- encode image observations using auto-encoder
- build predictive model on auto-encoder latent states
- use model error as exploration bonus

Schmidhuber (1991) and Schmidhuber (2010):

- exploration bonus for model errors
- exploration bonus for model gradients
- many other variations

6.4 Learning without reward functions

What if we want to recover diverse behavior without any reward function at all? Why would we want to do this? Perhaps by doing so, we can

- learn skills without supervision, then use them to accomplish new goals,
- learn sub-skills to use with hierarchical refinforcement learning,
- explore the space of many other possible behaviors.

In this subsection, we will discuss the following:

- Definitions & concepts from information theory,
- Learning without a reward function by reaching goals,
- A *state distribution-matching* formulation of reinforcement learning,
- Is coverage of valid states a *good* exploration objective?
- Beyond state covering: covering the *space of skills* .

6.4.1 Definitions & concepts from information theory

There are some useful identities in information theory that are related. The first two are

$$p(x)$$

which is the distribution of overvation x and the entropy of the distribution $p(x)$

$$\mathcal{H}(p(x)) = -\mathbb{E}_{x\sim p(x)}[\log p(x)]$$

which measures how "braod" $p(x)$ is. Another new concept is **mutual information**, which is defined as

$$\begin{aligned}\mathcal{I}(x;y) &= D_{\mathrm{KL}}(p(x,y)\|p(x)p(y)) \\ &= E_{(x,y)\sim p(x,y)}\left[\log \frac{p(x,y)}{p(x)p(y)}\right] \\ &= \mathcal{H}(p(y)) - \mathcal{H}(p(y|x)).\end{aligned}$$

Intuitively, $\mathcal{I}(x;y)$ measures how "entangled" x and y are. Related to RL, we are intersted in

- $\pi(s)$ - the state marginal distribution of policy π,
- $\mathcal{H}(\pi(s))$ - the state marginal entropy of policy π (which quantifies *coverage* of policy π),

- $\mathcal{I}(s_{t+1}; a_t) = \mathcal{H}(s_{t+1}) - \mathcal{H}(s_{t+1}|a_t)$ - "empowerment", which can be viewed as quantifying "control" in an information-theoretic sense.

Remark. How to understand $\mathcal{I}(s_{t+1}; a_t)$? For the first term, maximizing $\mathcal{H}(s_{t+1})$ means that we want more possible states to be covered in the next state s_{t+1}. For the second term, we want to have more control, i.e., to minimize the entropy of the next state given the action a_t.

6.4.2 Learning without a reward function by reaching goals

What if we don't have a concrete goal, but instead we want the agent to learn to reach a "proposed" goal? We can then propose goals $z_g \sim p(z)$, then train the agent to reach the proposed goal, which ultimately leads to the agent learning to reach any goal from the distribution of $p(z)$.

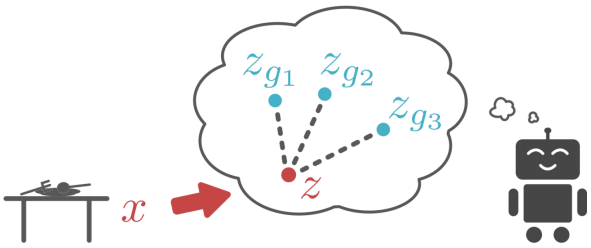

Figure 6.1: Learning to reach all goals in the distribution of $p(z)$.

To do this, we can adopt an algorithm similar to Alg. 32.

Why use a auto-encoder? How to understand $x_g \sim p_\theta(x_g|z_g)$? Why do we need $q_\phi(z_g|x_g)$?

Algorithm 32 Learning with proposed goals

1: **repeat**
2: Propose goal: $z_g \sim p(z)$ and $x_g \sim p_\theta(x_g|z_g)$
3: Attempt to reach goal using policy $\pi(a|x, x_g)$, and reach x'
4: Use data to update policy π
5: Use data to update $p_\theta(x_g|z_g)$ and $q_\phi(z_g|x_g)$
6: **until** some stopping criterion is met

Unfortunately, this algorithm might not work well, since the auto-encoder fashion of training would generate lots of similar goals, which is not what we want. Therefore, we may need to borrow some ideas from exploration. A simple solution is to reweight the state distribution $p(x)$ to make it diverse. Traditionally, in line 5, we adopt the following method:

$$\text{standard MLE: } \theta, \phi \leftarrow \arg\max_{\theta,\phi} \mathbb{E}[\log p(x')]$$

where we maximize the log-likelihood of the state x' that we actually observed. We can reweight the states to make them diverse:

$$\text{weighted MLE: } \theta, \phi \leftarrow \arg\max_{\theta,\phi} \mathbb{E}[w(x') \log p(x')]$$
$$w(x') = p_\theta(x')^\alpha$$
$$\text{key result: for any } \alpha \in [-1, 0), \mathcal{H}(p_\theta) \text{ actually increases}$$

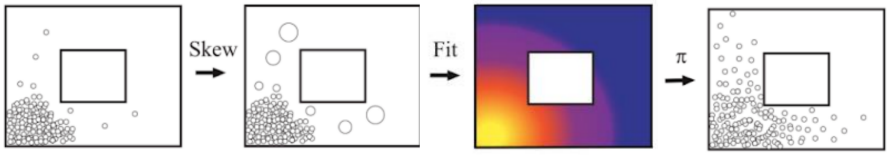

Figure 6.2: The Skew-Fit scheme.

The above analysis is pretty intuitive. Let's consider a more concrete example in RL. The objective to be optimized can be

$$\max \mathcal{H}(p(G)) - \mathcal{H}(p(G|S)) = \max \mathcal{I}(S; G) \tag{6.4.1}$$

where the first term $\mathcal{H}(p(G))$ measures the "diversity" of the goal distribution $p(G)$, and the second term $\mathcal{H}(p(G|S))$ measures the "control" of the agent in reaching the goal G given the current state S. Intuitively, as $\pi(a|S, G)$ gets better, the state S gets closer to G, which means $p(G|S)$ becomes more deterministic.

6.4.3 A state distribution-matching formulation

On one hand, the state marginal matching problem can be formulated as learning $\pi(a|s)$ so as to minimize $D_{KL}(p_\pi(s)\|p^*(s))$, where $p^*(s)$ is the target state distribution and $p_\pi(s)$ is the state distribution under policy π.

On the other hand, if a state is visited *often*, it is not *novel*. Therefore, we can add an exploration bonus to reward:

$$\tilde{r}(s) = \log p^*(s) - \log p_\pi(s).$$

We can then alternatively update π to maximize $\mathbb{E}[\tilde{r}(s)]$ and update the state distribution $p_\pi(s)$ to fit state marginal. However, this is **not** performing marginal matching, because each time we fit a policy to the states during one episode rather than the entire state distribution (Fig. 6.3). To actually perform marginal matching, we may consider averaging over all states so far by alternatively looping over step 1 and step 2:

1. learn policy π^k to maximize $\mathbb{E}[\tilde{r}(s)]$
2. update $p_{\pi^k}(s)$ to fit all states seen so far
3. return $\pi^*(a|s) = \sum_k \pi^k(a|s)$

Remark. $p_\pi(s) = p^*(s)$ is the Nash equilibrium for two player game between π^k and p_{π^k}.

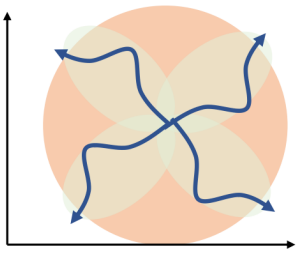

Figure 6.3: The outer orange circle is the target state distribution. The four lighter green ellipses are the state distributions observed in each episode. Darker blue lines are the policies learned in each episode.

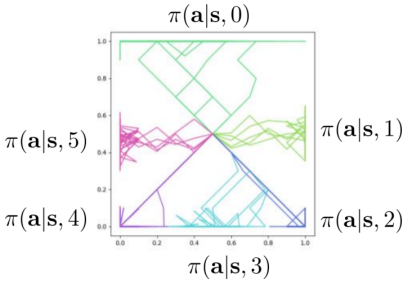

Figure 6.4: Performing diverse tasks is more like reaching different regions. Performing different skills is more like reaching the same region following different paths with the same color.

For more details, take a look at Lee et al. (2019) and Hazan et al. (2019).

Is state entropy really a good objective?

Is state entropy really a good objective?

Recall that the objective for the Skew-Fit method is $\max \mathcal{H}(p(G)) - \mathcal{H}(p(G|S)) = \max \mathcal{I}(S;G)$ and the objective for the state marginal matching problem is $\max D_{KL}(p_\pi(s) \| p^*(s))$. But when is this a good idea? Check out Gupta et al. (2018) and Hazan et al. (2019) for more details.

6.4.4 Learning diverse skills

Generally speaking, reaching diverse goals is not the same as performing diverse tasks. Intuitively, different skills should visit different state-space regions (Fig. 6.4).

Take a look at Gregor et al. (2016) and Eysenbach et al. (2018) for further details.

Learning diverse skills

6.5 Improving RL with Imitation

This section is covered in CS285 after Fall 2021 and is directly copied from the notes taken by harryhzhangOG.

Imitation learning is simple, stable supervised learning, but it requires demonstrations, and it must address distributional shift. Furthermore, the state of the art imitation learning can only be as good as the demo. In contrast, reinforcement learning can become arbitrarily good, but it requires reward function, and must address exploration. Also, it often does not have convergence guarantees.

6.5.1 Pretrain and Finetune

Can we somehow combine the two such that we have both demonstrations data and rewards? The answer is yes. The simplest idea that is practical and works is to pretrain with imitation learning and fine tune with RL. This idea is shown in Alg. 33.

Algorithm 33 Pretrain and Fine Tune

Require: Demonstrations data $\mathcal{D}$
1: Collect demonstration data $\mathcal{D} = \{(s_i, a_i)\}$
2: Initialize π_θ as $\max_\theta \sum_i \log \pi_\theta(a_i|s_i)$
3: **while** not done **do**
4: Run π_θ to collect experience
5: Improve π_θ with any RL algorithm

The problem with Alg. 33 is that in step 4, we might collect very bad experience due to distribution shift, so the first batch of data might be bad, thus destroying the initialization.

6.5.2 Off-policy RL

One way to mitigate the issue of forgetting the demonstrations is to use off-policy RL because off-policy RL can use any data, and if we let it use demonstrations as off-policy samples, since demonstrations are provided as data in every iteration, they are never forgotten. Furthermore, the policy can still become better than the demos, since it is not forced to mimic them. To achieve this, we could use off-policy policy gradients and off-policy Q-learning.

Recall policy gradients with importance sampling:

$$\nabla_\theta J(\theta) = \sum_{\tau \in \mathcal{D}} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\prod_{t'=1}^{t} \frac{\pi_\theta(a_{t'}|s_{t'})}{q(a_{t'}|s_{t'})} \right) \left(\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \right) \right]$$

The trick here is when we collect the sum of samples, in our samples, we include both our experience and demonstration. However, it seems a little weird because in policy gradients we actually want on-policy data, so why are we including off-policy data. To answer this question, let us build up some intuition by looking at the

Algorithm 34 Q-Learning with Replay Buffer and Target Network

Require: Some base policy for data collection; hyperparameter K and N

while true **do**
  Save target network parameters: $\phi' \leftarrow \phi$
  for N times **do**
    Collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using some policy, add it to replay buffer $\mathcal{B}$
    for K times **do**
      Sample a batch (s_i, a_i, s_i', r_i) from $\mathcal{B}$
      Set $y_i \leftarrow r(s_i, a_i) + \gamma \max_{a_i'} Q_{\phi'}(s_i', a_i')$
      Set $\phi \leftarrow \phi - \alpha \Sigma_i \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - y_i)$

optimal importance sampling distribution. Say we want to estimate $\mathbb{E}_{p(x)}[f(x)]$ using importance sampling:

$$\mathbb{E}_{p(x)}[f(x)] \simeq \frac{1}{N} \sum_i \frac{p(x_i)}{q(x_i)} f(x_i)$$

and it can be proven that

$$q \propto p(x)|f(x)|$$

gives us the smallest variance. Therefore, by taking off-policy demonstration samples, we are motivating importance sampling to use distributions that have higher reward than current policy in order to get closer to the optimal distribution. To construct the sampling distribution, first we need to figure out which distribution the demonstrations come from. First, we could use supervised behavioral cloning to learn π_{demo}. If the demonstration is from multiple distribution, we could instead use **fusion distribution** by

$$q(x) = \frac{1}{M} \sum_i q_i(x)$$

6.5.3 Q-learning with Demonstrations

Since Q-learning is already off-policy, there is actually no need to bother with importance weights like we did in policy gradients. Therefore, one simple solution is just drop demonstrations into the replay buffer.

We can modify Alg. 34 slightly such that we initialize $\mathcal{B}$ with some demonstrations data, and then do the same things as before.

6.5.4 Imitation as an Auxiliary Loss Function

Recall the imitation learning maximum likelihood training objective is

$$\sum_{(s,a) \sim \mathcal{D}_{demo}} \log \pi_\theta(a|s)$$

and the RL objective is

$$\mathbb{E}_{\pi_\theta}[r(s, a)]$$

to combine the two, we can come up with a hybrid objective:

$$\mathbb{E}_{\pi_\theta}[r(s, a)] + \lambda \sum_{(s,a) \sim \mathcal{D}_{demo}} \log \pi_\theta(a|s)$$

6.6 Further Readings

Chapter 11 Inverse Reinforcement Learning

So far in our RL algorithms, we have been assuming that the reward function is known a priori, or it is manually designed to define a task. What if we want to learn the reward function from observing an expert, and then use reinforcement learning? This is the idea of **inverse RL** , where we first figure out the reward function and then apply RL.

Why should we worry about learning rewards at all? From the imitation learning perspective, the agent learns via imitation by copying the *actions* performed by the expert, without any reasoning about outcomes of actions. However, the natural way that human learn through imitation is that human copy the *intent* of the expert, and thus might take very different actions. In RL, it is often the case that the reward function is ambiguous in the environment. For example, it is hard to hand-design a reward function for autonomous driving.

The inverse RL problem definition is as follows: we try to infer the *reward functions* from demonstrations, and then learn to maximize the inferred reward using any RL algorithm available. Formally, in inverse RL, we learn $r_\psi(s,a)$, and then use it to learn $\pi^*(a|s)$. However, this is an underspecified problem, because there are infinitely many reward function can explain the same behavior. One potential form is the **linear reward function**, which is a weighted sum of features:

$$r_\psi(s,a) = \sum_i \psi_i f_i(s,a) = \psi^\top \mathbf{f}(s,a), \tag{11.0.1}$$

or it could be a neural net with parameters ψ.

11.1 Feature Matching Inverse RL

For now, let us focus on the linear reward function design. Since it is a weighted sum of features, one natural interpretation to match the features is to match the **expectation** of important features. Let π^{r_ψ} be the optimal policy for reward function r_ψ, then to design the reward, we are picking ψ such that

$$\mathbb{E}_{\pi^{r_\psi}}[\mathbf{f}(s,a)] = \mathbb{E}_{\pi^*}[\mathbf{f}(s,a)] \tag{11.1.1}$$

The expectation of features under the unknown optimal policy right hand side expectation can be estimated using samples from expert, i.e., take N samples of features, and get the average. The left hand side expectation is a little involved. One way to do it is to use any RL algorithm to maximize r_ψ, which is defined using the right hand side samples, and then produce π^{r_ψ}, and then we can use this policy to generate more samples. However, this is still ambiguous because there could be multiple reward functions r_ψ that results in the same optimal policy π^{r_ψ}. One way to address this is using the **maximum margin principle**:

$$\max_{\psi,m} m \quad \text{s.t.} \quad \psi^\top \mathbb{E}_{\pi^*}[\mathbf{f}(s,a)] \geq \max_{\pi\in\Pi} \psi^\top \mathbb{E}_{\pi}[\mathbf{f}(s,a)] + m. \tag{11.1.2}$$

However, this is problematic when the space of policies is large and continuous, since there could be very similar policies. Hence, we need to "weight" the similarity between π and π^* so that similar policies do not need to abide by the m margin requirement. Using the "SVM trick" (with the use of Lagrangian dual), we can transform the above optimization into the following which also contains a function $D(\pi,\pi^*)$ that measures the similarity between policies:

$$\min_\psi \frac{1}{2}||\psi||^2 \quad \text{s.t.} \quad \psi^\top \mathbb{E}_{\pi^*}[\mathbf{f}(s,a)] \geq \max_{\pi\in\Pi} \psi^\top \mathbb{E}_{\pi}[\mathbf{f}(s,a)] + D(\pi,\pi^*). \tag{11.1.3}$$

However, such approaches have some issues. First, maximizing the margin is a bit arbitrary. Second, there is no clear model of expert suboptimality (can add slack variables), i.e., there is no reason why the expert is

sometimes not optimal. Furthermore, now we have a messy constrained optimization problem, which is not great for deep learning. For more details, see Ratliff et al. (2006) and Abbeel and Ng (2004).

11.2 Learning the Optimality Variable

Recall that in last chapter, we introduced the optimality variable $\mathcal{O}_t$ to indicate if the agent is acting optimally. It turns out that as we learn the reward function, we are also learning the optimality variable. The optimality variable is defined as $p(\mathcal{O}_t|s_t, a_t) = \exp(r_\psi(s_t, a_t))$. Since the reward parameter ψ is unknown, the optimality distribution should also depend on ψ: $p(\mathcal{O}_t|s_t, a_t, \psi)$. Recall that

$$p(\tau|\mathcal{O}_{1:T}, \psi) \propto p(\tau) \exp\left(\sum_t r_\psi(s_t, a_t)\right) \tag{11.2.1}$$

Note that we can ignore $p(\tau)$ in our optimiztion since it does not depend on ψ. We are given sample trajectories $\{\tau_i\}$ sampled from expert policy $\pi^*(\tau)$, we can maximize the log-likelihood of the observed trajectories using the maximum likelihood principle:

$$\max_\psi \frac{1}{N}\sum_{i=1}^N \log p(\tau_i|\mathcal{O}_{1:T}, \psi) \propto \max_\psi \frac{1}{N}\sum_{i=1}^N r_\psi(\tau_i) - \log Z \tag{11.2.2}$$

where Z is the **partition function** needed to normalize of probability with respect to τ.

11.2.1 Inverse RL Partition Function

In our maximum likelihood training, to make the probability with respect to τ sum to 1, we introduced the IRL partition function Z. Mathematically, Z is the integral of all possible trajectories:

$$Z = \int p(\tau) \exp(r_\psi(\tau)) d\tau$$

Then we take the gradient of the likelihood with respect to ψ after plugging in Z:

$$\begin{aligned} \nabla_\psi \mathcal{L} &= \frac{1}{N}\sum_{i=1}^N \nabla_\psi r_\psi(\tau_i) - \frac{1}{Z}\int p(\tau)\exp(r_\psi(\tau))\nabla_\psi r_\psi(\tau) d\tau \\ &\approx \mathbb{E}_{\tau\sim\pi^*(\tau)}[\nabla_\psi r_\psi(\tau_i)] - \mathbb{E}_{\tau\sim p(\tau|\mathcal{O}_{1:T},\psi)}[\nabla_\psi r_\psi(\tau)] \end{aligned} \tag{11.2.3}$$

The first expectation is estimated with **expert samples**, and the second expectation is estimated under the **soft optimal policy under current reward**. To increase the gradient, we want more expert trajectory and less current agent trajectory.

11.2.2 Estimating the Expectation

In the above derivation, the first expectation is easy to calculate, but the second one is hard. To calculate the second expectation, we need to do some "messaging":

$$\begin{aligned} \mathbb{E}_{\tau\sim p(\tau|\mathcal{O}_{1:T},\psi)}[\nabla_\psi r_\psi(\tau)] &= \mathbb{E}_{\tau\sim p(\tau|\mathcal{O}_{1:T},\psi)}\left[\nabla_\psi \sum_{t=1}^T r_\psi(s_t, a_t)\right] \\ &= \sum_{t=1}^T \mathbb{E}_{(s_t,a_t)\sim p(s_t,a_t|\mathcal{O}_{1:T},\psi)}[\nabla_\psi r_\psi(s_t, a_t)]. \end{aligned} \tag{11.2.4}$$

Note that the distribution $p(s_t, a_t|\mathcal{O}_{1:T}, \psi)$ can be rewritten using chain rule as:

$$p(s_t, a_t|\mathcal{O}_{1:T}, \psi) = p(a_t|s_t, \mathcal{O}_{1:T}, \psi)p(s_t|\mathcal{O}_{1:T}, \psi), \tag{11.2.5}$$

where

$$\begin{aligned} p(a_t|s_t, \mathcal{O}_{1:T}, \psi) &= \frac{\beta(s_t, a_t)}{\beta(s_t)} \\ p(s_t|\mathcal{O}_{1:T}, \psi) &\propto \alpha(s_t)\beta(s_t). \end{aligned}$$

Therefore, the distribution is directly proportional to the product of the backward message $\beta(s_t, a_t)$ and the forward message $\alpha(s_t)$:

$$p(a_t|s_t, \mathcal{O}_{1:T}, \psi)p(s_t|\mathcal{O}_{1:T}, \psi) \propto \beta(s_t, a_t)\alpha(s_t). \tag{11.2.6}$$

If we let $\mu_t(s_t, a_t) \propto \beta(s_t, a_t)\alpha(s_t)$, then the second expectation can be written as:

$$\begin{aligned}\mathbb{E}_{\tau\sim p(\tau|\mathcal{O}_{1:T},\psi)}[\nabla_\psi r_\psi(\tau)] &= \sum_{t=1}^{T} \int\int \mu_t(s_t, a_t)\nabla_\psi r_\psi(s_t, a_t) ds_t da_t \\ &= \sum_{t=1}^{T} \vec{\mu}_t^\top \nabla_\psi \vec{r}_\psi\end{aligned} \tag{11.2.7}$$

where $\vec{\mu}_t$ is the state-action visitation probability for each state-action tuple (s_t, a_t).

Remark. This is only feasible for small and discrete state and action spaces. Also, we need to know the dynamics to compute the forward and backward messages.

Now we are ready to sketch out our MaxEnt Inverse RL algorithm in Alg. 37 (Ziebart et al., 2008). We can use this to learn the reward function.

Algorithm 37 MaxEnt Inverse RL

Require: Some random reward parameter ψ

1: **while** True **do**
2: Given ψ, compute backward message $\beta(s_t, a_t)$
3: Given ψ, compute forward message $\alpha(s_t)$
4: Compute $\mu_t(s_t, a_t) \propto \beta(s_t, a_t)\alpha(s_t)$
5: Evaluate $\nabla_\psi \mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\psi r_\psi(s_{i,t}, a_{i,t}) - \sum_{t=1}^{T}\int\int \mu_t(s_t, a_t)\nabla_\psi r_\psi(\tau) ds_t da_t$
6: $\psi \leftarrow \psi + \eta\nabla_\psi \mathcal{L}$

Why is it called maximum entropy (MaxEnt)? Because in cases where $r_\psi(s_t, a_t) = \psi^\top \mathbf{f}(s_t, a_t)$, we can show that Alg. 37 oprimizes

$$\max_\psi \mathcal{H}(\pi^{r_\psi}) \text{ s.t. } \mathbb{E}_{\pi^{r_\psi}}[\mathbf{f}] = \mathbb{E}_{\pi^*}[\mathbf{f}].$$

Insight. The MaxEnt algorithm makes the reward function as diverse as possible, which says that you should not make any undue assumptions about the reward function that are not supported by the data.

11.3 Approximations in High Dimensions

There is some drawbacks for the MaxEnt algorithm. So far, MaxEnt Inverse RL requires

- Solving for soft optimal policy in the inner loop
- Enumerating all state-action tuples for visitation frequency and gradient

Besides, to apply this in practical problem settings, we need to handle

- Large and continuous state and action spaces
- States obtained via sampling only
- Unknown dynamics

Recall the gradient of likelihood is calculated as

$$\nabla_\psi \mathcal{L} = \mathbb{E}_{\tau\sim\pi^*(\tau)}[\nabla_\psi r_\psi(\tau)] - \mathbb{E}_{\tau\sim p(\tau|\mathcal{O}_{1:T},\psi)}[\nabla_\psi r_\psi(\tau)]$$

We know that the first expectation is easy to calculate by sampling expert data, but the second expectation is hard to calculate. One idea to calculate it is to **learn** the entire soft optimal policy $p(a_t|s_t, \mathcal{O}_{1:T}, \psi)$ using any max-ent RL algorithm and then run this policy to sample $\{\tau_j\}$ such that:

$$\nabla_\psi \mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \nabla_\psi r_\psi(\tau_i) - \frac{1}{M}\sum_{j=1}^{M} \nabla_\psi r_\psi(\tau_j) \tag{11.3.1}$$

where we estimate the second expectation using the current policy samples. To learn the entire soft optimal policy $p(a_t|s_t, \mathcal{O}_{1:T}, \psi)$, we have to optimize the following objective

$$J(\theta) = \sum_t E_{\pi(s_t,a_t)}\left[r_\psi\left(s_t, a_t\right)\right] + E_{\pi(s_t)}\left[\mathcal{H}\left(\pi\left(a|s_t\right)\right)\right] \tag{11.3.2}$$

which is highly impractical because this requires us to run an RL algorithm to convergence in every gradient step.

11.3.1 More Efficient Updates

Instead of learning the policy at each time step, we could improve the policy a little in each time step such that if the policy keeps getting better, we can generate good samples eventually. Now sampling from this improved distribution is not actually sampling from the distribution we want, which is $p(\tau|\mathcal{O}_{1:T}, \psi)$. Actually, we are getting a *biased* estimation of the distribution. Therefore, to resolve this issue, we use **importance sampling**:

$$\begin{aligned} \nabla_\psi \mathcal{L} &\approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\psi r_\psi(\tau_i) - \frac{1}{\sum_j w_j}\sum_{j=1}^{M} w_j \nabla_\psi r_\psi(\tau_j) \\ w_j &= \frac{p(\tau)\exp(r_\psi(\tau_j))}{\pi(\tau_j)}. \end{aligned} \tag{11.3.3}$$

And if we take a closer look at the importance ratio w_j:

$$\begin{aligned} w_j &= \frac{p(\tau)\exp(r_\psi(\tau_j))}{\pi(\tau_j)} \\ &= \frac{p(s_1)\prod_t p(s_{t+1}|s_t,a_t)\exp(r_\psi(s_t,a_t))}{p(s_1)\prod_t p(s_{t+1}|s_t,a_t)\pi(a_t|s_t)} \\ &= \frac{\exp(\sum_t r_\psi(s_t,a_t))}{\prod_t \pi(a_t|s_t)} \end{aligned}$$

With the importance ratio, each policy update with respect to r_ψ brings us closer to the target distribution.

11.4 Inverse RL as a Generative Adversarial Network

The idea of inverse RL looks like a game. Specifically, we have an initial policy π_θ, and expert demonstrations π^*. We sample trajectories τ_j from the initial policy, and τ_i from the expert policy. Then our gradient step looks like:

$$\nabla_\psi \mathcal{L} \simeq \frac{1}{N}\sum_{i=1}^{N} \nabla_\psi r_\psi(\tau_i) - \frac{1}{\sum_j w_j}\sum_{j=1}^{M} w_j \nabla_\psi r_\psi(\tau_j) \tag{11.4.1}$$

where demos are made more likely and samples are made less likely. Then we update the initial policy π_θ with respect to r_ψ:

$$\nabla_\theta \mathcal{L} \approx \frac{1}{M}\sum_{j=1}^{M} \nabla_\theta \log \pi_\theta(\tau_j) r_\psi(\tau_j) \tag{11.4.2}$$

which in turn changes the policy to make it harder to distinguish from demos.

This looks a lot like a Generative Adversarial Network (GAN). In a GAN, we have a generator that takes in some noise z, and outputs a distribution $p_\theta(x|z)$. We sample from the generator distribution $p_\theta(x)$. There is also demonstration data, for example, the real images, which we sample from its distribution $p^*(x)$. There is a discriminator parameterized by ψ that determines if the data is real: $D(x) = p_\psi(\text{real}|x)$. We update the discriminator parameter by maximizing the binary log likelihood:

$$\psi = \arg\max_\psi \frac{1}{N} \sum_{x \sim p^*} \log D_\psi(x) + \frac{1}{M} \sum_{x \sim p_\theta} \log(1 - D_\psi(x)) \tag{11.4.3}$$

where the log likelihood of the data is from demonstration is maximized and that of the data is from generator is minimized. We also update the generator parameter θ:

$$\theta \leftarrow \arg\max_\theta \mathbb{E}_{x \sim p_\theta} \log D_\psi(x) \tag{11.4.4}$$

so as to make it harder to distinguish from demos. Therefore, interestingly, we can frame the IRL problem as a GAN. In a GAN, the optimal discriminator can be showed to be:

$$D^*(x) = \frac{p^*(x)}{p_\theta(x) + p^*(x)} \tag{11.4.5}$$

For inverse RL, the optimal policy approaches $\pi_\theta(\tau) \propto p(\tau) \exp(r_\psi(\tau))$. Choosing the above optimal parameterization of the discriminator:

$$\begin{aligned} D_\psi(\tau) &= \frac{p(\tau) \frac{1}{Z} \exp(r(\tau))}{p_\theta(\tau) + p(\tau) \frac{1}{Z} \exp(r(\tau))} \\ &= \frac{p(\tau) \frac{1}{Z} \exp(r(\tau))}{p(\tau) \prod_t \pi_\theta(a_t|s_t) + p(\tau) \frac{1}{Z} \exp(r(\tau))} \\ &= \frac{\frac{1}{Z} \exp(r(\tau))}{\prod_t \pi_\theta(a_t|s_t) + \frac{1}{Z} \exp(r(\tau))} \end{aligned} \tag{11.4.6}$$

then we optimize the discriminator with respect to ψ such that:

$$\psi = \arg\max_\psi \mathbb{E}_{\tau \sim p^*}[\log D_\psi(\tau)] + \mathbb{E}_{\tau \sim \pi_\theta}[\log(1 - D_\psi(\tau))]$$

Now we don't need the importance ratio anymore, because it is subsumed into Z.

Can we just use a regular discriminator?

What if we don't want to recover the reward function? We could also use a general discriminator, i.e., D_ψ is just a normal binary neural net classifier. It is often simpler to set up optimization because we have fewer moving parts. However, the discriminator knows nothing at convergence, and generally the reward cannot be re-optimized.

Figure 11.1: IRL as adversarial optimization.

11.5 Further readings

For classic papers, Abbeel and Ng (2004) is a good introduction to inversed reinforcement learning. Ziebart et al. (2008) is an introduction to probabilistic method for inverse reinforcement learning. For modern papers, Finn et al. (2016) and Wulfmeier et al. (2015) talked about sampling-based method for MaxEnt IRL that handles unknown dynamics and deep reward functions. Ho and Ermon (2016) talked about IRL using GANs. Ho and Ermon (2016) further proposed learning robust reward functions with adversarial IRL.