17-740 Spring 2025

Algorithmic Foundations of Interactive Learning

**Lecture 18: Imitation Learning as Game-Solving**

Lecturer : Gokul Swamy Scribe: Jim Wang, Jason Wei, Sangyun Lee, Yang Zh

# 18.1 Outline

In this lecture, we will focus on three key questions:

1. Why do we need interaction in imitation learning?
   *A: to be able to tell that we've made a mistake that compounds. (necessity)*
2. What else do we need to tell which mistakes matter?
   *A: information about the set of rewards we could be judged on. (sufficiency)*
3. How do we learn a policy that recovers from mistakes that matter if we don't know what the reward function is?
   *A: find the policy that is the least distinguishable from the expert's under **any** reward function in the moment set $\mathcal{R}$.*

# 18.2 Why do we need interaction in IL?

At a high level, it is to be able to tell that we've made a mistake that compounds, i.e. one that compounds over the horizon.

## 18.2.1 The Pitfalls of Behavioral Cloning

Behavioral cloning is an offline / supervised learning approach to imitation learning. Given some set of expert demonstrations, the BC optimization problem is to solve

$$\arg\min_{\pi\in\Pi} \mathbb{E}_{\xi\sim\pi_E}\left[-\log\left(\prod_h^H \pi(a_h|s_h)\right)\right] = \arg\min_{\pi\in\Pi}\sum_h^H \mathbb{E}_{s_h,a_h\sim\pi_E}[-\log\pi(a_h|s_h)].$$

As we discussed in the DAgger lecture, an $\epsilon$ probability of making a mistake can often lead to a performance that scales *quadratically* with the horizon, rather than the expected linear scaling. This phenomenon is known as *compounding errors*.

### 18.2.2 What went wrong?

At train time, offline algorithms like BC look at

$$\ell_{BC}(\pi) = \mathbb{E}_{s_h,a_h\sim\pi_E}[-\log \pi(a_h|s_h)].$$

However, at test time, the learner actually sees $\mathbb{E}_{s_h\sim\pi} \neq \mathbb{E}_{s_h\sim\pi_E}$. Thus, $p_{\text{test}}(x) \neq p_{\text{train}}(x)$, which is called *covariate shift* in the machine learning literature.

Not being able to avoid compounding errors is a fundamental property of *all* offline algorithms, as we now sketch via a lower bound example.

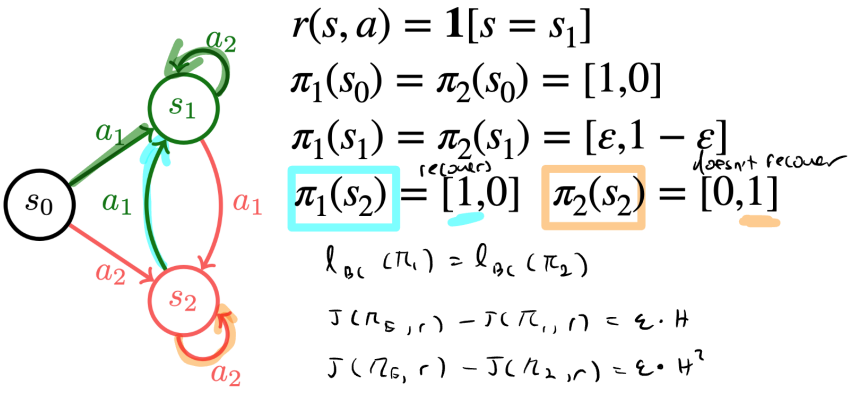


Figure 18.1: Suppose we have an expert policy $\pi_E$ that at $h = 1$, takes action $a_1$ to go from state $s_0$ to $s_1$, and then afterwards takes action $a_2$ to stay in state $s_1$ forever. Now suppose we have two policies $(\pi_1, \pi_2)$, both of which take $a_1$ at $h = 1$ and with probability $\varepsilon$ take action $a_1$ to end up in $s_2$ while at state $s_1$. However, these two policies differ in terms of their behavior at $s_2$, which is outside of the support of the expert. Specifically, $\pi_1$ moves immediately back to $s_1$ via $a_1$ when it gets to $s_2$, but $\pi_2$ stays in $s_2$ indefinitely by taking $a_2$. In this MDP, only state $s_1$ gives a reward of 1 per timestep ($r(s, a) = \mathbf{1}[s = s_1]$).

Observe **no offline IL algorithm can tell the difference between** $\pi_1$ **and** $\pi_2$ because they only differ on states outside of the support of the demonstrations. Mathematically, we have $J(\pi_E, r) - J(\pi_1, r) = \varepsilon H$, and $J(\pi_E, r) - J(\pi_2, r) = \varepsilon H^2$ (quadratic), despite $\ell_{BC}(\pi_1) = \ell_{BC}(\pi_2)$. In short, offline IL fundamentally can't differentiate between policies that recover and those that don't, which means it can't avoid compounding errors.

#### 18.2.2.1 Where does the $\varepsilon$ come from?

A natural question after going through the above example is where $\varepsilon$ comes from. Roughly speaking, there are three sources, each of which might matter for a particular problem:

1. Finite-sample error: limited number of expert demos.
   *A: Get more data.*

2. Optimization error: imperfect search over policy class.
   *A: Use more compute (e.g. use a better optimizer).*

3. Misspecification error: irreducible error from $\pi_e \notin \Pi$.
   *A: Use an interactive algorithm.*

We'll focus mostly on the third case for this lecture as it is the most fundamental / hard to solve via scaling data or compute. As we'll discuss further, interaction effectively generates samples from the test distribution, allowing us to handle covariate shift.

## 18.3 What else do we need to tell which mistakes matter?

At the highest level, we need some sort of *information about the* ***set*** *of rewards we could be judged on.* We'll use $\mathcal{R}$ to denote this set for the rest of the lecture.

### 18.3.1 Not All Mistakes are Made Equal

We now sketch a simple example of why some knowledge of $\mathcal{R}$ is necessary to avoid compounding errors.

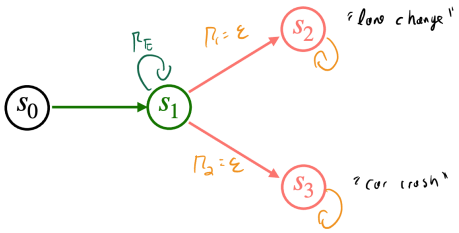


Figure 18.2: Assume $\pi_E$ goes from $s_0$ to $s_1$ and stays there indefinitely. Now at $s_1$, $\pi_1$ has probability $\varepsilon$ to go to $s_2$ and stay there indefinitely, and $\pi_1$ has probability $\varepsilon$ to go to $s_3$ and stay there indefinitely.

To be able to pick between $\pi_1$ and $\pi_2$, **we need to be able to tell which mistakes cost us performance**, which is what $\mathcal{R}$ represents. For example, if we know $s_2$ corresponds to a

"lane change" but $s_3$ corresponds to a "car crash," it becomes much easier to select between $\pi_1$ and $\pi_2$.

### 18.3.2 Moments in Imitation Learning

For example, in autonomous driving, the set of *moments* (basis rewards) could include:

$$\mathcal{R} = \{\text{distance to nearest car, distance to center of lane, distance from edge of road, distance from nearest person, speed/speed limit, ...}\}$$

Note that knowing $\mathcal{R}$ is often *much* easier than knowing $r$. Concretely, rather than needing to know the precise trade-offs between factors (e.g. how much better it is to arrive five minutes earlier by being 1 inch closer to the nearest vehicle), we simply need to know the set of criteria we could be judged under.

For the rest of this lecture, we'll assume *reward realizability*, i.e. $r \in \mathcal{R}$. However, given we don't know precisely which $f \in \mathcal{R}$ is R, we will attempt to **be good under *all* $f \in \mathcal{R}$!**

## 18.4 How do we learn a policy that recovers from mistakes that matter if we don't know what the reward function is?

More formally, we can attempt to have a bounded performance difference from the expert under all reward functions in $\mathcal{R}$, which naturally leads to the form of a zero sum game:

$$\max_{\pi \in \Pi} \min_{f \in \mathcal{R}} J(\pi, f) - J(\pi_E, f), \quad \text{where} \quad J(\pi, f) = \mathbb{E}_{\xi \sim \pi}\left[\sum_h^H f(s_h, a_h)\right] \tag{18.1}$$

Some notes on the above objective:

- If we dropped the first term and just tried to optimize $\max_{\pi \in \Pi} \min_{f \in \mathcal{R}} -J(\pi_E, f)$, we could just pick a reward function that is a constant everywhere. This is because the expert (as well as any other policy) is optimal under this reward function. This is what people mean when they talk about the ill-posedness of an inverse problem.
- This maximin objective is linear in $f$ but *not* convex in $\pi$, but we'll ignore the non-convexity for now. Also note that the range of this payoff is $[-H, H]$.

### 18.4.1 Game-Solving Searches the Pareto Frontier

Before we discuss the performance bounds of inverse RL, we now provide some intuition about what the solution (i.e. Nash equilibrium) to the above game looks like.

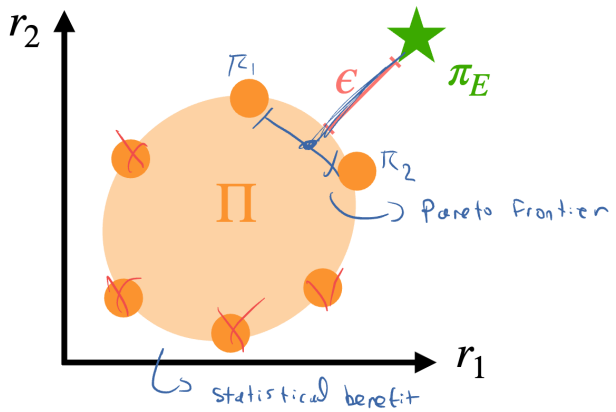


Figure 18.3: For each candidate policy in $\Pi$, we can look at its performance on each possible reward function $r \in \mathcal{R} = \{r_1, r_2\}$. The Pareto frontier is then the set of policies that cannot increase its performance on one reward function without decreasing the performance on some other reward function $(\alpha\pi_1 + (1-\alpha)\pi_2)$. A solution to the game must live on this Pareto frontier, effectively reducing the search space we look over, providing a *statistical* benefit (i.e. fewer samples needed to find a good policy than behavioral cloning).

We will now do the simplest proof in this entire course:

**Lemma 1** *Assume $\hat{\pi}$ is an $\varepsilon$-approximate equilibria for the IRL game and for simplicity assume $\pi_E \in \Pi$ and $r \in \mathcal{R}$. Then,*

$$J(\pi_E, r) - J(\pi, r) \leq \mathcal{O}(\varepsilon H). \tag{18.2}$$

**Proof:**

$$\min_{f \in \mathcal{R}} \frac{J(\pi_E, f) - J(\hat{\pi}, f)}{H} \leq \varepsilon \Rightarrow J(\pi_E, r) - J(\hat{\pi}, r) \leq \mathcal{O}(\varepsilon H). \tag{18.3}$$

■

Notice that the performance bound is linear in the horizon, which means we provably avoid compounding errors. Note that we don't need a queryable expert like DAgger.

To summarize, there are three key facts to remember about inverse RL:

1. Inverse RL lets avoid compounding errors without needing access to extra expert interaction.

2. Inverse RL reduces the search space of policies to just those that are on the Pareto frontier.

3. Inverse RL isn't merely picking a reward that makes the expert look optimal—it is fundamentally game-theoretic.

## 18.5 How do we solve the IRL Game?

We'll use BR to mean "Best Response" and NR to mean "No Regret". As we discussed in the game-solving lecture, we can solve a game by running a no-regret player against a best-response player or against another no-regret player. We have special names for these two flavors of game solving in IRL: *primal* and *dual*. In greater detail:

| | Dual | Primal |
|---|---|---|
| Policy Update | **BR**: RL | **NR**: GD (PG Step) |
| Reward Update | **NR**: OGD | **NR**: OGD |

A popular dual algorithm is Maximum Entropy Inverse RL (MaxEnt IRL) and a popular primal algorithm is Generative Adversarial Imitation Learning (GAIL). We will now discuss the former in greater detail as it is usually not presented from a game-theoretic lens.

## 18.6 MaxEnt Inverse RL

We seek the policy with maximum entropy that matches the expert's moments:

$$\max_{\pi} \mathbb{E}_{\xi\sim\pi}\left[\sum_{h}^{H} -\log \pi(a_h|s_h)\right] \quad \text{(maximize causal entropy)}$$

$$\text{s.t.} \forall f \in \mathcal{R},\ \mathbb{E}_{\xi\sim\pi}\left[\sum_{h}^{H} f(s_h,a_h)\right] = \mathbb{E}_{\xi\sim\pi_E}\left[\sum_{h}^{H} f(s_h,a_h)\right] \quad \text{(match expert moments)}$$

It may seem a bit daunting to solve a trajectory-level maximum entropy problem (rather than a single-step problem). However, as we will now derive, the solution follows a beautiful

form. First, forming the Lagrangian, we have

$$\max_{\lambda \in \mathbb{R}^{|\mathcal{R}|}} \min_{\pi} \mathbb{E}_{\xi \sim \pi} \left[ \sum_{h}^{H} - \log \pi(a_h | s_h) \right] + \sum_{f \in \mathcal{R}} \lambda^f (J(\pi, f) - J(\pi_E, f)). \tag{18.4}$$

Observe that this is just an entropy-regularized variant of our above inverse RL game. Another way to derive the game is via this constraint satisfaction perspective. [1]

Let us now solve this game via a dual strategy (i.e. with best responses over policies). For some fixed $\lambda_t$, we can write the best-response over $\pi$ as

$$\min_{\pi} \mathbb{E}_{\xi \sim \pi} \left[ \sum_{h}^{H} \log \pi(a_h | s_h) \right] + J \left( \pi, \sum_{f \in \mathcal{R}} \lambda_t^f f \right). \tag{18.5}$$

This is because (1) we can drop the $J(\pi_E, f)$ as it is constant with respect to $\pi$, (2) $J$ is a linear function with respect to the second argument. Next, observing that both $J$ and causal entropy are expectations over trajectories sampled from $\pi$, we have

$$\pi_t = \arg\min_{\pi \in \Pi} \mathbb{E}_{\xi \sim \pi} \left[ \sum_{h}^{H} \left( - \log \pi(a_h | s_h) + \sum_{f \in \mathcal{R}} \lambda_t^f f(s_h, a_h) \right) \right]. \tag{18.6}$$

Observe that what remains is just a standard RL problem with a somewhat special reward:

$$r_t(s_h, a_h) \triangleq \log \pi(a_h | s_h) + \sum_{f \in \mathcal{R}} \lambda_t^f f(s_h, a_h). \tag{18.7}$$

We can therefore apply our standard RL tools like value iteration to solve this problem. Let us proceed backwards in time, starting with $h = H$. Because there is nothing left to do,

$$V_t^\star(s_H) = 0. \tag{18.8}$$

For the inductive step (i.e. $h \in [0, H-1]$), value iteration tells us to solve

$$\pi_t^\star(\cdot | s_h) = \min_{p \in \Delta(\mathcal{A})} \mathbb{E}_p \left[ \log p(a) + \sum_{f \in \mathcal{R}} \lambda_t^f f(s_h, a) + \mathbb{E}_{T(s_h, a)}[V_t^\star(s_{h+1})] \right] \tag{18.9}$$

Observe that this is a single-step maximum entropy problem – i.e. the kind we previously covered how to solve! Recall that for MaxEnt problems of the form

$$\min_{p \in \Delta(\mathcal{X})} -\mathbb{H}(p) + \mathbb{E}_p[m(x)] \tag{18.10}$$

[1] In the misspecified setting, there might be no policy that satisfies the constraints. Thus, it is common to regularize the Lagrange multipliers with a quadratic penalty $\sum_{f \in \mathcal{R}} \lambda_f^2$, sometimes called an *augmented Lagrangian* or AuLa for short.

have solutions of the form

$$p^\star(x) = \frac{\exp(m(x))}{\sum_{x' \in \mathcal{X}} \exp(m(x'))}. \tag{18.11}$$

Here, we have

$$m(a) = \sum_{f \in \mathcal{R}} \lambda_t^f f(s_h, a) + \mathbb{E}_{T(s_h,a)}[V_t^\star(s_{h+1})]. \tag{18.12}$$

Matching terms, we now know the solution to the above in closed form:

$$\pi_t^\star(a_h|s_h) = \frac{\exp\left(\sum_{f \in \mathcal{R}} \lambda_t^f f(s_h, a_h) + \mathbb{E}_{T(s_h,a_h)}[V_t^\star(s_{h+1})]\right)}{\sum_{a \in \mathcal{A}} \exp\left(\sum_{f \in \mathcal{R}} \lambda_t^f f(s_h, a) + \mathbb{E}_{T(s_h,a)}[V_t^\star(s_{h+1})]\right)}. \tag{18.13}$$

The fact that $\pi_t^\star$ takes the form of a softmax is what leads to the above procedure being referred to as *soft value iteration*. More generally, *soft RL* refers to entropy-regularized RL problems. Recall that this softmax form is also what we derived when discussing Hedge / Multiplicative Weights and the Natural Policy Gradient. Thus, even though we took a completely different route, we ended up with a strikingly similar solution!

We can now back up another timestep to get the next value function:

$$V_t^\star(s_h) = \mathbb{E}_{a_h \sim \pi_t^\star(s_h)}[\log \pi_t^\star(a_h|s_h) + \lambda_t^f f(s_h, a_h) + \mathbb{E}_{T(s_h,a_h)}[V_t^\star(s_{h+1})]. \tag{18.14}$$

To close, recall that Bellman's principle of optimality says that an optimal policy acts optimally at the current step and then acts optimally in the future. Intuitively, the above derivation is telling us that the maximum entropy policy is maximally random at the current step and then maximally random in the future.