Samin Yeasar Arnob
ID: 260800927
COMP767: Reinforcement Learning
McGill University

Adversarial Inverse Reinforcement Learning in Dynamic Environment

1 Introduction

In Reinforcement Learning (RL) the fundamental idea is to provide an positive or negative reinforcement to Agent for it's action, so that it learns the expected behavior. And for that it's required to design a reward function. But such hand engineered reward function can be flawed or can miss to express the desired outcome. The way human and a RL agent perceive model of the world can be vastly different. Thus in inverse reinforcement learning, we want to learn the underlying reward function from the perspective of RL agent while it learns expert policy.
In this project, I have implemented an Adversarial Inverse Reinforcement Learning Algorithm. Where I have followed the implementation of Discriminator-Actor-Critic (DAC) Kostrikov et al. [2019] to implement the policy generator and followed the implementation of (AIRL) Fu et al. [2018] to implement the discriminator. Even though DAC uses Twin-delayed-deep-deterministic policy gradient (TD3) Fujimoto et al. [2018] as generator and takes deterministic action. But to implement AIRL, the policy requiress to be stochastic to compute discriminator output. So instead of TD3, I have used Soft-Actor-Critic (SAC) Haarnoja et al. [2018].
Goal of the project is to investigate robustness of the reward function learned from Adversarial Inverse Reinforcement Learning in Dynamic environments.

2 Background

Method of modern inverse reinforcement learning (IRL) build on top of maximum casual entropy IRL framework Ziebart et al. [2010], which does an entropy regularized Markov decision process (MDP), defined by the tuple $(S, A, T, r, \gamma, \rho_0)$. S,A are the state and action space, respectively, $\gamma \in (0, 1)$ is the discount factor. The dynamic or transition distribution $T(s'|s, a)$, initial state distribution $\rho_0(s)$ and reward function $r(s, a)$ are unknown in inverse reinforcement learning setup and can be learned through interaction with the MDP.
The goal of reinforcement learning is to find the optimal policy $\pi*$ that maximizes the expected entropy-regularized discounted reward under π,T and ρ_0:

$$\pi^* = arg\ max_\pi \mathbf{E}_{\tau\sim\pi}\Big[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) + H(\pi(.|s_t))\Big] \tag{1}$$

Where $\tau = (s_0, a_0 ... s_T, a_T)$ defines the sequence of states and actions induced by policy and dynamics. In Inverse reinforcement learning (IRL), expert demonstrations are given and agents are expected to produce trajectory which are similar to the expert and infer reward for given demonstration $D = \tau_1, \tau_2$. Ziebart et al. [2010] proves, probability with equal reward/cost are equal likely

$$P_\theta(\tau) \propto e^{R(\tau)} = \frac{1}{z} e^{R(\tau)} \tag{2}$$

$$\propto p(s_0) \prod p(s_{t+1}|s_t, a_t) e^{\gamma^t r_\theta(s_t, a_t)} \tag{3}$$

Where $z = \int_\tau e^{R(\tau)}$ is the partition function. It can be interpreted as solving a maximum likelihood problem

$$max_\theta \mathbf{E}_{\tau \sim D}[\log p_\theta(\tau)] \tag{4}$$

But it's not feasible to apply in large state-action space. GAIL Ho and Ermon [2016] was the first framework that drew connection between generative adversarial network and imitation learning which worked on continuous state-action space. It uses single neural network as generator, which tries to learn the distribution of expert policy and a discriminator network that works as a binary classifier to distinguish expert and generator trajectory. Optimizes two networks with following entropy regularized objective:

$$\mathbf{E}_{\pi_\psi}[\log D_\theta(s, a)] + \mathbf{E}_{\pi_E}[\log(1 - D_\theta(s, a))] - \lambda H(\pi) \tag{5}$$

where π_ψ and π_E is policy of generator and expert.
For being an imitation learning algorithm, instead of learning cost/reward function, it only recovers expert policy and thus fails miserably in dynamic environments. Later on, GAN-GCL Finn et al. [2016], an IRL algorithm provided with a discriminator that learns policy as well as the cost/reward function by computing following discriminator/reward optimization

$$D_\theta = \frac{p_\theta(\tau)}{p\theta(\tau) + q(\tau)} = \frac{1/z * e^{R_\theta}}{1/z * e^{R_\theta} + q(\tau)} \tag{6}$$

$$thus,\ L_{dis}(\theta) = E_{\tau \sim p}[-logD_\theta(\tau)] + E_{\tau \sim q}[-log(1 - D_\theta(\tau))] \tag{7}$$

Where, $p(\tau)$ is the data distribution and $q(\tau)$ is the policy distribution. But the paper contains no experimental proof. In AIRL Fu et al. [2018], it shows GAN-GCL suffer from variance issue as it considers a whole trajectory to compute discriminator update. Instead AIRL computes discriminator update for each (s,a,s') tuple

$$D_{\theta,\phi} = \frac{p_{\theta,\phi}(s,a,s')}{p_{\theta,\phi}(s,a,s') + q(s,a)} \tag{8}$$

$$= \frac{e^{f_{\theta,\phi}(s,a,s')}}{e^{f_{\theta,\phi}(s,a,s')} + q(s,a)} \tag{9}$$

$$= \frac{e^{f_{\theta,\phi}(s,a,s')}}{e^{f_{\theta,\phi}(s,a,s')} + \pi(a|s)} \tag{10}$$

where reward function, $f_{\theta,\phi}(s,a,s') = r_\theta(s,a) + \gamma\Phi_\phi(s') - \Phi_\phi(s')$ computes distangled reward Fu et al. [2018] Ng et al. [1999]. $r_\theta(s,a)$ is the reward network and Φ_ϕ reward shaping term controlled by the dynamics.

2.1 Discriminator Actor Critic (DAC)

Most widely popular imitation learning algorithm Generative Adversarial Imitation Learning (GAIL)Ho and Ermon [2016] has a major problem. The algorithm has low sample efficiency, requiring a very large number of environmental interactions in order to reach convergence. This is undesirable, since interactions can be expensive. To mitigate this issue, DAC utilizes the TD3 off-policy learning algorithm Fujimoto et al. [2018] to train the policy; this significantly increases sample efficiency.

DAC is an adversarial imitation learning algorithm. It uses an off-policy reinforcement learning algorithm to improve upon the sample efficiency of existing methods, and it extends the learning environment with absorbing states and uses a new reward function in order to achieve unbiased rewards and converts all terminal states in the environment to absorbing states. Combined, these changes remove the survival bias.

2.2 Adversarial Inverse Reinforcement Learning (AIRL)

Adversarial inverse reinforcement learning (AIRL), Fu et al. [2018] an inverse reinforcement learning algorithm based on adversarial learning. It provides for simultaneous learning of the reward function and value function, which enables us to both make use of the efficient adversarial formulation and recover a generalizable and portable reward function, in contrast to imitation learning. Even though performance of GAIL dominates over AIRL, it's valuable where there is considerable variability in the environment from the demonstration setting, GAIL and other IRL methods fail to generalize. It learn distangled reward function that's independent of the change in underlying dynamics.

3 Environment Setup: Dynamic Environment

Ultimate goal of the project is to see the performance of IRL algorithm when dynamics of the environment is changed. In IRL algorithm as we learn the underlying reward function, it's possible to retrain the agent for changed dynamics.

For experiments I have used MuJoCo Todorov et al. [2012] Ant-v2 environment. And to see performance under changed dynamics I have (1) added Gaussian noise (2) added TD3 exploration

noise (3) Crippled back legs: by setting fixed value to actions for back legs.
For a well trained policy in RL setting, under the changed dynamics I found a obvious performance drop.

Policy	TD3	SAC
Crippled leg	-2322847	-2322884
TD Noise	1941.51	977
Gaussian Noise	660.38	854
Actual Performance	3721	4482

4 Experimental Details

To get the expert data, I have trained SAC algorithm in RL setting where actual reward is available and used 4 trajectory (4000 $s, a, log\pi(a|s)$ tuples) data of the trained policy as expert data.
Reward network and value network both are 2-layer neural network(NN) with 32 hidden units. I have experimented on larger networks as well where reward consists of 2-layer NN with 256 hidden units and value with 100 hidden units (same architecture as DAC discriminator) but couldn't find much difference in performance. Yet more experiments are required to say anything concrete, which wasn't possible to do in the project.

4.1 Selection of Policy

Only difference in imitation learning and IRL is the way discriminator is computed. So I have used the algorithm described in DAC and replaced it's discriminator network with AIRL discriminator. But for to compute $D_{\theta,\phi}$ it requires to compute likelihood of action $\log \pi(a|s)$. So instead of deterministic policy TD3, I'm using SAC which is stochastic.
I have evaluated a performance of both policies in both RL and Imitation settings and found SAC to be more efficient while was able to reduce variance during imitation.

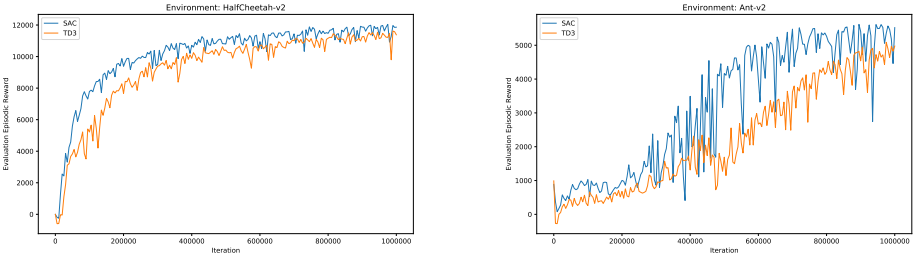

Figure 1: Performance comparison SAC and TD3 in RL setting

In both RL and Imitation setting SAC seem to perform better than TD3 along with reducing variance.

4.2 Discriminator update rule

Computed discriminator loss following Fu [2018]

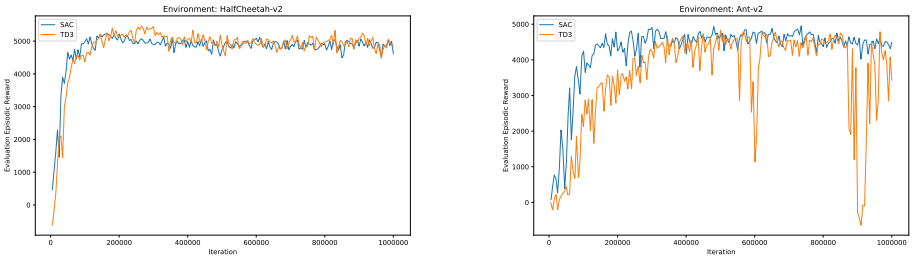

Figure 2: Performance comparison SAC and TD3 in Imitation setting

$$D_{\theta,\phi} = \frac{e^{f_{\theta,\phi}(s,a,s')}}{e^{f_{\theta,\phi}(s,a,s')} + \pi(a|s)} \tag{11}$$

And objective of the discriminator is to minimize binary cross entropy loss

$$loss = -(label) * \log D_{\theta,\phi}(s,a,s') - (1 - label) * \log(1 - D_{\theta,\phi}(s,a,s')) \tag{12}$$

4.3 Policy update rule

SAC being an off-policy algorithm, while training it's network by sampling $(s, a, s', done)$ from buffer I followed the AIRL policy update rule and computed reward $\hat{r}(s, a, s')$

$$\hat{r}_{\theta,\phi}(s,a,s') = \log D_{\theta,\phi}(s,a,s') - \log(1 - D_{\theta,\phi}(s,a,s')) \tag{13}$$

$$= \log \frac{e^{f_{\theta,\phi}(s,a,s')}}{e^{f_{\theta,\phi}(s,a,s')} + \pi(a|s)} - \log \frac{\pi(a|s)}{e^{f_{\theta,\phi}(s,a,s')} + \pi(a|s)} \tag{14}$$

$$= f_{\theta,\phi}(s,a,s') - \log \pi(a|s) \tag{15}$$

thus it becomes a entropy regularized policy objective and $f_{\theta,\phi}$ is the reward function.
The distangled reward function used in AIRL

$$f_{\theta,\phi} = r_\theta(s,a) + \gamma\Phi_\phi(s') - \Phi_\phi(s) \tag{16}$$

$$= r_\theta(s,a) + \gamma\Phi_\phi(s'|T(s'|s,a)) - \Phi_\phi(s) \tag{17}$$

and Φ can be any function that gives a measure of being at any state s and the equation makes reward function independent to the change in dynamics Ng et al. [1999]. In AIRL author uses value function of the states to measure this additional shaping terms, $r_\theta(s,a) + \gamma V(s') - V(s)$. I have used policy and discriminator update rules same as DAC while optimizing the objective of AIRL.

4.4 Implementation 1

For first implementation, I have used the exact loss computation as AIRL. It doesn't use any squashing function at the output of the discriminator $D_{\theta,\phi}$. But in my implementation this makes the learning unstable and computes "Nan" at the policy.

4.5 Implementation 2

To resolve the issue I have bounded the output of discriminator with sigmoid squashing function for all following experiments and it seems to solve the instability issue. For the implementation-2

- policy update using : $\hat{r}_{\theta,\phi}(s,a,s') \rightarrow \log D_{\theta,\phi}(s,a,s') - \log(1 - D_{\theta,\phi}(s,a,s'))$
- distangled reward function : $r_\theta(s,a) + \gamma V_\phi(s') - V_\phi(s)$

As discussed in section 4.3 reward computed to update policy is equivalent to entropy regularized reward function.

4.6 Implementation 3

As reward shaping term could be any representation of state (discussed in section 4.3), I have experimented computing value function of latent variable of the state as reward shaping term.
I have trained a VAE on the expert trajectory to learn a latent representation of the state expert visits and computed value of latent representation of states in the reward function $r_\theta(s,a) + \gamma V(z') - V(z)$. Used the code from PyTorch [2019] example of VAE implementation to learn latent representation.

- policy update using : $\hat{r}_{\theta,\phi}(s,a,s') \leftarrow \log D_{\theta,\phi}(s,a,s') - \log(1 - D_{\theta,\phi}(s,a,s'))$
- distangled reward function : $r_\theta(s,a) + \gamma V_\phi(z') - V_\phi(z)$

4.7 Implementation 4

In implementation 4, I have directly computed reward from the reward network to compute policy update with entropy as exploration bonus. Here entropy is not bounded by $(0,1)$, while generator takes a exploratory action, due to not having much previous information it gets high entropy bonus and this entropy bonus get's reduced for most of the (s,a) once policy have some understanding of the world.

- policy update using : $\hat{r}_{\theta,\phi}(s,a,s') \leftarrow r_\theta(s,a) + H(\pi)$
- distangled reward function : $r_\theta(s,a) + \gamma V_\phi(z') - V_\phi(z)$

5 Evaluate Performance

5.1 Performance Comparison

Comparing all the experiments, Implementation-2 gives the best performance in the Ant-v2 environment. Even though it doesn't perform as good as imitation, it's advantages of reward

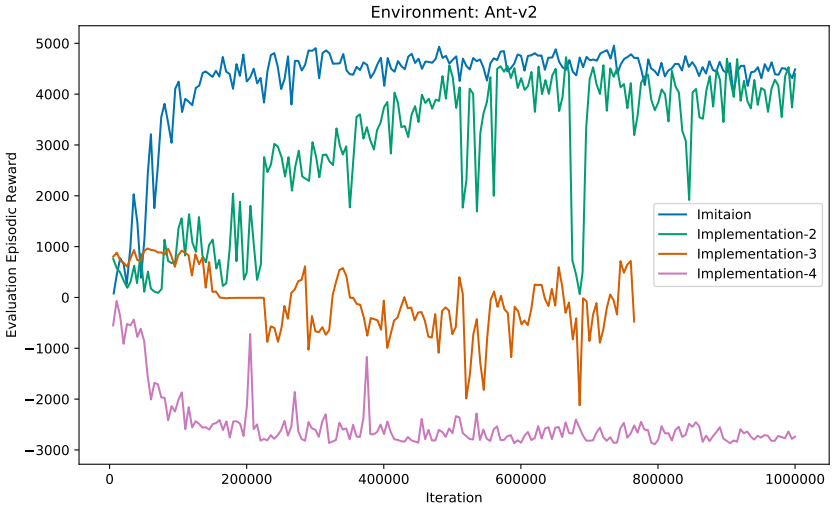

Figure 3: Performance comparison Imitation and IRL

learning comes into play in transfer learning when dynamics of the environment is changed. Using the learned reward function we can re-learn policy which is not possible in imitation.

5.2 Evaluate Learned Reward function

Before trying AIRL learned reward function in dynamic environment or transfer learning setting, I tried to evaluate it on RL setting. So I have trained a policy using the learned reward network $r_\theta(s, a)$ from Implementation-2. But it doesn't seem to learn anything.

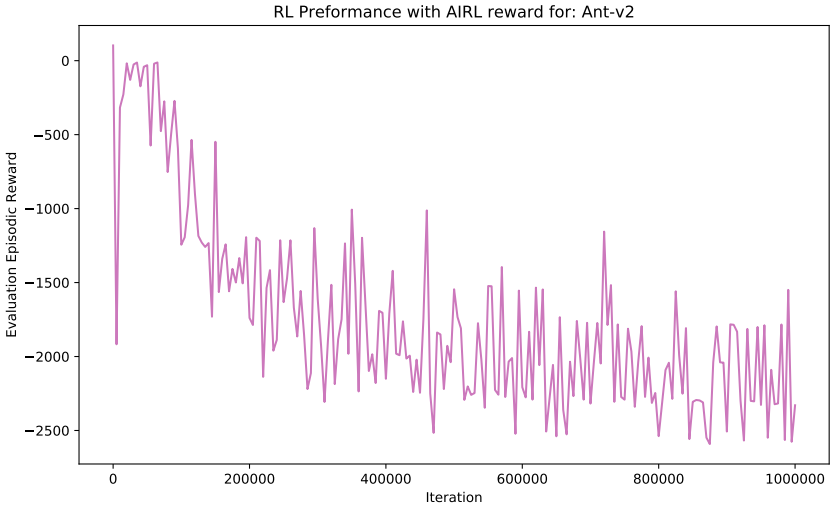

Figure 4: Performance using learned reward function from IRL in RL setting

As I'm submitting the report I'm still running the code to see the performance if used $r_{\hat{\theta},\phi}$ instead.

5.3 Performance in changing dynamics

As the learned reward function was not good enough to train policy in RL it becomes obvious it's not actually a good reward approximation of the environment. So, while training in the changed dynamics, the policy couldn't learn adaptive policy for the changed dynamic.

6 Conclusion

Even though I was able to get almost expert-like performance with slight variance using implementation-2 the learned reward function was proven not to be useful in learning policy at neither RL setting nor dynamic environments. There are various reasons why it fails:

- As I'm using sigmoid at the outer layer of the discriminator that could be one reason for learning a bad reward function
- AIRL algorithm uses TRPO Schulman et al. [2015] as it's policy, which does online learning. For each trajectory it updates it's policy and discriminator/reward function. On contrary, I'm using SAC (an off-policy actor-critic), so the discriminator update rule may not be suitable in this case.
- AIRL uses 50 expert trajectory while I took 4. More expert data maybe helpful to learn underlying dynamics, which wasn't possible to do within the submission deadline.
- AIRL get's it's best performance when makes the reward function only dependent on state , $r_\theta(s)$. I couldn't do that experiment within the submission deadline.

References

Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=Hk4fpoA5Km`.

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rkHywl-A-`.

Scott Fujimoto, Herke van Hoof, and Dave Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018. URL `http://arxiv.org/abs/1802.09477`.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL `http://arxiv.org/abs/1801.01290`.

Brian D Ziebart, J Andrew Bagnell, and Anind K Dey. Modeling interaction via the principle of maximum causal entropy. 2010.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016. URL http://arxiv.org/abs/1606.03476.

Chelsea Finn, Paul F. Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *CoRR*, abs/1611.03852, 2016. URL http://arxiv.org/abs/1611.03852.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2. URL http://dl.acm.org/citation.cfm?id=645528.657613.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

Justin Fu. Airl implementation. https://github.com/SaminYeasar/inverse_rl/blob/master/inverse_rl/models/airl_state.py, 2018.

PyTorch. exmples/vae. https://github.com/pytorch/examples/tree/master/vae, 2019.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.