Generalized Inverse Reinforcement Learning

James MacGlashan[*†]
Cogitai, Inc.
james@cogitai.com

Nakul Gopalan[*]
Brown University
ngopalan@cs.brown.edu

Michael L. Littman
Brown University
mlittman@cs.brown.edu

Amy Greenwald
Brown University
amy@cs.brown.edu

Abstract

Inverse Reinforcement Learning (IRL) is used to teach behaviors to agents, by having them learn a reward function from example trajectories. The underlying assumption is usually that these trajectories represent optimal behavior. However, it is not always possible for a user to provide examples of optimal trajectories. This problem has been tackled previously by labeling trajectories with a score that indicates bad, as well as good, behaviors. In this work, we formalize the IRL problem in a generalized framework that allows for learning from good and bad demonstrations, and everything in between. In our framework, users can score entire trajectories (as usual) as well as particular state-action pairs. This additional control allows the agent to learn preferred behaviors from a relatively small number of trajectories. We expect this framework to be especially useful in robotics domains, where the user can collect fewer trajectories at the cost of labeling bad state-action pairs, which might be easier than maneuvering a robot to collect additional (entire) trajectories.

Keywords: Inverse Reinforcement Learning

[1]Both authors contributed equally to this work.
[2]Work was done while at Brown University.

1 Introduction

Teaching desired behaviors to an agent is an important problem in artificial intelligence. One approach to this problem is to use Inverse Reinforcement Learning (IRL) [1, 4], in which demonstration trajectories are provided, based on which behavior is learned, by first learning a reward function, which is an intermediate generalizable representation of behavior. Recent IRL methods have considered scoring the trajectories collected, so that the agent learns to prefer behaviors with high scores and avoid behaviors with low scores. Our work builds on this idea, but goes one step further by allowing the user to point out the reasons for a high or a low score. More formally, we propose a generalized framework for IRL, where it is possible to score the state-actions pairs of a trajectory, as well as the trajectory itself. The hope is that this added information might facilitate the training of complex behaviors from fewer demonstration trajectories in problem domains like robotics, where it is particularly expensive to collect trajectories.

In this short paper, we first summarize the state of the art in Inverse Reinforcement Learning. We then discuss our framework, as well as two learning algorithms that can perform inference over the proposed framework. Finally, we show example reward functions and behaviors learned from demonstrations using our algorithms for a sample grid world problem.

2 Related Work

Two competing approaches to teaching behaviors to agents are supervised learning[6] and reinforcement learning [1, 4]. The advantage of using Reinforcement learning over supervised learning is that it allows the agent to repeat the learned behavior in a different environment than the one it was trained in, as a reward function is learned. A reinforcement learning problem is generally modeled as a Markov Decision Process (MDP), which is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \mathcal{R}, \gamma)$, as defined in [8]. In some applications, it can be very difficult to design a reward function $\mathcal{R}$, but it might be easy to provide example trajectories as behaviors to learn from. Inverse Reinforcement Learning [1, 4] is a problem reformulation, where behaviors are provided, and the goal is to teach agents behaviors by having them learn reward functions from example trajectories. Traditional Inverse Reinforcement learning approaches did not let agents learn via failure; instead, demonstrations were interpreted as optimal behaviors.

Previous methods [3, 7] for learning from imperfect demonstrations scored the trajectories. Burchfiel et al. [3] tries to estimate the weights of a parameterized reward function by learning optimal weights for the scoring function, given trajectories and scores from experts and non-experts. Shiarlis et al. [7] assumes a labeled dataset with failed and successful trajectories, and learns the weights for a parameterized reward function such that the expected policy under the learned reward function is far away from the failed demonstrations and close to the successful demonstrations. Both works report that they require fewer trajectories than vanilla IRL to learn complicated behaviors, as more label information is available. We propose providing still more direct information, as a user might be able to label bad segments of a trajectory directly. This approach would be useful in scenarios where collecting data is harder than labeling segments of a trajectory as good or bad.

3 Generalized IRL (GIRL)

In this section, we develop a graphical model for the IRL problem over which inference can be performed to solve for a parameterized reward function. We then describe inference techniques that can be used within this framework to solve for the reward function.

3.1 Graphical Model

Consider a demonstration trajectory of length N given to the user and the agent. The user might provide a final label L to the trajectory demonstrated. This label L is considered by us to be a random function of what they thought, positive or negative, about each of the actions A selected by the agent in the trajectory. The motivation for this model is that we consider a final label (L) that a user gives a trajectory of size N to be some random function of what they thought about each of the action selections (A) exhibited in the trajectory. However, these step-wise evaluations (X) are mostly unobserved in the data, unless specifically labeled by the user. The reward function parameterized by $\boldsymbol{\theta}$ that dictates the stepwise evaluation is also an unobserved variable. We have shown the plate model in Figure 1, where the observed variables are in shaded nodes and the unobserved variables are in unshaded nodes.

We model the probability that an action is evaluated as good or not as proportional to its selection probability according a softmax policy computed for the reward function with parameters $\boldsymbol{\theta}$. Specifically:

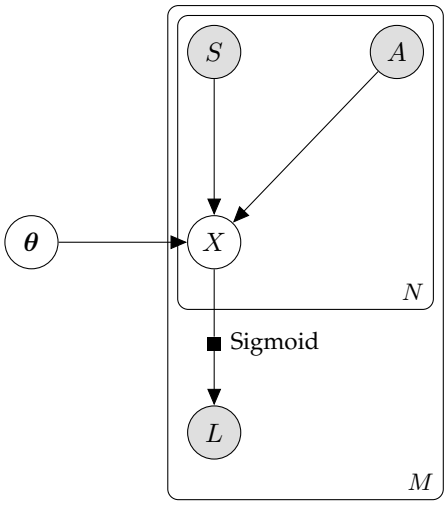


Figure 1: Plate Diagram of the Label Probability Model

$$\Pr(x_i = +1|s, a, \theta) = \pi(s, a|\boldsymbol{\theta}) \tag{1}$$

$$\Pr(x_i = -1|s, a, \theta) = 1 - \pi(s, a|\boldsymbol{\theta}), \tag{2}$$

where $\pi(s, a|\boldsymbol{\theta})$ is the softmax policy over Q-values computed for the reward function parameterized by $\boldsymbol{\theta}$:

$$\pi(s, a|\boldsymbol{\theta}) = \frac{e^{\beta Q(s,a|\boldsymbol{\theta})}}{\sum_{a'} e^{\beta Q(s,a'|\boldsymbol{\theta})}}, \tag{3}$$

β is a selectable parameter, and $Q(s, a|\boldsymbol{\theta})$ is the Q-value computed for the reward function parameterized by $\boldsymbol{\theta}$. This idea of having state-action pair labels have a probability distribution with respect to the expected optimal policy is similar to the ideas explored in SABL [5].

For the probability distribution of L, given the sequence of N step-wise labels, we would like a distribution that has the property that as more step-wise labels are positive, the probability of a positive trajectory label increases (and vice versa). Although there are many possible distributions that satisfy this property, for concreteness, we choose the sigmoid function. That is,

$$\Pr(L = +1|X_1, ..., X_n) = \frac{1}{1 + e^{-\phi \sum_i^N X_i}} \tag{4}$$

$$\Pr(L = -1|X_1, ..., X_n) = 1 - \Pr(L = +1|X_1, ..., X_n), \tag{5}$$

where ϕ is a selectable parameter that tunes how quickly of a majority of step-wise labels increases/decreases the probability of trajectory assignment. For example, when $\phi = 0$, trajectory labels are assigned uniformly randomly independently of step-wise labels. As $\phi \to \infty$, the sigmoid converges to a step function in which a trajectory containing even one more positive step-wise label than negative step-wise label will deterministically cause a positive trajectory label (and vice versa).

The label probability model described above gives a generalized IRL formulation over which inference can be performed, with or without labels. If the given data set has no trajectory labels, then all trajectories can be set to have the same positive label. Next we will describe inference over this model using Expectation Maximization (EM) and a faster EM method using importance sampling.

3.2 Inference

The problem with estimating the $\boldsymbol{\theta}$ parameters of our reward function is that we have a latent variable vector X (or more generally, some of the elements of the X vector are latent, and some may be observed), which prevents us from easily computing the likelihood of the model and maximizing parameters for it. The EM approach to solving this problem is to first choose values for $\boldsymbol{\theta}$; then choose a new $\boldsymbol{\theta}$ that maximizes the expected value of the log likelihood function where the distribution of the expectation is the probability distribution of latent variables (the Xs in our case) given the observations available and previous $\boldsymbol{\theta}$ choice; and then repeating this process. The maximization process can be performed using gradient ascent, which is similar to the ideas explore in MLIRL [2].

To formalize this process for our problem, first note that the likelihood of our parameters (and state-action sequence) given an $\boldsymbol{x}$ vector and label l is

$$\mathcal{L}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}|l, \boldsymbol{x}) = \Pr(l|\boldsymbol{x}) \prod_i \Pr(x_i|s_i, a_i, \boldsymbol{\theta}) \tag{6}$$

and the log likelihood is

$$\log \mathcal{L}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}|l, \boldsymbol{x}) = \log \Pr(l|\boldsymbol{x}) + \sum_i \log \Pr(x_i|s_i, a_i, \boldsymbol{\theta}). \tag{7}$$

Additionally, the gradient of the log likelihood is,

$$\nabla_{\boldsymbol{\theta}} \log \mathcal{L}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}|l, \boldsymbol{x}) = \sum_i \frac{\nabla_{\boldsymbol{\theta}} \Pr(x_i|s_i, a_i, \boldsymbol{\theta})}{\Pr(x_i|s_i, a_i, \boldsymbol{\theta})}. \tag{8}$$

To simplify the EM algorithm description, we will introduce the notation $\boldsymbol{x}_k$ to indicate the subset of observed elements in an $\boldsymbol{x}$ vector, and $\boldsymbol{x}_u$ to represent a possible assignment to the subset of the unobserved values of an $\boldsymbol{x}$ vector. Using this

notation, the expected value of the log likelihood under some candidate parameter $\boldsymbol{\theta}'$ for missing X elements distributed according to $\boldsymbol{\theta}$ is

$$E_{\boldsymbol{x}_u \sim \Pr(\boldsymbol{x}_u|l,\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})} \left[\log \mathcal{L}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}'|l,\boldsymbol{x})\right] = \sum_{\boldsymbol{x}_u} \Pr(\boldsymbol{x}_u|l,\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}) \log \mathcal{L}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}'|l,\boldsymbol{x})$$

Given that, the corresponding EM algorithm operating on a single trajectory is as follows (it is generalized to many trajectories by simply summing over each trajectory).

Algorithm 1 Labeled-IRL EM Algorithm

Require: initial $\boldsymbol{\theta}_0$, and data $\boldsymbol{s}$, $\boldsymbol{a}$, $\boldsymbol{x}_k$, l
  for $t = 0$ to K **do**
    $\boldsymbol{\theta}_{t+1} \leftarrow \arg\max_{\boldsymbol{\theta}'} \sum_{\boldsymbol{x}_u} \Pr(\boldsymbol{x}_u|l,\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}_t) \log \mathcal{L}(\boldsymbol{\theta}', \boldsymbol{s}, \boldsymbol{a} \mid l, \boldsymbol{x}_k, \boldsymbol{x}_u)$
  end for

To compute the expected value, we need to enumerate each of the possible assignments to the unknown $\boldsymbol{x}$ elements and compute the probability of them given the observed data and model parameters $\boldsymbol{\theta}$. This probability is computed as

$$\begin{aligned}
\Pr(\boldsymbol{x}_u|l,\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}) &= \frac{\Pr(l|\boldsymbol{x}_k,\boldsymbol{x}_u)\Pr(\boldsymbol{x}_u|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})\Pr(\boldsymbol{x}_k|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})\Pr(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}{\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})\Pr(\boldsymbol{x}_k|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})\Pr(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})} \\
&= \frac{\Pr(l|\boldsymbol{x}_k,\boldsymbol{x}_u)\Pr(\boldsymbol{x}_u|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}{\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})} \\
&= \frac{\Pr(l|\boldsymbol{x}_k,\boldsymbol{x}_u)\prod_i \Pr(\boldsymbol{x}_{u,i}|s_i,a_i,\boldsymbol{\theta})}{\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}.
\end{aligned}$$

A straightforward computation of $\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})$ requires marginalizing over all possible assignments to the unknown X elements; however we can exploit the fact that $\Pr(l|\boldsymbol{x}_k,\boldsymbol{x}_u,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})$ is a function of the sum of the feedback values, the marginalization can be reduced to a summation that iterates over a number of values that is a linear function of the number of unobserved feedbacks.

Unfortunately, even with an efficient means to compute $\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})$, when the number of unknown X variables is large, the number of $\boldsymbol{x}_u$ assignments enumerated in the expectation's outer sum grows exponentially, and the product series over each of unknown element probabilities in the above equation ($\prod_i \Pr(\boldsymbol{x}_{u,i}|s_i,a_i,\boldsymbol{\theta})$) can have underflow issues. A resolution to this problem is to estimate the expectation with sampling. Monte Carlo sampling is unfortunately intractable because it is not easy to sample from $\Pr(\boldsymbol{x}_u|l,\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})$; moreover, it would not address the underflow issue in the product series. However, it is easy to sample from $\Pr(\boldsymbol{x}_u|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})$ (removing the conditioning on the label), which we can use in importance sampling. With importance sampling, we can replace the expectation computation with the sample-estimate

$$\frac{1}{C}\sum_j^C \frac{\Pr(\boldsymbol{x}_u^j|l,\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}{\Pr(\boldsymbol{x}_u^j|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})} \log \mathcal{L}(l,\boldsymbol{x}_k,\boldsymbol{x}_u^j|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}), \tag{9}$$

where $\boldsymbol{x}_u^j$ is sample from the distribution $\Pr(\boldsymbol{x}_u|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})$. This simplifies further to:

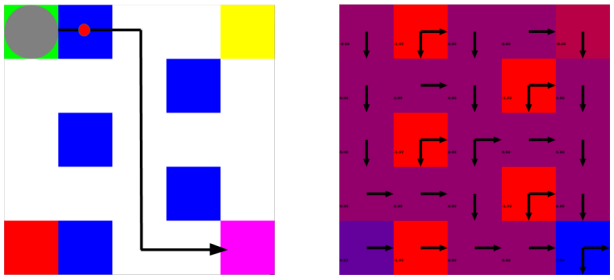
$$\begin{aligned}
\frac{\Pr(\boldsymbol{x}_u^j|l,\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}{\Pr(\boldsymbol{x}_u^j|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})} &= \frac{\Pr(l|\boldsymbol{x}_k,\boldsymbol{x}_u)\Pr(\boldsymbol{x}_u^j|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}{\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})} \frac{1}{\Pr(\boldsymbol{x}_u^j|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})} \\
&= \frac{\Pr(l|\boldsymbol{x}_k,\boldsymbol{x}_u)}{\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}
\end{aligned}$$

Consequently, we have removed the product series from the expectation weight, thereby avoiding underflow issues. Also, as noted previously, the $\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})$ term can be computed efficiently with dynamic programming. Now we can write a tractable EM algorithm where we can compute the maximization using gradient ascent.

Algorithm 2 Labeled-IRL Approximate EM Gradient Ascent Algorithm

Require: initial $\boldsymbol{\theta}_0$; data $\boldsymbol{s}$, $\boldsymbol{a}$, $\boldsymbol{x}_k$, l; and learning rate α
  for $t = 0$ to K **do**
    draw $j = 1$ to C samples of $\boldsymbol{x}_u^j \sim \Pr(\boldsymbol{x}_u|\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}_t)$
    for $j = 1$ to C **do**
      $w_j \leftarrow \frac{\Pr(l|\boldsymbol{x}_k, \boldsymbol{x}_u)}{\Pr(l|\boldsymbol{x}_k, \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}_t)}$ ▷ Expectation step
    end for
    $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}_t$
    for 1 to M **do** ▷ Gradient ascent maximization loop
      $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}' + \alpha \frac{1}{C} \sum_j^C w_j \sum_{x_i \in \boldsymbol{x}_k \cup \boldsymbol{x}_u^j} \frac{\nabla_{\boldsymbol{\theta}'} \Pr(x_i|s_i, a_i, \boldsymbol{\theta}')}{\Pr(x_i|s_i, a_i, \boldsymbol{\theta}')}$
    end for
    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}'$
  end for

4 Results on Grid World



(a) (b)

Figure 2: (a) Shows the single input trajectory to train the behavior. The red dot shows the state for which the agent got a negative feedback. (b) Shows the output policy and Value function learned for each state, given the labeled trajectory.

We present results on a grid world, using the EM method. The grid world is shown in Figure 2a. We want to teach a behavior in which the agent walks to the pink cell, while avoiding the blue cells. During demonstration the agent walks over a single blue cell. We mark the state-action pair that landed the agent in the blue with a negative feedback, shown with a red dot, in Figure 2a. We used one trajectory with eight steps, with $\beta = 1$, and $\phi = 1$. The overall label for the trajectory is positive as the agent reached the pink cell. After performing GIRL we learn a reward function and we have displayed the value function and the optimal policy for the entire grid as shown in Figure 2b. It is obvious that the agent now prefers to walk to the pink grid cell while avoiding the blue cells. Teaching this behavior would require two separate trajectories, if failed trajectories were included, in all the state of the art failure IRL methods discussed previously. Hence, we can learn complex behaviors with fewer example trajectories.

5 Conclusion

Our approach allows us to provide individual state-action pair feedback for IRL, along with labels on state-action pairs, which allows for more expressive feedback with fewer sample trajectories. The inference algorithms we developed performs this inference in real time on our toy example. The algorithm itself will be very useful in teaching behaviors with active learning as well as in two agent games, where one agent teaches the rules of the game to the other.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[2] Monica Babes, Vukosi Marivate, Kaushik Subramanian, and Michael L Littman. Apprenticeship learning about multiple intentions. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 897–904, 2011.

[3] Benjamin Burchfiel, Carlo Tomasi, and Ronald Parr. Distance minimization for reward learning from scored trajectories. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[4] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *ICML*, pages 663–670, 2000.

[5] Bei Peng, James MacGlashan, Robert Loftin, Michael L Littman, David L Roberts, and Matthew E Taylor. A need for speed: adapting agent action speed to improve task learning from non-expert humans. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 957–965, 2016.

[6] Stefan Schaal et al. Learning from demonstration. *Advances in neural information processing systems*, pages 1040–1046, 1997.

[7] Kyriacos Shiarlis, Joao Messias, and Shimon Whiteson. Inverse reinforcement learning from failure. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1060–1068, 2016.

[8] Philip S. Thomas. A notation for markov decision processes. *CoRR*, abs/1512.09075, 2015. URL `http://arxiv.org/abs/1512.09075`.