy IRL

- Assumes high reward trajectories are exponentially more likely:

$$P_{\boldsymbol{\Phi}}(\tau) = \frac{1}{Z(\boldsymbol{\Phi})} \exp(R_{\boldsymbol{\Phi}}(\tau))$$

Normalization

- Is this reasonable?

MaxEnt RL Objective

- Maximize: $f(\boldsymbol{\phi}) = \sum_{\tau \in \mathcal{D}} \log \frac{1}{Z(\boldsymbol{\phi})} \exp(R_{\boldsymbol{\phi}}(\tau))$
- Gradient:

Optimal policy for current estimate of reward

$$\nabla_{\boldsymbol{\phi}} f = \left(\sum_{\tau \in \mathcal{D}} \nabla_{\boldsymbol{\phi}} R_{\boldsymbol{\phi}}(\tau) \right) - |\mathcal{D}| \sum_{\tau} P_{\boldsymbol{\phi}}(\tau) \nabla_{\boldsymbol{\phi}} R_{\boldsymbol{\phi}}(\tau)$$
$$= \left(\sum_{\tau \in \mathcal{D}} \nabla_{\boldsymbol{\phi}} R_{\boldsymbol{\phi}}(\tau) \right) - |\mathcal{D}| \sum_{s} b_{\gamma,\boldsymbol{\phi}}(s) \sum_{a} \pi_{\boldsymbol{\phi}}(a \mid s) \nabla_{\boldsymbol{\phi}} R_{\boldsymbol{\phi}}(s,a)$$

State occupancy probs
Under current optimal policy
(can be computed by dynamic programming)

MaxEnt RL Interpretation

- For reward as linear combination of features

$$\boxed{\left(\sum_{\tau \in \mathcal{D}} \nabla_{\boldsymbol{\phi}} R_{\boldsymbol{\phi}}(\tau) \right)} - \boxed{|\mathcal{D}| \sum_{s} b_{\gamma,\boldsymbol{\phi}}(s) \sum_{a} \pi_{\boldsymbol{\phi}}(a \mid s) \nabla_{\boldsymbol{\phi}} R_{\boldsymbol{\phi}}(s,a)}$$

- Gradient of R is just reward feature values
- Gradient is 0 when observed feature counts match feature counts of policy π

MaxENT RL in Practice

- Iterates between:
  - Updating reward weights
  - Updating policy WRT reward weights
- Stop when gradient is close to 0, implying policy feature expectations match expert's
- Requires multiple approximations to work in large state spaces

GAIL

- Generative Adversarial Imitation Learning
- Q: What do GANs do best?
- A: Match probability distributions
- Insight: A policy is a distribution over actions, which implies a distribution over trajectories
- In practice:
  - Impressive experiment results vs. previous approaches
  - Be cautious about issues w/GANs (they are fiddly)

The Arc of Imitation Learning

- Linear programs: ~2000
- Quadratic programs: ~2005
- GANs: ~2015

- Can you guess what's next?

Asking a Different Question

- Assumptions in inverse RL:
  - Expert is always optimal
  - No observed metric of performance
- Observe a single score per trajectory
- Olympic judges problem:
  - We see the final scores
  - What are skaters optimizing?

Learning From Scored Trajectories

- Infer reward weights from total trajectory scores
- Reduces IRL to linear regression
- See El Asri et al. [2013], Burchfiel et al. [2016]

LfD Summary

- Includes mimicking experts and inverse RL
- Mimicking is simpler, but arguably less robust
- Inverse RL:
  - Typically assumes (near) expert optimality
  - Approaches combine optimization techniques with distribution matching techniques