A Practitioner's Guide to Maximum Causal Entropy Inverse Reinforcement Learning, Starting from Markov Decision Processes

Eric Heim, ETC SEI

February 2019

This guide is meant to describe both the semantics and mechanics of the Maximum Causal Entropy (MaxCausalEnt) Inverse Reinforcement Learning (IRL) algorithm [4]. Throughout the remainder of this document, we provide a measure of formal definition of the algorithm, starting from the basics, adding some intuition as we go. We intentionally skip a large amount of prior, related, and theoretic work that motivates and contextualizes the algorithm. For further reading on these subjects, see the works referenced throughout. Finally, the gray break out boxes are notes meant to provide broader context, and can be skipped without breaking the flow of the guide.

1 Background

Stated most simply, MaxCausalEnt IRL is an algorithm that attempts to solve the Inverse Reinforcement Learning (IRL) problem. In turn, the IRL problem aims to model observed behavior of an agent or agents acting in an environment. Before introducing the MaxCausalEnt IRL algorithm, we begin by first formally defining the IRL problem, starting from the formal model utilized by IRL methods: The Markov Decision Process.

1.1 Markov Decision Processes

IRL models an agent's behavior in an environment as a Markov Decision Process (MDP):

Definition 1 *A Markov Decision Process (MDP) is defined as the tuple* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P_T, R, \rho_0, \mathcal{S}_\phi)$*, where*

- $\mathcal{S}$ *A finite or infinite set of states (sometimes called the* environment*), each denoted as* $S \in \mathcal{S}$
- $\mathcal{A}$ *A finite or infinite set of actions, each denoted as* $A \in \mathcal{A}$
- P_T *A transition probability distribution* $P_T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ *where* $P_T(S_j | A_i, S_i)$ *denotes the probability of an agent being in state* S_j *after taking action* A_i *while in state* S_i*.*
- R *A reward function* $R : \mathcal{S} \times \mathcal{A} \mapsto [R_{min}, R_{max}] \subseteq \mathbb{R}$*, where* $R(S, A)$ *denotes the reward the agent receives from taking action* A *while in state* S*.*
- P_0 *An initial state distribution* $P_0 : \mathcal{S} \mapsto [0, 1]$*, where* $\rho_0(S)$ *indicates the probability of an agent starting in state* S*.*
- $\mathcal{S}_\phi$ *A set of terminal states.*

NOTE: Various works define MDPs in slightly different ways, some of which have a drastic effects on the method. For instance, Bayesian IRL [2] assumes rewards are a function of state alone. Many methods assume a discount factor (commonly denoted as γ_t) that decays future rewards as a function of time. Some don't assume a distribution over initial states and few assume a set of terminal states. For this guide we will use the above definition of an MDP, but note that other definitions exist, some of which are compatible with the methods described herein.

For a running example, let us consider modeling behaviors of ships as they navigate from various starting locations to a single port. Let us represent the state ships can be in as their latitude and longitude, and their actions to be

directions they can travel. For simplicity we assume both the states and actions are discretized (though this need not be the case, for RL, IRL or other methods that utilize MDPs). More specifically, let sthe state space be a 100 by 100 regular grid of positions ($\mathcal{S} = \{0, ..., 100\} \times \{0, ..., 100\}$), and the action space be the four cardinal directions ($\mathcal{A} = \{'N', 'S', 'E', 'W'\}$). We also consider time to be discretized. As such, at a particular time step t an agent (ship in our case) will be in a coordinate in the grid and choose to take one of the four actions.

P_T determines what state an agent will be in time step $t+1$ after taking taking action in a state at time t. For problems with *deterministic dynamics*, there exists a single state S' for each state $S \in \mathcal{S}$ and each action $A \in \mathcal{A}$ such that $P_T(S'|A, S) = 1$ and for all other states $S'' \in \mathcal{S} - S'$, $P_T(S''|A, S) = 0$. In our example, this means that if a ship is in state $(0, 1)$ and takes action $'N'$ it will always end up in the same state, say $(1, 1)$. For *non-deterministic dynamics*, this means that in our example if a ship is in state $(0, 1)$ and takes action $'N'$, it could end up in one of many states, say $(1, 1)$ (went north one space), $(1, 2)$ (went north two spaces), or $(0, 1)$ (stayed in the same space) each with a different probability.

P_0 indicates how probable it is for agents to be in a state at time $t = 0$ (i.e. initial states). In our running example, the starting locations of ships would have some non-zero probability while all other locations would have probability zero. If an agent transitions to a terminal state $S \in \mathcal{S}_\phi$, then it is assumed to have completed it's task and ceases to perform actions. In the ship example, the location of the port is the single terminal state.

R is a signal that indicates which actions and states are *preferred*. For instance, if $R(S, A) > R(S, A')$ then when in state S, action A is preferred over action A'. In our example, if the port is in coordinate $(50, 50)$, it would make sense that $R((49, 50), 'N') > R((49, 50), 'S')$ in that the first action gets the ship to the port, while the second one takes it farther from the port. If $R(S, A) > R(S', A)$ then it is preferable to take action A in state S than it is when in state S'. Again in our example, if the port is in coordinate $(50, 50)$, it would make sense that $R((49, 50), 'N') > R((99, 100), 'N')$ for the same reason as in the last example.

1.2 Reinforcement Learning

In (forward) *Reinforcement Learning* (RL), the goal is to learn a policy $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ that when used to control an agent in a MDP, it maximizes the reward the agent achieves. Here, π defines a probability distribution of actions, given a state (i.e. $\sum_{A \in \mathcal{A}} \pi(A|S) = 1$ for all $S \in \mathcal{S}$). More formally, the goal in RL is to maximize the expected reward of a policy, that is formalized through a *value function*:

Definition 2 *In reinforcement learning, the value (or state-value) function measures the expected reward of starting in state S and following the policy π. In the discrete state and action MDP case it is defined recursively as:*

$$V_\pi(S) = \sum_{A \in \mathcal{A}} \left[\pi(A|S) \left[R(S, A) + \sum_{S' \in \mathcal{S}} P_T(S'|A, S) V_\pi(S') \right] \right] \tag{1}$$

Intuitively, this measures how "good" it is to be in a particular state if you were to proceed to take actions according to policy π. If you assume π and P_T are deterministic (i.e. the outer sum is over one action for which $\pi(A|S)$ is 1, and the inner sum is over a single state, for which P_T is 1), value function is simply the sum of the rewards as the policy proceeds from state S until a terminal state, or in other words, the *cumulative reward* from that state. In the more general case, the value function measures the *expected cumulative reward*.

A related and important concept in RL is the "Q" function:

Definition 3 *In reinforcement learning, the "Q" (cost-to-go, or state-action-value) function measures the expected reward of starting in state S, taking action A and following the policy π. In the discrete state and action MDP case it is defined in terms of the value function:*

$$Q_\pi(S, A) = R(S, A) + \sum_{S' \in \mathcal{S}} P_T(S'|A, S) V_\pi(S') \tag{2}$$

The first term is the immediate reward the agent receives for taking action A in state S, and the second is the expected future rewards for continuing to use the policy (the "cost to go") in the future. It is worth noting that (1) can be written in terms of (2):

$$V_\pi(S) = \sum_{A \in \mathcal{A}} \pi(A|S) Q_\pi(S, A) \tag{3}$$

Together, (1) and (2) create a recurrence relation that ends in a terminal state.

An optimal policy π^* is one that given a state S, gives high probability to actions that maximize $Q_\pi(S, A)$. Stated another way, if we let $\bar{\pi} : \mathcal{S} \mapsto \mathcal{A}$ be a deterministic policy that maps states to actions (e.g. a deterministic policy can be derived from a stochastic one: $\bar{\pi}(S) = \text{argmax}_{A \in \mathcal{A}} \pi(A|S)$), we can define an optimal policy in terms of the Q function:

Definition 4 *The optimal deterministic policy for an MDP with Q function defined in Def. 2, is defined as:*

$$\bar{\pi}^*(S) = \underset{A \in \mathcal{A}}{\text{argmax}}\, Q_\pi(S, A) \tag{4}$$

Building on our example, assume that a ship receives a large positive reward for transitioning into the location of the port, and a small negative reward for any other transition. The optimal policy would be one that spent as few time steps as possible transitioning to states that were not the port (getting negative reward each time), and transitioning to the port state (getting positive reward) with the fewest steps.

Given MDPs with known dynamics, few states, and few actions, the optimal policy can be exactly computed via dynamic programming. However in practice, many times the dynamics of the environment are unknown and performing dynamic programming over a large state and action spaces is infeasible. For these reasons, RL remains an active research topic for which there are many competing methods, each with their own benefits.

1.3 Inverse Reinforcement Learning

Inverse Reinforcement Learing (IRL) assumes we are given a tuple MDP/R, which is simply the MDP defined in Def. 1 WITHOUT a reward function R. Instead we are given a set of observed *behaviors* (also known as *sequences*, or *trajectories*) $\mathcal{B} = \{B_1, ...B_n\}$, where $\mathcal{B} \ni B = ((S_1, A_1), ..., (S_{m_B}, A_{m_B})) \in (\mathcal{S} \times \mathcal{A})^{m_B}$ that are assumed to be samples of policy $\pi^{\mathcal{B}}$. The goal of IRL is to learn the unknown function R that caused an agent or agents to produce those behaviors. IRL is named as such because it effectively solves the inverse problem from RL: Instead of learning a policy from a samples from a reward function, the IRL problem attempts to learn a reward function from samples from a policy.

1.3.1 Feature Matching IRL Algorithms

Many IRL methods (including Maximum Causal Entropy) solve the IRL problem by way of *feature matching*. Let $f : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}^d$ be a given *feature function* over states and actions. For finite state and action MDPs (such as our discretized ship navigation example), we can equivalently assume that we are given a feature vector $\mathbf{f}_{S,A}$ for each state action pair $(S, A) \in \mathcal{S} \times \mathcal{A}$, and will use this notation in the remainder of this guide. Given observed trajectories $\mathcal{B}$, we can compute the *empirical feature counts* $\mathbf{f}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{B \in \mathcal{B}} \mathbf{f}_B$, where $\mathbf{f}_B = \frac{1}{|B|} \sum_{(S,A) \in B} \mathbf{f}_{S,A}$. This single vector can be seen as a representation of the result of running the policy $\pi^{\mathcal{B}}$ for a fixed amount of time. Assume that an IRL method learns a reward $\hat{R}$. Also, assume that an RL algorithm learns a policy $\hat{\pi}$ using MDP/R $\cup \hat{R}$. A feature matching IRL method attempt to ensure the following:

$$\mathbb{E}\left[\mathbf{f}_{S,A}|\hat{\pi}\right] = \mathbf{f}_{\mathcal{B}} \tag{5}$$

Intuitively, this means that the IRL method would learn a reward such that a policy learned from that reward would visit the same states and perform the same actions as in the observed trajectories, in expectation. It is important to note that feature matching implies that an IRL method must not only find a reward function $\hat{R}$, but also a policy $\hat{\pi}$ learned from $\hat{R}$. Because of this, most feature matching IRL methods proceed by repeating the following steps until convergence:

1. Learn a policy $\hat{\pi}$ from the current estimate of the reward $\hat{R}$ using a procedure at least similar to (forward) RL.
2. Update the reward $\hat{R}$ with a loss that measures how well $\hat{\pi}$ matches features.

This general procedure is how MaxCausalEnt IRL learns a reward function (and also a policy as a result). In the following section, we provide more depth into how specifically it accomplishes this.

2 Maximum Causal Entropy Inverse Reinforcement Learning

Note: While the work that first introduced MaxCausalEnt IRL [4] is a good source on the method, Brian Ziebart's dissertation [3] is a more comprehensive report. The dissertation is much more thorough and explicit in it's derivations and notations. As such, we would recommend those seeking a deeper understanding of MaxCausalEnt IRL to read [3], and those wanting to get a higher-level description to read [4].

The task of matching features for IRL does not lead to a well-posed learning algorithm. Indeed, the IRL problem is ill-posed: There exists many reward functions that can explain behavior. What differentiates different feature matching methods is how they handle this ambiguity. MaxCausalEnt IRL does this by using the principle of *maximum entropy*.

In general, the principle of maximum entropy states that a probability distribution that represents the state of knowledge should be the one with the highest entropy (contains least information). For IRL, this means that a policy that mimics observed behaviors will match features for observed states and actions, but should have maximal entropy otherwise. Said more formally, if the state S never appears in a trajectory $B \in \mathcal{B}$, then a learned policy $\hat{\pi}(A|S)$ should be equal for all $A \in \mathcal{A}$. Said yet *another* way, the distribution over actions is *least commited* when there are no observations.

Note It is this point where the narrative of the method gets a little complicated. In order to apply the principle of maximum entropy to IRL problems, the authors assume a *conditionally causal* relationship between states and actions in a trajectory under a policy:

$$P_\pi(B) = P_\pi\left(\mathcal{A}^B || \mathcal{S}^B\right) = \prod_{t=1}^{m_B} \pi\left(A_t^B | \mathcal{S}_{1:t}^B, \mathcal{A}_{1:t-1}^B\right) \tag{6}$$

Here:

- $\mathcal{A}^B$ ($\mathcal{S}^B$) are the actions (states) in B.
- A_t^B is the action at time t in B.
- $\mathcal{A}_{1:t}^B$ ($\mathcal{S}_{1:t}^B$) are the actions (states) in B from time steps 1 through t.

(6) defines the joint distribution of actions in a trajectory conditioned on the states in the trajectory. Specifically, the joint decomposes into a product of the policy evaluated at each action in the trajectory. Note that the policy (probability of an action) is dependent on *all actions and states before it in the trajectory*. Stated more succinctly, current behavior is dependent on all past behavior within a trajectory.

To reinforce the distinction between a causally conditioned model $\pi(A_t|\mathcal{S}_{1:t}, \mathcal{A}_{1:t-1})$ and one with standard conditioning assumptions $\pi(A_t|S_t)$, lets consider the following scenario in the ship vessel modeling example. Consider the case where two type of ships are traversing to a port. One type of ship comes from the east, and the other from the west. On their route to the port, each must go through a small channel (i.e. the same state). After the channel the two types of ships take different routes to the port: the ship from the east goes northeast around an island, and the ship from the goes northwest around it. Assume that an equal number of ships of each kind were observed. A non-causal conditional model that only considers the current state would determine that, when exiting the channel, going northeast or northwest is equally as likely. A causal conditional model would determine that if the ship comes from the east, it will go northeast after exiting the channel; When a ship comes from the west, it would go northwest after the channel. The reason for this is that the non-causal conditional model has no concept of past states or actions, while the causal conditional model is a function of all previous action and states.

HOWEVER, the authors make the argument not to use the full power of causal conditioning for IRL. If they were to do so, they would be making the assumption that the problem being solved is *Non-Markovian* (i.e. dynamics are NOT conditioned on only the previous state and action, and policies are NOT conditioned on only the current state). While this assumption leads to a much stronger model, it is much more complex to solve for, and doesn't match the assumptions of our MDP. As such, the authors inherit the Markov assumption into their

model and utilize the standard conditioning assumption:

$$\pi\left(A_t|\mathcal{S}_{1:t}, \mathcal{A}_{1:t-1}\right) = \pi\left(A_t|S_t\right) \tag{7}$$

So, at the end of the day, their definition of π does not change from the one previously discussed: The probability of an action under a policy is only conditioned on the state the agent is in.

Using the principle of maximum entropy, the authors define an optimization problem that maximizes the *maximum causal entropy* of the policy $\hat{\pi}$ subject to the feature matching constraint in (5). Informally, the reason the maximum entropy is deemed causal is because of the conditional relationship among states S and actions A in $\hat{\pi}$. The idea is that at ensuring that the causal entropy of a policy is maximized makes it least committed, while satisfying the feature matching constraints with respect to a policy ensures that behaviors are modeled correctly in the policy.

The authors go on to show that their optimization problem satisfies Slater's condition and is convex. Thus, by strong duality, solving the dual of their optimization problem will result in a globally optimal solution. By differentiating the Lagrangian and then equating to zero, the authors show that the policy satisfying their maximum causal entropy IRL optimization problem takes the form:

$$\pi_\Theta\left(A_t|S_t\right) = e^{Q^{\text{soft}}_{\pi_\Theta}(A_t,S_t) - V^{\text{soft}}_{\pi_\Theta}(S_t)} \tag{8}$$

$$V^{\text{soft}}_{\pi_\Theta}\left(S_t\right) = \log \sum_{A_t \in \mathcal{A}} e^{Q^{\text{soft}}_{\pi_\Theta}(A_t,S_t)} \tag{9}$$

$$Q^{\text{soft}}_{\pi_\Theta}\left(A_t, S_t\right) = R_\Theta\left(S_t, A_t\right) + \sum_{S' \in \mathcal{S}} P_T\left(S'|A_t, S_t\right) V^{\text{soft}}_{\pi_\Theta}\left(S'\right) \tag{10}$$

$$R_\Theta\left(S, A\right) = \Theta^T \mathbf{f}_{S,A} \tag{11}$$

where, Θ are dual parameters. With this definition of π_Θ, the MaxCausalEnt IRL problem can be solved by finding Θ. Note, that through this derivation a couple important things are revealed. First, the learned reward function is simply the inner product of Θ and a feature vector. Second, given fixed Θ, finding $\hat{\pi}$ is very similar to solving the (forward) RL problem (note the similarity of (9) and (10) to (1) and (2)). What remains is a method to solve for Θ.

Because the optimization problem is convex, gradient descent techniques can be applied to find a globally optimal solution for Θ. The gradient of the dual MaxCausalEnt IRL optimization problem is defined as:

$$\nabla L_{Dual}\left(\Theta\right) = \mathbf{f}_{\mathcal{B}} - \mathbb{E}\left[\mathbf{f}_{(S,A)}|\hat{\pi}_\Theta\right] \tag{12}$$

$$= \mathbf{f}_{\mathcal{B}} - \sum_{(S,A) \in \mathcal{S} \times \mathcal{A}} P\left(S\right) \hat{\pi}_\Theta\left(A|S\right) \mathbf{f}_{S,A} \tag{13}$$

$$= \mathbf{f}_{\mathcal{B}} - \sum_{(S,A) \in \mathcal{S} \times \mathcal{A}} D_S \hat{\pi}_\Theta\left(A|S\right) \mathbf{f}_{S,A} \tag{14}$$

In Eq. (12) we see that the gradient is simply the difference between the empirical feature counts that are input to the problem (the observed behaviors) and the expected feature counts under our learned policy. Expanding the expectation into the sum over all states and actions (Eq. (13)), we see that the expectation is a function of the probability of being in a state, the learned policy, and the feature vectors. Finally, in Eq. (14) we rewrite the probability of being in a state as a variable D_S, which we call the *visitation frequency* of state S. From this derivation we see that in order to compute the gradient of the dual we first need to find two things:

1. The policy $\hat{\pi}_\Theta$
2. The frequency of visiting each state D_S when executing policy $\hat{\pi}_\Theta$

From this we can outline a high-level algorithm sketch for MaxCausalEnt IRL:

Note: To reiterate a previous point: Many different IRL algorithms behave similarly to Algorithm 1. The most common trait of IRL algorithms is the need to repeatedly find a policy under the current estimate of the reward function, and then using this policy, update parameters of the reward function. Finding such a policy is

Algorithm 1 High Level MaxCausalEnt IRL

procedure HL-MaxCausalEnt(MDP/R, $\mathcal{B}, \forall_{(S,A)\in\mathcal{S}\times\mathcal{A}}\mathbf{f}_{S,A}$) ▷ Input is the MDP without the reward, observed behaviors, and (for the deterministic case) feature vectors for all state/action pairs
  $\Theta \leftarrow randomInit$ ▷ Randomly initialize Θ
  repeat
    Find policy $\hat{\pi}_\Theta$ under reward R_Θ
    Find state visitation frequencies D under policy $\hat{\pi}_\Theta$
    Find R_Θ under policy $\hat{\pi}_\Theta$ and state visitation frequencies D
  until convergence
  return $R, \hat{\pi}_\Theta$
end procedure

exactly the (forward) RL problem for which any of the current state-of-the-art methods can be employed. For MaxCausalEnt IRL, the authors use a method that that aligns with the formalism established in this section. We sketch this algorithm in the next section.

In the next section we will outline the algorithms necessary to fill out Algorithm 1.

Note: A much longer and more formal treatment of the topics in this section can be found in [3]. Of particular interest are Chapters 5, 6, 9, and 10. More specifically, we utilize:

- The MaxCausalEnt optimization problem is defined in Definition 6.1
- The proof sketch deriving the optimal policy in their optimization problem is shown in Theorem 6.2
- The optimal policy in a Markovian model is in Corollary 6.5
- The optimal policy re-expressed as Soft Bellman Equations (the definition above) is in Theorem 6.8
- The gradient of the dual is in Equation 10.1

Also note that in order to keep this document self-consistent, our notation differs from what is found in [3].

3 Algorithm Sketches for MaxCausalEnt IRL

Note: Chapters 9 and 10 of [3] outline a multitude of algorithms for different solving the MaxCausalEnt optimization problem under different assumptions, and with certain approximations. In this document, we will focus on the most basic form of the MaxCausalEnt algorithm, but note that other forms exist.

In this section we will take the approach of defining the MaxCausalEnt algorithm from the top down, beginning with the gradient ascent procedure for finding Θ, specifying what parts of the algorithm need dedicated procedures, and then defining those. As such, this section will "unroll" the MaxCausalEnt algorithm into it's constituent procedures. We will also attempt to provide insight into how often each procedure is used and what data structures are necessary. Further, each algorithm in this section is in reference to an algorithm in [3], which will be noted in parentheses.

A couple important notes before going into the algorithm:

- The non-hyperparameter inputs to MaxCausalEnt IRL are:
  1. MDP/R (i.e. we know the state space $\mathcal{S}$, action space $\mathcal{A}$, state transition probabilities P_T, initial state distribution P_0, and set of terminal states $\mathcal{S}_\phi$)
  2. $\mathcal{B}$ (i.e. we have access to observed behaviors)
  3. $\mathcal{F} = \forall_{(S,A)\in\mathcal{S}\times\mathcal{A}}\mathbf{f}_{S,A}$ (i.e. we have feature vectors associated with each state and action pair).

- The algorithm is repeated takes gradient steps as in standard gradient ascent with the gradient step on line 10. Lines 5-9 compute the gradient over all states and actions.
- To simplify gradient computation we pre-compute $\mathbf{f}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{B \in \mathcal{B}} \frac{1}{|B|} \sum_{(S,A) \in B} \mathbf{f}_{S,A}$, as it is fixed throughout the algorithm.
- With $\mathbf{f}_{\mathcal{B}}$ pre-computed, the algorithms below focus mostly on finding the second term of (14). For notational brevity we define:

$$\nabla_{\Theta} F(S, A) = D_S \hat{\pi}_{\Theta}(A|S) \mathbf{f}_{S,A}. \tag{15}$$

We begin by outlining the gradient ascent algorithm (Algorithm 2), which more formally defines Algorithm 1.

Algorithm 2 Gradient Ascent for MaxCasualEnt IRL (Algorithm 10.4)

```
1: procedure MaxCausalEntGradAscent(F, f_B, MDP/R, λ, t)
2:     Θ ← randomInit                                  ▷ Randomly initialize Θ
3:     repeat
4:         ∇F ← 0                                      ▷ Initialize accumulator for gradient to all zeros
5:         for all ∈ S do
6:             for all A ∈ A do
7:                 ∇F ← ∇F + ∇_Θ F(S, A)               ▷ Sum the gradients over all state/action pairs
8:             end for
9:         end for
10:        Θ ← Θ + (λ/t)(f_B − ∇F)                     ▷ Take a gradient step
11:        t ← t + 1                                   ▷ Increase the learning rate decay
12:    until convergence
13:    return Θ
14: end procedure
```

Notes on Algorithm 2:

- We introduce two hyperparameters here:
  1. λ - A learning rate. Problem dependent and typically set through cross-validation
  2. t - A learning rate decay. The authors consider 1000 as a value for this, but can be set to other values via cross-validation. Note that this is incremented every gradient step and thus reduces the effective learning rate of the algorithm as it proceeds.
- This algorithm is standard gradient ascent with learning rate decay. Lines 5-9 compute the gradient as a sum of each gradient over all states and actions. Line 10 takes the gradient ascent step. Line 11 increases the learning rate decay.
- Each gradient descent step requires $\nabla_{\Theta} F$ to be computed for each state and action pair.
  - This implies that we can enumerate all states and actions. For continuous state and action spaces, this is not possible and other gradient computation methods must be used.
  - For large state and/or action spaces, such an enumeration can be extremely computationally and memory expensive. This procedure is known to as a *batch* gradient ascent procedure. Algorithm 10.5 in [3] defines a stochastic gradient technique that computes $\nabla_{\Theta} F$ for a single state and action each step each time a gradient step is taken. This can lead to slower per iteration convergence, but could drastically improve the practical run-time of the gradient ascent procedure.

What remains to be defined from Algorithm 2 is how to compute $\nabla_{\Theta} F(S, A)$ (defined in (15)) Since, D is dependent on $\hat{\pi}_{\Theta}$ we will next show how to find the policy parameterized by Θ. By 8, 9, and 10 we can see that if we find $V^{soft}_{\hat{\pi}_{\Theta}}$, then we can compute $\hat{\pi}_{\Theta}$. We find $V^{soft}_{\hat{\pi}_{\Theta}}$ with Algorithm 3.

Notes on Algorithm 3:

- We introduce a new hyperparameter here:

Algorithm 3 Soft Value Function Calculation Algorithm (Algorithm 9.1)

1: **procedure** VSoftCalculation(MDP/R, $\mathcal{F}, \Theta, \phi : \mathcal{S} \mapsto \mathbb{R}$)
2: **for all** $S \in \mathcal{S}$ **do**
3: $V^{Soft}(S) \leftarrow -\infty$
4: **end for**
5: **repeat**
6: **for all** $S \in \mathcal{S}$ **do**
7: $V^{Soft}(S)' \leftarrow \phi(S)$
8: **end for**
9: **for all** $S \in \mathcal{S}$ **do**
10: **for all** $A \in \mathcal{A}$ **do** ▷ This is for all $A \in \mathcal{A}$ such that A is a valid action when in S
11: $V^{Soft}(S)' \leftarrow \text{softmax}\left(V^{Soft}(S)', \Theta^T \mathbf{f}_{S,A} + \sum_{S \in \mathcal{S}} P_T(S'|S,A) V^{Soft}(S')\right)$
12: **end for**
13: **end for**
14: **for all** $S \in \mathcal{S}$ **do**
15: $V^{Soft}(S) \leftarrow V^{Soft}(S)'$
16: **end for**
17: **until** convergence
18: **return** V^{Soft}
19: **end procedure**

1. ϕ - A terminal state reward function. For MDPs with a set of terminal states $\mathcal{S}_\phi$, ϕ can be set to:

$$\phi(S) = \begin{cases} -\infty, & \text{if } S \notin \mathcal{S}_\phi \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

For numerical stability, the $-\infty$ can be set to a really small number, and reliable results can still be achieved

- The function denoted "softmax" is simply defined as:

$$\text{softmax}(x, y) = \log(e^x + e^y) \tag{17}$$

and can be found more efficiently by using Algorithm 9.2 in [3].

Note: To see how the "softmax" function gained it's name, it is potentially useful to rewrite it as:

$$\text{softmax}(x, y) = \log(e^x + e^y) = \max(x, y) + \log\left(1 + e^{\max(x,y) - \min(x,y)}\right) \tag{18}$$

Here, you can see that the softmax of two numbers is the maximum of the two plus a monotonically increasing function of how much more the maximum is more than the minimum. In this way, the softmax function upper bounds the maximum function, and increases as the difference between the minimum and maximum increases.

- The key line in Algorithm 3 is 11, which iteratively updates the value function estimation for each state. The softmax function takes two arguments:
  1. The old value before the update
  2. The Q^{soft} function as defined in (10). This itself has two terms:
     (a) The immediate reward for taking an action in the state (1st term)
     (b) The expected future reward (2nd term)

  In this way, the recurrence of (9) is computed dynamically, updating values in different states each iteration until V^{Soft} no longer changes between iterations.

Note: Algorithm 3 is very similar to the well-known value iteration method [1] for (forward) RL. Indeed, one can view Algorithm 3 as a "soft" version of value iteration as replacing the "softmax" function with a simple "maximum" recovers the Value Iteration algorithm.

Note: Algorithm 3 is designed for MDPs with a finite number of states in actions, and cannot be directly applied to cases where state and action spaces are infinite (e.g. when states and actions are continuous). However, infinite state and actions spaces can be handled via techniques similar to how the value iteration algorithm is extended to these cases.

- One way of viewing Algorithm 3 is that it propagates values from terminal states through all potential trajectories. The algorithm begins by assigning all values of V^{Soft} to $-\infty$, except the designated terminal states, which are assigned a constant defined by ϕ. After that, the value of each state is updated as a function the values in other states. Since other the values of other states are initially $-\infty$, only values of states which can reach a terminal state in a single action are updated to be not $-\infty$. In the next iteration, states that can reach any of the states updated to not have value $-\infty$ last iteration in a single action are updated to be not $-\infty$, and so on.
- Note that the sum expressed in line 11 is over all states. However, it can be drastically simplified by keeping record of what states have non-zero probability of transitioning to when taking each action in each state. The sum then is only over states S' for which $P_T\left(S'|A,S\right)$ is non-zero.

After running Algorithm 3 we have a learned policy $\hat{\pi}_\Theta$. The last part of computing the gradient defined in (15), is finding the visitation frequencies under the policy $\hat{\pi}_\Theta$. We illustrate this procedure in Algorithm 4.

Algorithm 4 Expected State Visitation Fequency Calculation (Algorithm 9.3)

1: **procedure** StateFrequencies(MDP/R, $\pi : \mathcal{A} \times \mathcal{S} \mapsto [0,1]$)
2: **for all** $S \in \mathcal{S}$ **do**
3: $D_S \leftarrow 0$
4: **end for**
5: **repeat**
6: **for all** $S \in \mathcal{S}$ **do**
7: $D'_S \leftarrow P_0\left(S\right)$
8: **end for**
9: **for all** $S \in \mathcal{S}$ **do**
10: **for all** $A \in \mathcal{A}$ **do** ▷ This is for all $A \in \mathcal{A}$ such that A is a valid action when in S
11: **for all** $S' \in \mathcal{S}$ **do**
12: $D'_{S'} \leftarrow D'_{S'} + D_S \pi\left(A|S\right) P_T\left(S'|A,S\right)$
13: **end for**
14: **end for**
15: **end for**
16: **for all** $S \in \mathcal{S}$ **do**
17: $D_S \leftarrow D'_S$
18: **end for**
19: **until** convergence
20: **return** $\forall_{S \in \mathcal{S}} D_S$
21: **end procedure**

Notes on Algorithm 4:

- Algorithm 4 behaves much like Algorithm 3 in that it computes the visitation frequency of a state dynamically as a function of other state's visitation frequencies. The inner most loop stating on line 11 iterates over each state S' and estimates the frequency of an agent under policy π will visit that state as a function of all states S that can reach S' in a single action A, the probability of taking A in S, and the probability of transitioning to S' from S when taking A. The algorithm converges when state frequencies change negligibly between iterations of the outer-most loop.
- The loop beginning on line 11 can be simplified by keeping record of possible transition states as can be done with the sum on line 11 of Algorithm 3.

With Algorithm 4, we have everything neccessary to find the MaxCausalEnt solution to the IRL problem.

4 Summary and Conclusions

MaxCausalEnt IRL is a technique for modeling sequential behavior from observation. The algorithm adopts a Markov Decision Process as the model of observed behaviors and attempts to learn the reward structure indicated by the observations. It also employs the concept of maximum entropy to handle ambiguities from lack of a complete set of observations. Practical considerations can be summarized as follows:

The inputs to the MaxCausalEnt IRL algorithm are:

1. A formal definition of the problem in the form of a Markov Decision process without a reward function: MDP/R
2. A set of observed behaviors $\mathcal{B}$ (i.e. we have access to observed behaviors)
3. A set of feature/action vectors $\mathcal{F} = \forall_{(S,A)\in\mathcal{S}\times\mathcal{A}}\mathbf{f}_{S,A}$

The hyperparameters in the MaxCausalEnt IRL algorithm are:

1. $\lambda \in \mathbb{R}$ - A learning rate
2. $t \in \mathbb{Z}^+$ - A learning rate decay
3. $\phi : \mathcal{S} \mapsto \mathbb{R}$ - A terminal state reward function.
4. Stopping criteria for Algorithms 2, 3, and 4

To find a reward function and policy:

1. Perform gradient ascent as defined in Algorithm 2. To compute the gradient at each step:
   (a) Find the policy for under the current reward using Algorithm 3.
   (b) Find the visitation counts for each state under the current policy using Algorithm 4

By learning a model using MaxCausalEntIRL a multitude of tasks are possible:

1. **Anomaly Detection**: If the states and actions in a new observed behavior are compared to the policy or soft Q-function learned using MaxCausalEnt IRL, it can be determined how probable the new behavior is under the model. If the new trajectory does not follow the learned policy or takes actions that have low Q-function value, then it can be deemed anomalous.
2. **Simulation of Observed Behavior** Trajectories can be simulated by taking sequential actions according to the probability distribution defined by the policy learned in MaxCausalEnt IRL.
3. **Teaching from Expert Behavior** If a MaxCausalEnt IRL model is learned from expert demonstrations, it can be used to teach novices how to perform tasks. For instance, if a novice finds themselves in a state where they are unsure of the next action, the learned policy can show them what the expert would do.
4. **Inverse Optimal Control** Instead of learning a policy using reinforcement learning, MaxCausalEnt IRL can be used to learn a control policy from demonstrated behaviors, potentially reducing the need for exploration common in using reinforcement learning for control tasks.
5. **Providing a Tool for Explaining Behavior** Because IRL is meant to model observed behavior, the various components (the learned policy, reward function, value function, etc.) can be probed to see the relative importance of particular decisions. For example, if a particular state and action receives high reward, this may indicate a important step for the agent being observed in accomplishing a task.

References

[1] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[2] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2586–2591. Morgan Kaufmann Publishers Inc., 2007.

[3] Brian D Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. 2010.

[4] Brian D Ziebart, J Andrew Bagnell, and Anind K Dey. Modeling interaction via the principle of maximum causal entropy. In *Proceedings of the 27th International Conference on Machine Learning*, pages 1255–1262. Omnipress, 2010.