Statistical Techniques in Robotics (16-831, S21) Lecture #24 (Wednesday, May 5)

MaxEnt IRL and GAIL

Lecturer: Kris Kitani *Scribes: Samuel Triest, Ingrid Navarro*

1 Review

This section briefly surveys the topics covered during the previous lecture. These include: Matrix Game IRL, Maximum Margin IRL, and Maximum Margin Planning.

1.1 Matrix Game IRL

In the previous lecture we derived a game theoretic approach to the IRL problem following [1]. In this paradigm, we used the policy value shown below to derive our objective:

$$V(\pi) = \theta \cdot \mu(\pi), \tag{1}$$

We discussed that the Matrix Game IRL wants to find the value of the expert's policy π_E that is better than any other policy, i.e.,

$$V(\pi_E) \geq V(\pi) \tag{2}$$
$$\theta \cdot \mu(\pi_E) \geq \theta \cdot \mu(\pi) \tag{3}$$

In fact, we want to have the largest possible difference between these two terms. We, thus, wrote this objective function as a minimum over the whole function with a maximum expected value of the next best policy

$$\min_{\theta}\{\max_{\pi} \theta \cdot \mu(\pi) - \theta \cdot \mu(\pi_E)\}. \tag{4}$$

By further re-arranging the terms of this objective, we revealed the notation of the two-player game (5) where $G \in \mathbb{R}^{N \times M}$ represents the game, $\theta \in \mathbb{R}^{1 \times N}$ is the row player, and $\psi \in \mathbb{R}^{M \times 1}$ is a column player. Here, N is the number of features and M is the number of deterministic policies:

$$\min_{\theta}\{\max_{\pi} \theta^{\top} G \psi\} \tag{5}$$

1.2 Maximum Margin IRL

We also surveyed the Max-Margin IRL algorithm introduced in [2]. Max-Margin IRL is formulated as a quadratic programming problem where we are not guaranteed to recover a true reward function but we will find a policy that performs as well as the expert.

In Max-Margin IRL, a reward function is not available but we assume that it can be expressed as linear combination of known "features". Thus, in order to recover the unknown reward, the agent observes expert behavior. From this, we set up the objective for this algorithm as a maximization

problem where we want to find a value function for which the expert policy does better than any other policy by a margin t:

$$\begin{aligned} &\max_{\theta} \ t \\ &\text{s.t. } \theta^T \mu(\pi^*) \geq \theta^T \mu(\pi) + t \quad \forall \pi \end{aligned} \tag{6}$$

We further refined this objective by adding a regularization term to bound the parameters, θ. Then, by using Reinforcement Learning (RL) we can find n valid policies that will serve to compare against the expert policy. Finally, we re-wrote the objective as the minimization problem shown below:

$$\begin{aligned} \min_{\theta,t} \quad & \lambda ||\theta||_2 - t \\ \text{s.t.} \quad & \theta^T(\mu(\pi^*) - \mu(\pi)) \geq t \quad \forall \pi_n \in \Pi \end{aligned} \tag{7}$$

1.3 Maximum Margin Planning

Finally, we also discussed Structured Output Max-Margin IRL, also known as Max-Margin Planning (MMP)[3], an algorithm that learns to plan by solving a Structured Maximum Margin prediction problem over a space of policies. MMP in contrast to Max-Margin IRL allows demonstration of policies from more than a single MDP, i.e., it allows demonstrations from multiple feature maps with different start and goal states.

First, we observed that the minimization objective derived in (7) sets a margin of 1. Instead, MMP uses the notion of a variable margin, $l(\pi^*, \pi)$, that depends on the distance between policies.

$$\begin{aligned} \min_{\theta,\xi} \quad & \lambda ||\theta||_2 + \sum_i \xi_i \\ \text{s.t.} \quad & \theta^T(\mu(\pi^*) - \mu(\pi)) \geq l(\pi^*, \pi_i) - \xi_i \quad \forall i \end{aligned} \tag{8}$$

The intuition behind this paradigm is that if two policies are significantly different, then their margin should large. In particular, we discussed the structural loss $l(\pi^*, \pi_d) = l_d^T \eta^*$ with $\eta(a, s) \in \mathbb{R}^{|S||A|}$ as an occupancy measure that counts the number of times a state-action pair has been visited.

For this algorithm, an alternative representation of the policy value is used:

$$V(\pi) = \theta^T F \eta_\pi \tag{9}$$

where $F \in \mathbb{R}^{N \times |S||A|}$ is a feature map for each state-action pair, and $\mu = F\eta$ is the expected feature count. Using this representation, we expressed the final objective for Max-Margin planning:

$$\min_{\theta} \frac{1}{2}||\theta||_2^2 + \frac{\lambda}{D} \sum_d \beta_d \big\{ \max_{\eta} (\theta^T F_d + l_d^T)\eta - \theta^T F_d \eta_d \big\} \tag{10}$$

2 Summary

2.1 Maximum Entropy Inverse Reinforcement Learning (MaxEnt IRL)

2.1.1 Prerequisite: Entropy (of a distribution)

As one might expect from the title of this section, the notion of entropy is central to the construction of the MaxEnt IRL algorithm [4]. The entropy $\mathcal{H}$ of a distribution $p(X)$ is defined as follows:

$$\mathcal{H}(p(X)) = \sum_{x \in sup(p(X))} [-p(x)log(p(x))] \tag{11}$$

Alternatively, entropy can be expressed in a continuous setting:

$$\mathcal{H}(p(X)) = \int_x -p(x)log(p(x))dx \tag{12}$$

Intuitively, entropy measures the "randomness" of a distribution, with the uniform distribution over a support having maximum possible entropy, and a distribution with all mass at a single outcome having minimum possible entropy.

2.1.2 Problem Setup

As mentioned in [5], the unconstrained IRL problem has many solutions for a given expert policy π_E. LP-IRL addresses this by introducing both an L_1 regularizer on reward weights and a constraint on the reward function.

MaxEnt IRL instead addresses IRL solution ambiguity via an entropic regularizer. The general idea of this regularizer follows from the principle of maximum entropy, which states that given a set of distributions that explain a set of data equally well, the best choice is the one that has the highest entropy.

The MaxEnt IRL setup assumes that we are given a set of expert demonstrations, or a set of trajectories $\zeta_E^{(1)} \dots \zeta_E^{(N)}$ sampled from some underlying expert trajectory distribution $\tilde{p}(\zeta)$ (note that given an MDP, a trajectory distribution can be induced solely by a policy π). The objective of MaxEnt IRL is thus to find a distribution $p(\zeta)$ that closely resembles $\tilde{p}(\zeta)$ while obeying the maximum entropy principle. The mathematical framework for how this optimization occurs will be presented in the following sections.

2.1.3 The Objective

It is straightforward to enforce that our decision variable $p(\zeta)$ follow the principle of maximum entropy:

$$\max_{p(\zeta)}[\mathcal{H}(p(\zeta))] = \min_{p(\zeta)}[-\mathcal{H}(p(\zeta))] \tag{13}$$

$$\max_{p(\zeta)}[\mathcal{H}(p(\zeta))] = \min_{p(\zeta)}[-\sum_{\zeta} p(\zeta)(-log(p(\zeta)))] \tag{14}$$

$$\max_{p(\zeta)}[\mathcal{H}(p(\zeta))] = \min_{p(\zeta)}[\sum_{\zeta} p(\zeta)log(p(\zeta))] \tag{15}$$

2.1.4 The Constraints

The MaxEnt IRL formulates "closeness" to $\tilde{p}(\zeta)$ as matching expected feature counts, and treats it as an optimization constraint. This constraint has the following form:

$$\sum_{\zeta} p(\zeta)\mu(\zeta) = \sum_{d} p(\zeta_d)\mu(\zeta_d) \tag{16}$$

Finally, we also constrain $p(\zeta)$ to be a valid distribution, i.e.:

$$\sum_{\zeta} p(\zeta) = 1 \tag{17}$$

2.1.5 The MaxEnt IRL Optimization

The full MaxEnt IRL optimization is the following (constraints set to 0 for the next step):

$$\begin{aligned} \min_{p(\zeta)} \quad & \sum_{\zeta} p(\zeta)log(p(\zeta)) \\ \text{s.t.} \quad & \sum_{d}[p(\zeta_d)\mu(\zeta_d)] - \sum_{\zeta}[p(\zeta)\mu(\zeta)] = 0 \\ & \sum_{\zeta}[p(\zeta)] - 1 = 0 \end{aligned} \tag{18}$$

We cannot evaluate the objective and constraints directly as we cannot enumerate all of trajectory space. However, we can apply techniques from constrained optimization in order to generate an update rule.

2.1.6 Aside: Lagrangians and Constrained Optimization

Consider a generic constrained minimization:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned} \tag{19}$$

where $f(x), c(x)$ are smooth (i.e. have derivatives). It is well-known that any solution to this problem must have the following property:

$$\nabla f(x) = \lambda \nabla c(x) \tag{20}$$

This essentially states that the gradient of the objective must be parallel to the gradient of the constraint manifold. Were this not true, there would exist a direction along the constraint manifold

(locally, the hyperplane with $\nabla c(x)$ as its normal) that would result in a decrease in $f(x)$. This property is commonly expressed via a function known as the Lagrangian:

$$L(x, \lambda) = f(x) + \lambda^T c(x) \tag{21}$$

Due to the above first-order necessary condition, the Lagrangian has the following property at an optimum:

$$\nabla L(x, \lambda) = 0 \tag{22}$$

2.1.7 Solving the Constrained Optimization

We can use the first-order necessary condition to get an insight into the MaxEnt IRL optimization:

$$L(p(\zeta), \theta, V) = \sum_{\zeta}[p(\zeta)log(p(\zeta))] + \theta^T(\sum_{d}[p(\zeta_d)\mu(\zeta_d)] - \sum_{\zeta}[p(\zeta)\mu(\zeta)]) + V\sum_{\zeta}[p(\zeta)] - 1 \tag{23}$$

We then take its gradient w.r.t. $p(\zeta)$:

$$\nabla_{p(\zeta)} L(p(\zeta), \theta, V) = log(p(\zeta)) + 1 + V - \theta^T \mu(\zeta) \tag{24}$$

This is 0 at an optimum:

$$0 = log(p(\zeta)) + 1 + V - \theta^T \mu(\zeta) \tag{25}$$

Solving for $p(\zeta)$:

$$p(\zeta) = \frac{exp(\theta^T \mu(\zeta))}{exp(V + 1)} \tag{26}$$

We can draw two insights from equation 25:

1. The optimal distribution $p(\zeta)$ is a Boltzmann distribution with energy [1] $(\theta^T \mu(\zeta))$ and partition function $Z = exp(V + 1)$. This follows from [6].
2. The Lagrange multiplier θ^T can be interpreted as a reward function that is linear in features. (Recall that reward is linear in features iff. value is linear in feature counts).

2.1.8 Deriving the MaxEnt IRL Update

Using the insights from the previous section, we now know that the optimal distribution family is Boltzmann. From this, [6] show that we solve this optimization by maximizing the likelihood of the expert demonstrations (we can use θ as out decision variable, as $p(\zeta)$ is parameterized by θ):

$$L(\theta) = \theta^T \sum_{d}[p(\zeta_d)\mu(\zeta_d)] - log(\sum_{\zeta}[exp(\theta^T \mu(\zeta)])) \tag{27}$$

Computing the gradient of $L(\theta)$ gives us the following:

[1] energy is essentially an unnormalized probability

$$\nabla_\theta L(\theta) = \sum_d [p(\zeta_d)\mu(\zeta_d)] - \nabla_\theta log(\sum_\zeta [exp(\theta^T \mu(\zeta)]) \tag{28}$$

$$\nabla_\theta L(\theta) = \sum_d [p(\zeta_d)\mu(\zeta_d)] - \frac{1}{\sum_\zeta [exp(\theta^T \mu(\zeta)]} (\sum_\zeta [\nabla_\theta exp(\theta^T \mu(\zeta)]) \tag{29}$$

$$\nabla_\theta L(\theta) = \sum_d [p(\zeta_d)\mu(\zeta_d)] - \frac{1}{\sum_\zeta [exp(\theta^T \mu(\zeta)]} (\sum_\zeta [\mu(\zeta) exp(\theta^T \mu(\zeta)]) \tag{30}$$

$$\nabla_\theta L(\theta) = \sum_d [p(\zeta_d)\mu(\zeta_d)] - \sum_\zeta \mu(\zeta) \frac{[exp(\theta^T \mu(\zeta)]}{\sum_\zeta [exp(\theta^T \mu(\zeta)]} \tag{31}$$

$$\nabla_\theta L(\theta) = \sum_d [p(\zeta_d)\mu(\zeta_d)] - \sum_\zeta \mu(\zeta) p(\zeta) \tag{32}$$

$$\nabla_\theta L(\theta) = \mu_d - \bar{\mu} \tag{33}$$

Where μ_d and $\bar{\mu}$ are the expected feature counts of the expert and optimal policy induced by θ respectively.

2.1.9 The MaxEnt IRL Algorithm

As implied by the last section, given a reward function θ, we need to run RL in order to get the expected feature counts $\bar{\mu}$. A practical algorithm for MaxEnt IRL is presented in Algorithm 1.

Algorithm 1 MaxEnt IRL $(K, \eta, \zeta_d, \phi(\cdot), \mathcal{E})$

$\mu_d = \frac{1}{D} \sum_d \sum_t \gamma^t \phi(s_t^{(d)})$ ▷ Precompute expected feature counts for expert
for $i \in 1 \ddots N$ **do**
  $\pi = \text{SoftValueIteration}(\mathcal{E}, \theta)$ ▷ Compute the optimal policy in $\mathcal{E}$ for reward θ
  $\mu = \frac{1}{K} \sum_k \sum_t \gamma^t \phi(s_t^{(k)})$ ▷ Compute expected feature counts of π via sample rollouts.
  $\theta = \theta - \eta(\mu_d - \bar{\mu})$ ▷ Gradient update
end for

2.2 Generative Adversarial Imitation Learning (GAIL)

2.2.1 Prerequisite: Generative Adversarial Networks (GAN)

This section provides a brief overview of Generative Adversarial Networks (GANs) [7] and serves as an introduction to the notation and key concepts required in Section 2.2.

Problem Setup. Figure 1 outlines a basic GAN framework. It consists of a *generator*, $G(z)$, which given some prior input noise $z \sim q(z)$ produces data points x^{fake}. The framework further defines a *discriminator*, $D(x)$, which given a point x outputs the probability that x came from the true data rather than the generator. As such, the goal of a GAN is to learn a model G able to generate fake data points that are indistinguishable from the true data.

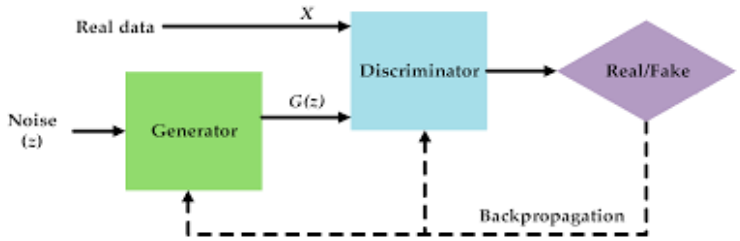

Figure 1: GAN Framework

Discriminator Formulation. Assume there exists a generator G as defined above. Let us assume we have a dataset $\mathcal{D}^* = \{x_1, ..., x_N\}$ which contains true data points, and a dataset $\mathcal{D} = \{x_1^{'}, ..., x_M^{'}\}$ containing generated data. We can define a distance function $D_\phi(x) \in [0, 1]$, to score whether a data point produced by the generator is similar to a data point from the true data.

Now, since we have $\mathcal{D}^*$ and $\mathcal{D}$, we can learn such distance function by setting the following objective:

$$\max_{\phi} \Big\{ \sum_{x \sim \mathcal{D}^*} [D_\phi(x)] + \sum_{x \sim \mathcal{D}} [1 - D_\phi(x)] \Big\} \tag{34}$$

i.e., we want to maximize the distance function parametrized by ϕ such that the difference between discriminating against a true data point and a fake data point is as large as possible.

Note that we can also express the objective above as an expectation over samples (35) and using base distributions p and q (36):

$$\max_{\phi} \Big\{ \mathbb{E}_{D^*}[D_\phi(x)] + \mathbb{E}_{\mathcal{D}}[1 - D_\phi(x)] \Big\} \tag{35}$$

$$\max_{\phi} \Big\{ \sum_{x \sim p(x)} [D_\phi(x)] + \sum_{z \sim q(z)} [1 - D_\phi(G_\theta(z))] \Big\} \tag{36}$$

Generator Formulation. Now, assume there exists a discriminator D_ϕ as formulated in the previous section. Our goal is to define a generator G_θ which is able to trick the discriminator. To do so, we want to learn its parameters θ following the objective below:

$$\min_{\theta} \Big\{ \sum_{z \sim p(z)} [1 - D_\phi(G_\theta(z))] \Big\} \tag{37}$$

GAN Formulation. The GAN formulation combines both the generator objective and the discriminator objective together to learn both G_θ and D_ϕ simultaneously. It does so by setting the

following min-max game:

$$\min_{\theta} \max_{\phi} \Big\{ \sum_{x \sim \mathcal{D}^*} [D_\phi(x)] + \sum_{x \sim \mathcal{D}} [1 - D_\phi(x)] \Big\} \tag{38}$$

$$\min_{\theta} \max_{\phi} \Big\{ \sum_{x \sim \mathcal{D}^*} [\ln\{D_\phi(x)\}] + \sum_{x \sim \mathcal{D}} [\ln\{1 - D_\phi(x)\}] \Big\} \quad \text{"cross-entropy" formulation} \tag{39}$$

If the distance function is differentiable, we can optimize for its parameters using gradient ascent with the following update equation:

$$\phi = \phi + \mathbb{E}_{\mathcal{D}^*}[\nabla_\phi \ln\{D_\phi(x)\}] + \mathbb{E}_{\mathcal{D}}[\nabla_\phi \ln\{1 - D_\phi(x)\}] \tag{40}$$

Similarly, we can define the generator as a differentiable function and optimize it using gradient descent:

$$\theta = \theta - \mathbb{E}_{\mathcal{D}}[\nabla_\theta \ln\{1 - D_\phi(x)\}] \tag{41}$$

GAN Algorithm. Finally, the GAN algorithm is shown below:

Algorithm 2 GAN

function GAN
  $\mathcal{D}^* = \{x_m\}_{m=1}^M \sim p^*(x)$ ▷ Obtain true data points from the distribution $p^*(x)$
  for $k = 1, \dots K$ **do**
    $\mathcal{D}_\theta = \{x_n\}_{n=1}^N \sim G_\theta(z)|q(z)$ ▷ Obtain data points from the distribution $q(z)$
    $\phi = \phi + \mathbb{E}_{\mathcal{D}^*}[\nabla_\phi \ln\{D_\phi(x)\}] + \mathbb{E}_{\mathcal{D}_\theta}[\nabla_\phi \ln\{1 - D_\phi(x)\}]$ ▷ Discriminator update
    $\theta = \theta - \mathbb{E}_{\mathcal{D}_\theta}[\nabla_\theta \ln\{1 - D_\phi(x)\}]$ ▷ Generator update
  end for
return $\boldsymbol{\theta}$
end function

2.3 Generative Adversarial Imitation Learning

2.3.1 Problem Setup

GAIL [8] proposes a general framework for extracting a policy directly from data, as if it was obtained using RL followed by IRL. GAIL is formulated as a model-free imitation learning (IL) algorithm that exploits generative adversarial training to fit distributions of state-action pairs that define expert behavior.

The GAIL framework defines a policy that is trying to generate trajectories similar to those produced by an expert. Then, it defines a scoring function which attempts to detect the generated trajectories.

2.3.2 Discriminator Formulation

To formulate the GAIL discriminator, assume we are given a dataset containing expert trajectories $\mathcal{D}^* = \{s_1, a_1, \dots, s_N, a_N\}$, as well as fake trajectories $\mathcal{D}_\theta = \{s_1, a_1, \dots, s_M, a_M\}$.

We want to learn a discriminator, $D_\phi(s, a) \in [0, 1]$, that is able to distinguish between a true state-action pair and generated one. This discriminator can framed as a cost function, where expert action induce low cost whereas fake actions induce a higher cost.

Now, based on the objective derived in (34) we can formulate a slightly different optimization problem:

$$\max_{\phi} \Big\{ \sum_{s,a \sim \mathcal{D}_\theta} [D_\phi(s, a)] + \sum_{s,a \sim \mathcal{D}^*} [1 - D_\phi(s, a)] \Big\} \tag{42}$$

where instead of looking at a data point x as before, we are looking at a state-action pair.

2.3.3 Generator Formulation

The GAIL framework proposes a neural-based policy $G_\theta(s)$ that produces an action given a state. Note that this closely resembles the formulation of a generator network, except that instead of using noise as the input to the network, it uses a state.

To generate a full trajectory, the policy is used auto-regressively along with an environment denoted by $\mathcal{E}$. The policy receives a state s_t from the environment and uses it to produce an action a_t, s.t. $a_t = G_\theta(s_t)$. Then, the state-action pair is used as an input to the environment to produce the next state, $\mathcal{E}(s_t, a_t) \to s_{t+1}$. The new state becomes the current state of the policy and the process is repeated, thus, generating a trajectory of state-action pairs.

Now, suppose we are given a cost function (discriminator) as formulated above. Thus, following the optimization problem in (37), the GAIL objective for the generator can be expressed as:

$$\min_{\theta} \Big\{ \sum_{s,a \sim \mathcal{D}_\theta} [D_\phi(s, a)] \Big\} \tag{43}$$

2.3.4 GAIL Formulation

As derived in Section 2.2.1, we can combine the discriminator and generator objectives together to derive the GAIL min-max objective:

$$\min_{\theta} \max_{\phi} \Big\{ \sum_{s,a \sim \mathcal{D}_\theta} [\ln\{D_\phi(s, a)\}] + \sum_{s,a \sim \mathcal{D}^*} [\ln\{1 - D_\phi(s, a)\}] \Big\} \tag{44}$$

2.3.5 GAIL Algorithm

Finally, the figure below shows a simplified version of the GAIL algorithm.

Algorithm 3 Simplified GAIL

1: **function** GAIL
2: $\mathcal{D}^* = \{\zeta_n^*\}_{m=1}^M$ where $\zeta_m^* = \{s^{(0)}, a^{(0)}, \ldots, s^{(T_m)}\} \sim \mathcal{E}|\pi^*$
3: **for** $k = 1, \ldots K$ **do**
4: $\mathcal{D}_\theta = \{\zeta_n\}_{n=1}^N$ where $\zeta_n = \{s^{(0)}, a^{(0)}, \ldots, s^{(T_n)}\} \sim \mathcal{E}|\pi_\theta$
5: $\phi = \phi + \mathbb{E}_{\mathcal{D}_\theta}[\nabla_\phi \ln D_\phi(a, s)] + \mathbb{E}_{\mathcal{D}^*}[\nabla_\phi \ln\{1 - D_\phi(a, s)\}]$
6: $\theta = \theta - \mathbb{E}_{\mathcal{D}_\theta}[\nabla_\theta \ln \pi_\theta(a|s) Q(a, s)]$
7: **end for**
8: **return** $\boldsymbol{\theta}$
9: **end function**

The algorithm above is similar to Algorithm 2.2.1. The main difference lies in that the generator's optimization step in line 6 uses the policy gradient update which depends on $Q(s, a)$.

The actual implementation in [8] follows the algorithm shown below. This version of the algorithm uses the Adam optimizer [9] to update the discriminator parameters, which is depicted in line 5. Then, in line 6, the generator's update includes an additional term which serves as a regularizer. Furthermore, the algorithm uses Trust Region Policy Optimization (TRPO) [10] to update the corresponding parameters.

Algorithm 4 GAIL

1: **function** GAIL(λ)
2: $\mathcal{D}^* = \{\zeta_n^*\}_{m=1}^M$ where $\zeta_m^* = \{s^{(0)}, a^{(0)}, \ldots, s^{(T_m)}\} \sim \mathcal{E}|\pi^*$
3: **for** $k = 1, \ldots K$ **do**
4: $\mathcal{D}_\theta = \{\zeta_n\}_{n=1}^N$ where $\zeta_n = \{s^{(0)}, a^{(0)}, \ldots, s^{(T_n)}\} \sim \mathcal{E}|\pi_\theta$
5: $\phi = \text{ADAM}(\mathbb{E}_{\mathcal{D}_\theta}[\nabla_\phi \ln D_\phi(a, s)] + \mathbb{E}_{\mathcal{D}^*}[\nabla_\phi \ln\{1 - D_\phi(a, s)\}])$
6: $\theta = \text{TRPO}(\mathbb{E}_{\mathcal{D}_\theta}[\nabla_\theta \ln \pi_\theta(a|s) Q(a, s) - \lambda \nabla_\theta H(\pi_\theta)])$
7: **end for**
8: **return** $\boldsymbol{\theta}$
9: **end function**

2.3.6 Theoretical Contributions

Three main contributions that the GAIL paper made are:

1. Unifies IRL with the following objective:

$$IRL_\psi(\pi^*) = \max_c \min_\pi (-H(\pi) + \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t c(s, a)]) - \mathbb{E}_{\pi^*}[\sum_{t=0}^{\infty} \gamma^t c(s, a)] - \psi(c), \tag{45}$$

   where the terms represent the maximum causal entropy, the expert policy cost and the regularizer, respectively.

2. Shows that the choice of regularizer leads to different types of IRL

3. GAIL's regularizer is given by:

$$\psi_{GAIL}(c) = \begin{cases} \mathbb{E}_{\pi^*}[-c(s, a) - \ln\{1 - e^{c(s,a)}\}] & \text{if } c(s, a) \leq 0 \\ +\infty & \text{otherwise} \end{cases} \tag{46}$$

References

[1] Umar Syed and Robert E. Schapire. A game-theoretic approach to apprenticeship learning. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1449–1456. Curran Associates, Inc., 2007.

[2] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In Carla E. Brodley, editor, *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, volume 69 of *ACM International Conference Proceeding Series*. ACM, 2004.

[3] Nathan D. Ratliff, J. Andrew Bagnell, and Martin Zinkevich. Maximum margin planning. In William W. Cohen and Andrew W. Moore, editors, *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, volume 148 of *ACM International Conference Proceeding Series*, pages 729–736. ACM, 2006.

[4] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[5] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[6] Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

[7] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *Commun. ACM*, 63(11):139–144, 2020.

[8] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[10] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.