Lecture 18: Inverse RL and Intent Inference

Logistics:

- Project Proposals. Feedback has been released. Come to office hours to discuss in more detail with any of the course staff.
- HW7 due on Mon. Won't cover material from today, so get started soon.

Slides: `https://docs.google.com/presentation/d/1pN9xKedfYfX7hSrXUVTu7KOYGWQ15DyUS6c0fYJQpVo/edit#slide=id.p`

1 Inverse RL

Problem statement for **inverse RL**:

- Input: a dataset of trajectories, $\{\tau\}$
- Output: a reward function for which these trajectories are optimal, $r(s,a)$.

Underlying question: *if the human were acting optimally, what reward function were they trying to optimize*? This problem is also known as *inverse optimal control*.

1.1 Demo.

1.2 A Naïve Approach

Choose whichever reward function gives highest reward to the demonstrated behavior:

$$\max_{r(s,a)} \mathbb{E}_{\pi^*}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\right]. \tag{1}$$

- Limitation 1: Multiplying the reward by $2\times$ or adding $+10$ to the reward will result in assigning *even* higher reward to the demonstrated behavior.
- Limitation 2: Will assign very low rewards to outcomes that are hard to achieve. E.g., somewhat surprising how often snowboarders and skateboarders in the Olympics fail to stick the landing. One might presume that they are bad. One might presume that they are trying to fall. Or, one might (correctly) presume that what they are trying to do is very very hard, so succeeding even a small fraction of the time is commendable.

So, if we want to build an inverse RL algorithm that works, we're going to need a way of accounting for the difficulty of different tasks. That is, we'll need a way of accounting for how hard it is to maximize this reward function. Intuitively, this means that we'll want an objective that looks something like the following:

$$\max_{r(s,a)} \mathbb{E}_{\pi^*}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\right] - \textsc{Difficulty}(r(\cdot,\cdot)). \tag{2}$$

The key question is how we choose this measure of difficulty.
Q: How to measure this difficulty?

Intuitively, you might measure difficulty by looking at how much reward you get if you *try* to maximize this reward function. That is, if you learn a policy

$$\pi_r(a \mid s) \in \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\right], \tag{3}$$

you could measure difficulty as

$$\textsc{Difficulty}(r(\cdot,\cdot)) = \mathbb{E}_{\pi_r}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\right]. \tag{4}$$

Note that the reward function appears in two places on the right hand side. This intuition is approximately correct. We'll show that the *gradient* of Eq. 2 using this measure of difficulty is exactly the right thing to do. To build up to that, we first have to take a detour into energy-based models.

2 Energy-Based Models

Q: Who has seen energy-based models before?

Ordinarily when we think about machine learning, we think about a model $p_\theta(y \mid x)$ or $p_\theta(x)$ that supports two operations:

- sampling
- assigning probabilities to samples.

However, the latter requirement often restricts the set of models we might consider. For example, when dealing with continuous data, we often assume that $p_\theta(y \mid x) = \mathcal{N}(\mu_\theta(x), \sigma_\theta(x))$; this corresponds to assuming that conditional distribution has a Gaussian form. However, many real world datasets are not Gaussian; they often have multiple modes.

For example, think about the states visited by the volunteer in the demo at the start of today's class. They often picked up one object; they often picked up the other object. If you think about (say) the position of their hand, the distribution over these hand positions is very multi-modal. What sort of model should we use to represent this distribution?

Energy-based models provide a flexible way of modeling such distributions. The key idea is to learn an energy function $E_\theta(x)$ (i.e., a neural network) that maps inputs to scalar values. We will want higher probability inputs to receive lower energies; intuitively, you can think about data as being generated by rolling a ball around on this energy landscape, so you're more likely to find the ball at points of lower energy.[1]

Formally, we'll say that an energy function $E_\theta(x)$ encodes the following probability distribution:

$$p_\theta(x) \triangleq \frac{e^{-E_\theta(x)}}{\int e^{-E_\theta(x')}dx'}. \tag{5}$$

The negative sign in the exponent means that lower energies correspond to higher probabilities. The denominator is a normalizing constant that ensures that the probability distribution integrates to one. This sort of probabilistic model is known as an **energy-based model**.

Energy based models are good because they are flexible and can encode many different types of distributions, including multi-modal distributions. We are going to ignore sampling from these models; the interested reader should check out (Du and Mordatch, 2019; Liu and Wang, 2016). We'll now turn to fitting these models to data. Given a dataset $p^*(x)$, we'll want to find the parameters θ s.t. $p_\theta(x) \approx p^*(x)$. We will measure this discrepancy using the forward KL divergence (i.e., maximum likelihood):

$$\max_\theta \mathbb{E}_{p^*(x)}[\log p_\theta(x)]. \tag{6}$$

However, directly optimizing this objective is challenging because computing the estimated probability $p_\theta(x)$ requires estimating the normalizing constant $\int e^{-E_\theta(x')}dx'$. Note that we really only need the gradient of this objective. To start, we'll take the gradient of

[1] This analogy can be made exact via Stein Variational Gradient Descent (Liu and Wang, 2016).

the maximum likelihood objective (w.r.t. θ) and then substitute the definition of the energy-based model:

$$\nabla_\theta \mathbb{E}_{p^*(x)}[\log p_\theta(x)] = \nabla_\theta \mathbb{E}_{p^*(x)}\left[\log \frac{e^{-E_\theta(x)}}{\int e^{-E_\theta(x')}dx'}\right] \tag{7}$$

$$= \nabla_\theta \mathbb{E}_{p^*(x)}\left[-E_\theta(x) - \log \int e^{-E_\theta(x')}dx'\right] \tag{8}$$

$$= \nabla_\theta \mathbb{E}_{p^*(x)}[-E_\theta(x)] - \log \int e^{-E_\theta(x')}dx' \tag{9}$$

$$= -\mathbb{E}_{p^*(x)}[\nabla_\theta E_\theta(x)] - \frac{\int e^{-E_\theta(x)}\nabla_\theta - E_\theta(x)dx}{\int e^{-E_\theta(x')}dx'} \quad \text{(chain rule)}$$

$$\overset{(a)}{=} -\mathbb{E}_{p^*(x)}[\nabla_\theta E_\theta(x)] - \mathbb{E}_{p_\theta(x)}[\nabla_\theta - E_\theta(x)dx] \tag{10}$$

$$= -\mathbb{E}_{p^*(x)}[\nabla_\theta E_\theta(x)] + \mathbb{E}_{p_\theta(x)}[\nabla_\theta E_\theta(x)dx]. \tag{11}$$

In *(a)*, we recognize that the orange terms correspond to the PDF of the energy model, so we can replace the integral with an expectation. This gradient has an intuitive interpretation. The first term says that we look at data from the true distribution ($p^*(x)$) and decrease the energy assigned to those points (i.e., we want them to become local minima). The second term says that we look at data sampled from the energy model itself, and increase the energy assigned to those points. Intuitively, this corresponds to looking at data hallucinated by the model, and updating the model to assign lower probability to its own hallucinations.[2]

3 Maximum Entropy Inverse RL (Ziebart et al., 2008)

These energy-based models will provide the key for doing inverse RL. The connection is that we'll use the sum of rewards as the negative energy function.

3.1 Recall: MaxEnt RL

Recall from last time that we can interpret MaxEnt RL as performing a sort of distribution matching. The policy induces a certain distribution over trajectories:

$$\pi(\tau) = p_0(s_0)\prod_t p(s_{t+1} \mid s_t, a_t)\pi(a_t \mid s_t). \tag{12}$$

We now introduce a desired distribution over trajectories, with a higher weight associated with trajectories with higher returns:

$$p^*(\tau) = \tfrac{1}{Z}p_0(s_0)\prod_t p(s_{t+1} \mid s_t, a_t)e^{r(s_t,a_t)} = \tfrac{1}{Z}e^{\sum_t r(s_t,a_t)}p_0(s_0)\prod_t p(s_{t+1} \mid s_t, a_t), \tag{13}$$

where $Z = \int e^{\sum_t r(s_t,a_t)}p_0(s_0)\prod_t p(s_{t+1} \mid s_t, a_t)d\tau$. Note that this looks similar to an energy based model with energy function $E_\theta(\tau) = -\sum_t r(s_t, a_t)$. The difference is that there are some extra terms for the initial state distribution and the dynamics.

[2]While we've focused on unconditional models above, it is easy to extend these methods to conditional distributions.

In MaxEnt RL, we tried to align these two distributions, measuring the distance with a reverse KL divergence. There, we were optimizing over the policy (i.e., $p^{\pi}(\tau)$):

$$\min_{\pi(a|s)} D_{\mathrm{KL}}(\pi(\tau)\|p^*(\tau)) = \mathbb{E}_{\pi(\tau)}\left[\log \frac{\pi(\tau)}{p^*(\tau)}\right] \tag{14}$$

$$= \mathbb{E}_{\pi(\tau)}\Big[\cancel{\log p_0(s_0)} + \sum_t \big(\cancel{\log p(s_{t+1} \mid s_t, a_t)} + \log \pi(a_t \mid s_t)\big) \tag{15}$$

$$- \cancel{\log p_0(s_0)} - \sum_t \big(\cancel{\log p(s_{t+1} \mid s_t, a_t)} + r(s_t, a_t)\big) + \log Z\Big] \tag{16}$$

$$= -\mathbb{E}_{\pi(\tau)}\left[\sum_t (r(s_t, a_t) - \log \pi(a_t \mid s_t))\right] \cancel{+ \log Z}. \tag{17}$$

Note that the $\log Z$ term depends on the policy but not the reward function, so it can be ignored when optimizing w.r.t. the policy.

3.2 MaxEnt Inverse RL

In inverse RL, we assume that the policy $\pi(a \mid s)$ is given to us and we want to try to guess what reward function we're optimizing. We can directly use the same KL divergence as in MaxEnt RL (Eq. 14) to do this, optimizing over the reward function instead of the policy:

$$\min_{r(a|s)} D_{\mathrm{KL}}(\pi(\tau)\|p^*(\tau)) \tag{18}$$

$$= \mathbb{E}_{\pi(\tau)}\left[\log \frac{\pi(\tau)}{p^*(\tau)}\right] + \log Z \tag{19}$$

$$\overset{(a)}{=} -\mathbb{E}_{\pi(\tau)}\left[\sum_t (r(s_t, a_t) - \log \pi(a_t \mid s_t))\right] + \log \int e^{\sum_t r(s_t, a_t)} p_0(s_0) \prod_t p(s_{t+1} \mid s_t, a_t) d\tau \tag{20}$$

$$= -\mathbb{E}_{\pi(\tau)}\left[R(\tau) - \sum_t \log \pi(a_t \mid s_t)\right] + \log \int e^{R(\tau)} p_0(s_0) \prod_t p(s_{t+1} \mid s_t, a_t) d\tau. \tag{21}$$

In *(a)*, we used the same simplifications as in Eq. 17 and substituted the definition of Z. The last line introduces $R(\tau) = \sum_t r(s_t, a_t)$ to simplify notation. To optimize this objective w.r.t. the reward function, we will take gradients. The gradient will end up looking similar to the gradient of the energy based model (Eq. 11).

Q: Work on this gradient.

Working out the gradient. We will assume that reward function $r(s, a) = r_\theta(s, a)$ as parameters θ (i.e., it is a neural network mapping state-action pairs to scalar values) and

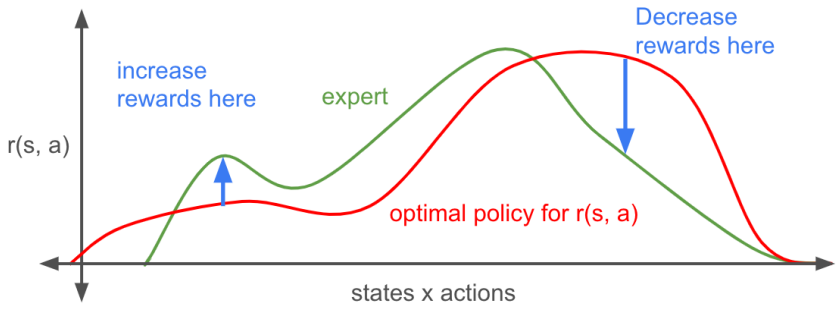

Figure 1: Illustration of the MaxEnt IRL gradients

will find the gradient of the objective w.r.t. these parameters:

$$\nabla_\theta \, D_{\mathrm{KL}}(\pi(\tau)\|p^*(\tau)) \tag{22}$$

$$= -\nabla_\theta \mathbb{E}_{\pi(\tau)} \left[R_\theta(\tau) - \sum_t \log \pi(a_t \mid s_t) \right] + \nabla_\theta \log \int e^{R_\theta(\tau)} p_0(s_0) \prod_t p(s_{t+1} \mid s_t, a_t) d\tau \tag{23}$$

$$= -\mathbb{E}_{\pi(\tau)} \left[\nabla_\theta R_\theta(\tau) - \nabla_\theta \cancel{\sum_t \log \pi(a_t \mid s_t)} \right] + \frac{\nabla_\theta \int e^{R_\theta(\tau)} p_0(s_0) \prod_t p(s_{t+1} \mid s_t, a_t) d\tau}{\int e^{R_\theta(\tau)} p_0(s_0) \prod_t p(s_{t+1} \mid s_t, a_t) d\tau} \tag{24}$$

$$= -\mathbb{E}_{\pi(\tau)} \left[\nabla_\theta R_\theta(\tau) \right] + \frac{\int \nabla_\theta R_\theta(\tau) \int e^{R_\theta(\tau)} p_0(s_0) \prod_t p(s_{t+1} \mid s_t, a_t) d\tau}{\int e^{R_\theta(\tau)} p_0(s_0) \prod_t p(s_{t+1} \mid s_t, a_t) d\tau} \tag{25}$$

$$= -\mathbb{E}_{\pi(\tau)} \left[\nabla_\theta R_\theta(\tau) \right] + \mathbb{E}_{p_r^*(\tau)} \left[\nabla_\theta R_\theta(\tau) \right] \tag{26}$$

$$= -\mathbb{E}_{\pi(\tau)} \left[\nabla_\theta \sum_r r_\theta(s_t, a_t) \right] + \mathbb{E}_{p_r^*(\tau)} \left[\nabla_\theta \sum_r r_\theta(s_t, a_t) \right]. \tag{27}$$

As before, we recognize that the orange terms correspond to the probability density of $p_r^*(\tau)$, allowing us to rewrite the integral as an expectation. Note that we have included the subscript "r" to indicate that this is the optimal trajectory distribution for reward function $r(s, a)$.

Intuitively, this gradient makes sense. The first terms says that we should look at states visited by the policy (the expert) and increase the reward on those states.[3] The second term says that we should find the optimal *MaxEnt* policy for the current estimated reward function $r_\theta(s, a)$ and decrease the rewards associated with those states. See Fig. 1.

3.3 Complete Method (Wulfmeier et al., 2015)

1. Initialize random reward function
2. Apply MaxEnt RL to this reward function
3. Update reward function using Eq. 27
4. Go to Step 2.

To complete the story, we return to the measure of difficulty we introduced in Eq. 2. There, we measured the difficulty of the reward function as how much reward we got if we tried to maximize that reward function. In the end, we see that this is almost exactly what MaxEnt IRL does – it subtracts off the reward you'd get if you tried to maximize this reward function *with a MaxEnt RL* policy. That last bit is what's missing from the heuristic introduced at the start of lecture.

[3] We're looking at the gradient of the divergence. To minimize the divergence, we take steps in the negative gradient direction, corresponding to the positive reward direction.

3.4 Special Cases

Goal Inference. In this setting we have a small number of candidate goals, and want to infer which a human is attempting to reach. This problem setting is important in human-robot interaction (Javdani et al., 2015), and is frequently studied in cognitive science (Baker et al., 2009)

Rationality Constant. (Ghosal et al., 2023) In this setting, we assume that the reward function is of the form $r(s,a) = \frac{1}{\beta} \cdot r^*(s,a)$ where $r^*(s,a)$ is known, and we just attempt to infer the "temperature" β. This temperature can loosely be interpreted as the expert's degree of rationality – how much reward maximization they are doing versus entropy maximization. This β corresponds to the α parameter we introduced when discussing MaxEnt RL.

Other notes. People usually talk about MaxEnt IRL (Ziebart et al., 2008) in terms of feature matching. But this doesn't really make sense when dealing with high-dimensional settings. This is why I've introduced this all in terms of states, rather than features. See Wulfmeier et al. (2015) for some more discussion about the deep RL setting.

4 Is MaxEnt RL Actually Useful?

- If you just want to mimic the expert, do BC or DAGGER. No need to convert demonstrations to a reward function and back to a policy.
- Prior work argues that inferring the reward function allows you to find a policy that is better than the expert. This argument is falicious because if violates the central assumption of MaxEnt IRL – that assumption that the expert was acting optimally. If you (correctly) apply MaxEnt IRL, you'll recover a reward function for which the expert's behavior is optimal, so if you then optimize that reward function with MaxEnt RL you'll get back that same policy.
- Inverse RL is useful in setting where we want to understand behavior (science, rather than engineering). E.g., in autonomous driving, we might want to know how much riders car about safety versus smoothness vs speed (Basu et al., 2017).
- Optimality assumption – humans are clearly not optimal, so inferring their reward functions is fraught. But it does result in a good predictive model (e.g., when are papers submitted for a conference)

References

Baker, C. L., Saxe, R., and Tenenbaum, J. B. (2009). Action understanding as inverse planning. *Cognition*, 113(3):329–349.

Basu, C., Yang, Q., Hungerman, D., Singhal, M., and Dragan, A. D. (2017). Do you want your autonomous car to drive like you? In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 417–425.

Du, Y. and Mordatch, I. (2019). Implicit generation and modeling with energy based models. *Advances in Neural Information Processing Systems*, 32.

Ghosal, G. R., Zurek, M., Brown, D. S., and Dragan, A. D. (2023). The effect of modeling human rationality level on learning rewards from multiple feedback types. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 5983–5992.

Javdani, S., Srinivasa, S. S., and Bagnell, J. A. (2015). Shared autonomy via hindsight optimization. *Robotics science and systems: online proceedings*, 2015.

Liu, Q. and Wang, D. (2016). Stein variational gradient descent: A general purpose bayesian inference algorithm. *Advances in neural information processing systems*, 29.

Wulfmeier, M., Ondruska, P., and Posner, I. (2015). Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., Dey, A. K., et al. (2008). Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA.