Model Free IRLF - Proposal

Kyriacos Shiarlis[1], Shimon Whiteson[2]

***Abstract*— Inverse Reinforcement Learning (IRL) methods allow agent to learn complex tasks from demonstration making them especially appealing in cases where reward functions are either too sparse or non-existent. Recent developements in the field have allowed IRL to perform well in substancially larger spaces than before. This paper describes methods that extend the Model-free Inverse Reinforcement Learning methods allowing failed demonstrations to be incorporated in the learning. We begin with a a general description of IRL and describe how it can be applied to large statespaces. We then go on to show that the framework can be easily extended to exploit data from failed trajectories i.e., trajectories with a high cost. Our experiments show that our framework can be used to learn an imitation policy in large state spaces faster and better.**

I. Introduction

II. Generative Adversatial Imitation learning

We will denote the expert policy as π_E and the agent policy by π_A. We look for a cost function $c \in C$ that appoints low cost exclusively to data from π_E.

In the original IRL setting the optimisation problem can be broken down into two loss functions to be minimised.

$$\mathcal{L}_c = E_{\pi_E}[c(s,a)] - E_{\pi_A}[c(s,a)] \tag{1}$$

$$\mathcal{L}_\pi = -H(\pi_A) + E_{\pi_A}[c(s,a)] \tag{2}$$

Where $H(\pi_A)$ is the causal entropy of the policy π_A and promotes policies to be more flat. The first minimisation is performed using gradient descent by taking the derivative of $\mathcal{L}_c$ with respect to c. The second is performed by planning or reinforcement learning (RL) on the underlying MDP. The first minimisation attempts to assign a large cost to state action pairs used by π_A and low cost to the expert trajectories (π_E). The second minimisation tries to find the policy with the lowest possible cost.

In practice the two minimisations are never carried out in full for every iteration. Instead the problem is seen as reinforcement learning where the cost function is constantly changing (in the inner loop). In turn RL can be seen as an adaptive sampler that samples the lowest cost regions of c(s,a) and thus allows the gradient with respect to Equation 1 to be computed more accurately.

Using different models for the cost function $c(s,a)$ and the RL step in equation 2 can yield a range of algorithms. A recent method called Generative Adversarial Imitation learning assumes a cost function of the form $c(s,a) = -\log(D(s,a))$ where D(s,a) is a discriminative classifier that maps $S \times A \rightarrow 1$. The discriminator attempts to output D(s,a) =1 for samples originating from the expert and D(s,a) for samples originating from the agent. Equation 1 hence takes the following from:

$$\mathcal{L}_c = -E_{\pi_E}[\log(D(s,a))] - E_{\pi_A}[\log(1 - D(s,a))] \tag{3}$$

Note however that the following form can also be used, which is more consistent with Equation 1

$$\mathcal{L}_c = -E_{\pi_E}[\log(D(s,a))] + E_{\pi_A}[\log(D(s,a))] \tag{4}$$

and is more consistent with the cost function used in the RL step in practice.

$$\mathcal{L}_\pi = -H(\pi_A) - E_{\pi_A}[\log(D(s,a))] \tag{5}$$

In this paper we will be using the GAIL formulation mainly because of its intuitive nature. Most ideas in this paper however generalise to other forms of this cost function.

An important feature of GAIL and in general IRL, is that the trajectories are in essence labeled as coming from an expert. This in turn implies that the demonstrated trajectories should have low cost which brings about the formulation we have seen above. In the next sections we will intoduce *failed* demonstartions into the framework, i.e., demonstrations with that have a high cost under the underlying cost function.

III. GAIL from failure

In this section we assume the presence of failed demonstrations in our dataset. In contrast with expert demonstrations which are assumed to have minimal cost, failed demonsrations represent trajectories with a high cost. We will represent the policy of failed demonstrations as pi_f and we would like our final policy π_A to be as different from pi_F as possible. In this section we document several ways to incorporate this aditional data in the GAIL framework, discussing the benefits and caveats of each approach.

A. Different Views of failure

Since in IRL we try to make trajectories from π_E have less cost than all other trajectories, a natural first natural extension of the original formulation would be to treat data from π_E as just data for which the cost should be high, resulting in:

$$\mathcal{L}_c = E_{\pi_E}[-\log(D(s,a))] + E_{\pi_A}[\log(D(s,a))] + E_{\pi_F}[\log(D(s,a))] \tag{6}$$

[1]Informatics Institute, University of Amsterdam, The Netherlands, {k.c.shiarlis,j.messias}@uva.nl

[2]Department of Computer Science, University of Oxford, United Kingdom, shimon.whiteson@cs.ox.ac.uk

As a result the additional data will be used in order to better estimate the gradient of $\mathcal{L}_c$. While this approach is very simple and straightforward it only makes use of the information that π_F is not π_E and does not use the fact that these are failed demonstrations with a very high (if not the highest possible) cost. We would ideally however like the to use the additional data to facilitate the search for a policy that not only resembles π_E maximally but it is also maximally different from π_F. To do this we could introduce another cost estimator and as a consequence a third optimisation problem.

$$\mathcal{L}_{c'} = E_{\pi_A}[\log(D'(s,a))] - E_{\pi_F}[\log(D'(s,a))] \quad (7)$$

Minimising this expression assigns low cost to π_A and a high cost to the failed demonstrations. Note that as before $D'(s,a) = 1$ denotes that (s,a) belongs to the data. The cost function $\log(D'(s,a))$ can then be used to drive the policy away from π_F using an augmented version of the minimisation in equation 2:

$$\mathcal{L}_\pi = -H(\pi) - E_\pi[[\log(D(s,a))) + \lambda \log(D'(s,a))]. \quad (8)$$

Where λ is a parameter that determines how much attention we pay to the failed demonstrations. An important observation at this point is that the relationship between the third term in 8 and the minimisation in is no longer adversarial. This is reasonable since we are clearly not interested in matching the distribution π_F but to avoid it.

A second observation is that we have ended up with a parameter, λ that we need to tune. To do so we will use the relationship between π_F and π_E which we have not yet exploited. Asume therefore a third classifier $D''(s,a)$ which discriminates between π_E and π_F this minimizing,

$$\mathcal{L}_{D''} = E_{\pi_E}[\log(D''(s,a))] + E_{\pi_F}[\log(1 - D''(s,a))] \quad (9)$$

At the minimum this classifier provides us with a probability that a state action pair comes from π_E or π_F which in turn acts as a state-dependent weight for the state action pairs encountered during Equation 8 resulting in,

$$\mathcal{L}_\pi = -H(\pi) - E_\pi[[D''(s,a)\log(D(s,a))) + (1 - D''(s,a))\log(D'(s,a))]. \quad (10)$$

Intuitively, if a state-action pair is seen as being closer to π_E i.e., D"(s,a)¿0.5 more weight is given to the cost assigned by $D(s,a)$ conversely if D"(s,a)¡0.5 more weight is given to $D'(s,a)$. In adition with providing some way to weigh the two cost terms this formulation has a the propery of dealing with the partial trajectory problem mentioned earlier (need to mention this earlier). If a part of trajectories from π_F and π_E is shared then the state action pairs will not be separable resulting in $D''(s,a) = 0.5$ which would potentially cancel the terms in 10 out.

B. Learning from Failure as a three class problem

A key observation that can be made now is that the two cost functions (classifiers) D(s,a) and D'(s,a) along with their respective optimisations $\mathcal{L}'_c$, $\mathcal{L}_c$ can be essentially folded into a single three class classification problem that tries to classify samples from each policy (π_E,π_F,π_A) into its respective class (Proof needed). The minimisation can then be seen as minimizing the cross entropy for this classifier. The policy optimisation step can be then seen as trying to fool this classifier into not only classifying the policy as part of the expert but also as trying to minimise the chances that a state action pair is classified as coming from π_F. We denote as $P_C(\pi = \pi'|s,a)$ the probability of classifying a sample as coming from a policy π'. The new optimisation problem would then be:

$$\mathcal{L}_C = -\Big[E_{\pi_E}[\log(P_C(\pi = \pi_E))] + \quad (11)$$

$$E_{\pi_A}[\log(P_C(\pi = \pi_A))] + \quad (12)$$

$$E_{\pi_F}[\log(P_C(\pi = \pi_F))]\Big] \quad (13)$$

$$\mathcal{L}_\pi = -H(\pi) + E_{\pi_A}(\log(P_C(\pi = \pi_E|s,a)) - \log(P_C(\pi = \pi_F|s,a))) \quad (14)$$

C. An example

Need to come up with this.

IV. Experiments

A. Continuous control

Demonstrate the validity of the method in continuous control tasks.

Demonstrations will be collected as follows:

- Solve the task using an RL method.
- Collecte N_E best trajectories from the learning.
- Collect N_F worse trajectories from the learning.
- Use this data as successful and failed demonstrations.

Using these relatively simple tasks I would like to investigate if learning from failure can:

- Make learning more stable (do learning at various learning rates)
- Make learning more data efficient by needing less expert demonstrations
- make learning better in general.

Preliminary results here.

B. Video games

I will use 2 video games Pong and Mario cart. Pong I will use because it can be easily solved using simple Deep RL methods such as DQN. Mario cart I will use because I found a way to collect demonstrations from it. With pong I will do a similar analysis to the continuous control case. With mario I will only learn using actual demonstrations and not the real reward signal so show that this can be applied to real world settings.