[Re] Goal-conditioned Imitation Learning

Austin Sumigray, Soma A. Hota, Rashi Dhar
Department of Computer Science, Brown University, Providence, RI 02906
`austin_sumigray@brown.edu, soma_arunkanti_hota@brown.edu, rashi_dhar@brown.edu`

Abstract

Often the desired behaviour of an agent can be represented by means of a reward function in a specific state action space. However, creating a reward function by hand that is effective with multiple goals is often extremely time-consuming. Techniques like Hindsight Experience Replay (HER) have demonstrated how an agent is able to learn policies able to reach many goals, without the need of a reward, while Generative Adversarial Imitation Learning (GAIL) is able to learn more quickly, but is limited by the capability of the demonstrator. goalGAIL combines the two in order to perform sample-efficiently, but still allow for surpassing a demonstration.

1 Introduction

Imitation Learning is supervised learning applied to the Reinforcement Learning setting. Here learning from expert trajectories can be accomplished in two different ways:

Behavioral Cloning:

(i) It a supervised learning problem that maps state/action pairs to policy
(ii) Requires a large number of expert trajectories
(iii) Copies actions exactly, even if they are of no importance to the task.

Inverse Reinforcement Learning(IRL):

(i) Here, the agent learns the reward function from expert trajectories, then derives the optimal policy
(ii) Expensive to run
(iii) Indirectly learns optimal policy from the reward function.

The portion of goalGAIL we replicated doesn't use behavioral cloning, and instead uses GAIL to learn a policy directly in a way that is quicker than using IRL to learn a cost function and then RL to extract a policy. In addition, they make a modification to GAIL in order to allow it to perform in goal-described environments. We have used two baselines: HER and GAIL on top of which we implement the proposed method goalGAIL in the paper. goalGAIL uses a powerful data augmentation strategy that extracts more information from the demonstrations. This is useful for situations when we have few demonstrations. The demonstrations are broken down into separate valid trajectories to reach the states within the demonstration.

Hindsight Experience Replay (HER): Reinforcement Learning has led to many successes in the field recently in learning a policy for sequential decision-making problems. However a large problem with many algorithms is the need to generate a reward suited to the problem at hand. The reward needs to be indicative of success towards the larger problem, however must also not lead towards incorrect or in-optimal solutions. In addition, when a goal is changed the rewards allocated must change also. This leads reward-shaping to be a very time-heavy problem. One obvious solution is to find some way to automatically generate the reward, or come up with some algorithm that can learn well from unshaped rewards.

Behavioral Cloning is one way to avoid having to learn a reward. By generating an expert trajectory, state action pair demonstrations can be kept in a dataset. Once these demonstrations are collected, they can then be used to run a supervised learning algorithm on them. Thus, using regression an agent can be trained to mimic the expert. In fact, you can condition the cloning on the difference between the action chosen in a given situation, versus the action that the expert would have chosen. Then, by repeatedly showing the recorded state-action pairs to the agent, you can shape the agent to make the same decisions. You can even add to the dataset through techniques such as HER above in order to be more sample-efficient.

However behavioral cloning, while easy to understand has some major problems. Since the policy is based on what the expert would have done from every given situation, increasingly more expert demonstrations are needed. This makes it very data hungry as the state dimension increases. In addition, variance will increase quickly as the states currently seen and actions required diverge from what the agent has seen demonstrated. Inverse reinforcement learning (IRL), on the other hand, learns a cost function that prioritizes entire trajectories over others, so compounding error, a problem for methods that fit single-timestep decisions, is not an issue. Since it is able to learn what an expert "values" through learning their cost function or rewards, it is better able to extrapolate for unseen exact situations.

HER also allows the algorithm to perform exactly this kind of reasoning and can be combined with any off-policy RL algorithm.[2] It is applicable whenever there are multiple goals which can be achieved, e.g. achieving each state of the system may be treated as a separate goal. Not only does HER improve the sample efficiency in this setting, but more importantly, it makes learning possible even if the reward signal is sparse and binary. The primary idea behind HER is to replay each episode with a different goal than the one the agent was trying to achieve, e.g. one of the goals which was achieved in the episode. Then, instead of a single negative reward applied to a single goal which was the objective of the exploration policy, information can be acquired about how to reach other locations. The intuition is that although a given trajectory may have done a bad job at leading you to one location, it was a good way of leading to a different one. This can be shown in that it did, in fact lead to that location. In addition, changing these goals, since the reward is relative to the changed goal, will not change the optimal policies generated. [4]

Algorithm Overview:

Our implementation of goalGAIL following the paper can be largely broken up into four sections. The core pieces involved in the experiment we replicated are HER, GAIL, DDPG, and the changes that were made to these algorithms in order to allow them to function together. This last part is where the new information from the goalGAIL paper lies.

HER is used in order to provide more sample-efficiency to the algorithm. The core idea behind HER is that when a goal is the objective, and a sample episode is generated, you are left with whatever reward that sample obtained. If the original reward is very sparse and it was not seen in the sample, you are left with very little information gained. However, your sample does contain information which would otherwise be wasted. By treating pieces of the trajectory as though they were in fact the original goal, a constant negative cost can instead be turned into positive indicators. This leads an agent to have an easier time traversing a state space, especially if the goal might change in the future. For instance, if an agent is attempting to reach goal A at location A but instead reaches location B, under normal circumstances each transition would only indicate that it is a poor way to lead to location A. However HER can be used to relabel the rewards and goals such that instead of learning one poor way to reach location A, you instead learn information about how well the transitions did at leading you to the place the agent ended up reaching.
This is especially useful in situations like robotics, where a robot might be expected to learn how to push or place an object in a complex space. Policies may be learned that will allow the robot to learn

how to manipulate objects within the space with a fraction of the trajectories that would be required if a new policy was required to be trained for each set of starting and ending locations.

Generative Adversarial Imitation Learning(GAIL): GAIL is not exactly Inverse Reinforcement Learning because it learns the policy and not the reward function, directly from data. [5] Yet, it is better than Behavioural Cloning and able to be near to or sometimes slight better than an expert, because it is not constrained to be always exactly next to the expert. It is similar to Generative Adversarial Networks and consists of two main function approximation(networks), one for the policy π other for Discriminator D which fit the parameterized policy π_θ, with weights θ and discriminator network D_w with weights w. The Policy (Generator) network π_θ is trained originally using a trust region policy optimization (TRPO) and the discriminator network D is a supervised learning problem trained with an ADAM gradient step on expert trajectories. However recent work shows that it works with an off-policy training algorithm as well which will come in useful later on.

Steps to train GAIL:

i. Sample expert trajectories
ii. Optimize policy π_θ
iii. Optimize Discriminator D

In GAIL [2], a discriminator D$_\psi$ is trained to distinguish expert transitions (s, a) $\sim$ τ_{expert} from agent transitions (s, a) $\sim$ τ_{agent}, while the agent is trained to "fool" the discriminator into thinking the agent is the expert. Formally, the discriminator is trained to minimize $\mathcal{L}_{GAIL}$ = $\mathbb{E}_{(s,a)\sim\tau_{agent}[logD_\psi(s,a)]+\mathbb{E}_{(s,a)\sim\tau_{expert}[log(1-D_\psi(s,a)]}}$; while the agent is trained to maximize $\mathbb{E}_{(s,a)\sim\tau_{agent}[logD_\psi(s,a)]}$ by using the output of the discriminator $logD_\psi(s,a)$ as reward.

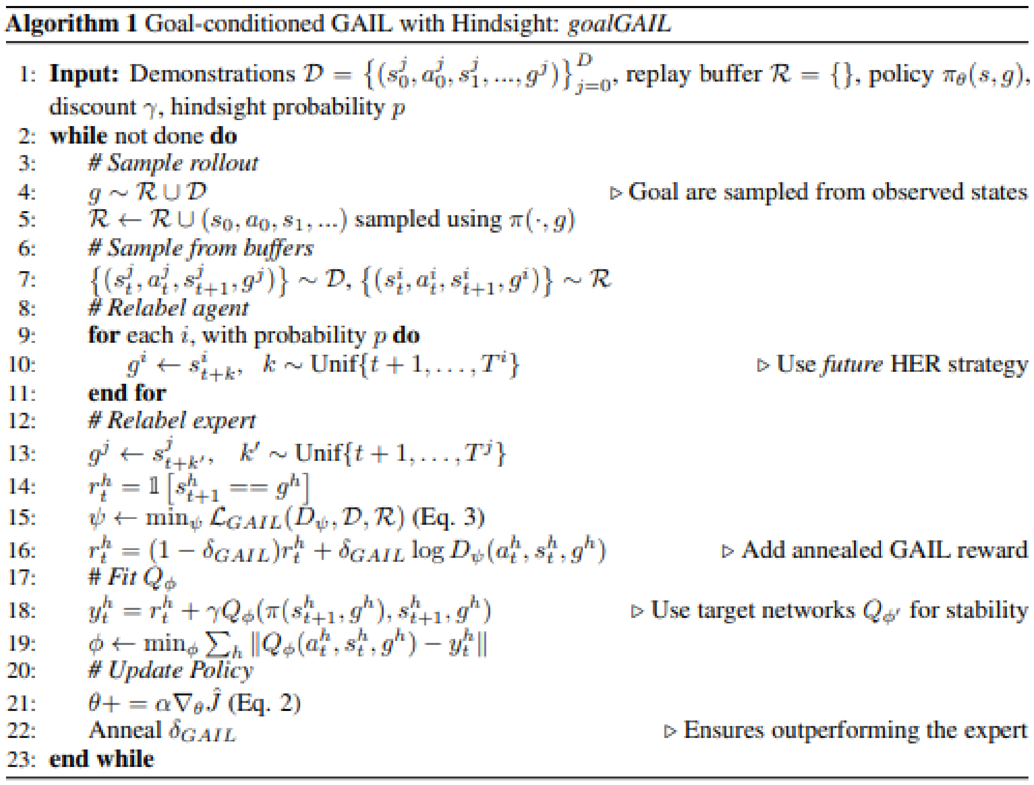
Deep Deterministic Policy Gradients (DDPG): The easiest way to explain DDPG is by breaking down the name. The Deep portion is fairly simple: it uses a deep network. It actually uses a couple of deep networks, since it uses a replay buffer to store state, action, reward, and terminal information acquired from a main network. The addition of the replay buffer makes this an off-policy method, since the exploration is done on a different policy than the training. This allows for stabilizing training. T[3]he Deterministic part comes into play with the policy itself, which is designed for continuous spaces and is deterministic. Since it is deterministic, in order to explore some noise is added to the exploration policy. The second network then comes into play and is referred to as a target network. Each network both outputs a Q value which is the value of choosing the most optimal action, as well as a policy action itself. Then the gradients come into play as the Q value is updated by using gradient descent (assuming that the Q function is differentiable with respect to action since the action space is continuous) to update the Q value of the current policy by comparing the target q value of the target's policy to the current. Then the policy is updated by using gradient ascent to push the current policy in the direction of the highest Q value.

Goal-oriented Generative Adversarial Imitation Learning (goalGAIL): This is the proposed method which merges the two baselines: HER and GAIL. goalGAIL is an algorithm that uses demonstrations to speed up agents learning on goal-conditioned tasks. The updates to the algorithms above are in making both GAIL and DDPG work with goal-oriented demonstrations. In addition, rather than simply using relabeling on the original collected trajectories like in HER, the expert trajectories are also relabeled. This is done by breaking down each trajectory into a series of steps. The intuition is that if a trajectory shows that the expert way of reaching C from A is A -> B -> C, then this trajectory also shows that the expert way of reaching B is from A to B. This allows a small amount of expert trajectories to be increased into a larger amount. Finally, the authors of the original paper hook GAIL into DDPG by having there be two rewards passes into DDPG. There is the original reward gained from the trajectories. In addition there is a reward based on how well the new trajectory fools the GAIL discriminator into thinking it is an expert trajectory. The

GAIL portion of the algorithm leverages additional rollouts collected by the learning agent in a self-supervised manner. This greatly speeds up initial learning. The expert trajectory generation is combined with HER in order to perform more sample efficiently and generalize to reaching all goals. The discriminator is conditioned on the goal $D_\psi(a, s, g)$ and is trained by minimizing the loss:

$$\mathcal{L}_{GAIL}(\mathcal{D}_\psi, \mathcal{D}, \mathcal{R}) = \mathbb{E}_{(s,a,g)\sim\mathcal{R}}[logD_\psi(s, a, g)] \\ + \mathrm{E}_{(s,a,g)\sim\mathcal{D}}[log(1 - D_\psi(s, a, g)]$$

Finally, the reward portion from GAIL is annealed, which allows it to outperform the demonstrator. Since GAIL is unable to do much better than an expert (since the goal is to fool the discriminator as to whether or not it itself is the expert, and outperforming the expert would cause it to look differently), the loss is slowly reduced, until the only reward received is from reaching the goal. $[logD_\psi(s, a)]$



Algorithm 1 Goal-conditioned GAIL with Hindsight: *goalGAIL*

```
1: Input: Demonstrations $\mathcal{D} = \{(s_0^j, a_0^j, s_1^j, ..., g^j)\}_{j=0}^{D}$, replay buffer $\mathcal{R} = \{\}$, policy $\pi_\theta(s, g)$,
   discount $\gamma$, hindsight probability $p$
2: while not done do
3:     # Sample rollout
4:     $g \sim \mathcal{R} \cup \mathcal{D}$                                  ▷ Goal are sampled from observed states
5:     $\mathcal{R} \leftarrow \mathcal{R} \cup (s_0, a_0, s_1, ...)$ sampled using $\pi(\cdot, g)$
6:     # Sample from buffers
7:     $\{(s_t^j, a_t^j, s_{t+1}^j, g^j)\} \sim \mathcal{D}$, $\{(s_t^i, a_t^i, s_{t+1}^i, g^i)\} \sim \mathcal{R}$
8:     # Relabel agent
9:     for each $i$, with probability $p$ do
10:        $g^i \leftarrow s_{t+k}^i,\ \ k \sim \mathrm{Unif}\{t+1, \dots, T^i\}$           ▷ Use future HER strategy
11:    end for
12:    # Relabel expert
13:    $g^j \leftarrow s_{t+k'}^j,\ \ k' \sim \mathrm{Unif}\{t+1, \dots, T^j\}$
14:    $r_t^h = \mathbb{1}\left[s_{t+1}^h == g^h\right]$
15:    $\psi \leftarrow \min_\psi \mathcal{L}_{GAIL}(D_\psi, \mathcal{D}, \mathcal{R})$ (Eq. 3)
16:    $r_t^h = (1 - \delta_{GAIL})r_t^h + \delta_{GAIL} \log D_\psi(a_t^h, s_t^h, g^h)$         ▷ Add annealed GAIL reward
17:    # Fit $Q_\phi$
18:    $y_t^h = r_t^h + \gamma Q_\phi(\pi(s_{t+1}^h, g^h), s_{t+1}^h, g^h)$          ▷ Use target networks $Q_{\phi'}$ for stability
19:    $\phi \leftarrow \min_\phi \sum_h \|Q_\phi(a_t^h, s_t^h, g^h) - y_t^h\|$
20:    # Update Policy
21:    $\theta += \alpha \nabla_\theta \hat{J}$ (Eq. 2)
22:    Anneal $\delta_{GAIL}$                                    ▷ Ensures outperforming the expert
23: end while
```

Figure 1: goalGAIL Algorithm (from the original paper)

2 Method

Reproducibility of the original results
We reproduced the authors results on the Fetch Pick and Place environment for Figure 3[1] where they show that goalGAIL takes off and converges faster than HER and is able to outperform the demonstrator unlike standard GAIL. We used the paper description, and took the same hyperparameters as the original paper, unless stated otherwise below, or originally unspecified. Since HER, GAIL, and DDPG were based on pre-existing baselines, we referred to the baseline code referenced by the authors, as well as the Pick and Place OpenAI Gym environment. However to allow for replication to be meaningful we implemented each of them in Tensorflow 2 rather than the original Tensorflow environment the authors used. We also had some difficulty in making the original environment work, since these pieces were tightly coupled with RLLab which is sadly discontinued, and which required a wide variety of library versions which the yaml provided described for what we believe is Mac-based OS. We then merged them together using the algorithm described in the paper.

The exact training procedure is detailed below.

Trajectory Generation:
We generate expert trajectories and trajectories from the learned policy through a series of rollouts. The expert trajectories follow a constructed policy. These policies are then relabeled in order to provide additional trajectories for intermediate states. The learned trajectories follow the current exploration policy which is modified slightly in its actions by a random value. These trajectories are then plugged into HER which returns some relabeled trajectories with the goals and rewards shifted. These are then passed into the GAIL portion of the training algorithm. The loss computed here for the discriminator corresponding to the agent (the amount the discriminator believes that the agent is the expert) is then saved as a reward, which is then passed into the off-policy algorithm DDPG[6] along with the reward gained through original goal location as well as HER.

Working Algorithm (HER, GAIL, + DDPG):

```
I. Initialize DDPG
II. Initialize Replay Buffer $\mathbb{R}$
III. for each epoch (epoch=1):
    A. For each episode (episode=1):
        i. For the expert policy and the learned policy
            a. Sample a goal g and initial state $s_0$
            b. Sample an action $a_t$ using the behavioral policy from DDPG modified by random
               noise
            c. Execute the action $a_t$ and observe a new state $s_{t+1}$
            d. Set the reward gained
            e. Store the transition in replay buffer $\mathbb{R}$
        ii. For the expert policy, for a randomly chosen subset of transitions $T_e$
            a Modify the original goal of an episode to be the state achieved.
            b Modify the reward of the episode to be 1.
            c Add this transition to the replay buffer in addition to the original
        iii. For the learned policy, for a randomly chosen subset of transitions $T_l$
            a. Modify the original goal of an episode to be the state achieved.
            b. Modify the reward of the episode to be 1.
            c. Modify the original episode in the replay buffer
    B. Sample a small batch b from the replay buffer $\mathbb{R}$
    C. Perform one step of optimization using DDPG and b
        i. Run GAIL on an item from the expert as well as the learned policies.
        ii. Update GAIL using the GAIL loss to improve the discriminator.
        iii. Update each episode in the batch to include a reward from the GAIL loss.
    D. Test the newly trained policy
```

HER:
This takes in the replay buffer which is defined in run_experiment and ddpg, and returns the relabeled transitions. The expert relabeling as described above also takes place in this portion of the code in addition to the HER-relabeled agent transitions.

GAIL:
The original GAIL portion of the author's code was created by Google Research for Tensorflow 1. We were able to use the original GAIL paper as well as this code as a reference in order to re-implement GAIL in Tensorflow 2. The portion of the GAIL algorithm which is most important to the goalGAIL algorithm is the discriminator of the GAN. Essentially, this portion expects some expert trajectories, some policy input, and the discriminator itself. It then uses the policy to sample a trajectory. The Discriminator is then updated with a loss based on how unsure it is that the expert policy is the expert policy, along with how sure it is that the sampled trajectory is from the expert policy. Formally, this is $\mathcal{L}_{GAIL} = \mathbb{E}_{(s,a)\sim\tau_{agent}}[log D_\psi(s,a)] + \mathbb{E}_{(s,a)\sim\tau_{expert}[log(1-D_\psi(s,a)]}$

The policy is then updated to move itself along the gradient such that next time it would fool the discriminator better. For the implementation of goalGAIL we are replicating, this is done by using DDPG to generate the policy, and the loss for the GAN based on the agent as the reward.

The inputs for each side are the observation, goal, and action. The paper also demonstrates that it is viable to use the next state observation in place of the action, since the next state is usually information-rich enough to substitute for the action. However, the graph we were replicating did not do this. There is also an alpha which is a random number which determines how much an interleaved version of the inputs pulls from either the generated policy inputs or the expert. An output is then produced for the GAN and the expert which is how much the discriminator thinks each input belongs to the expert policy. It then generates a GAN_loss which is as described above. However it then determines an output for the interleaved version above, and computes a gradient for how each part of that interleaved input contributed for that output. It then uses this gradient to add to the loss obtained above. This is the portion which borrows from the WGAN way of doing things. This is to prevent any individual weights from getting too large. It's a penalty applied when any individual piece adds too much to the final result, which results in lowering that piece.

goalGAIL:
goalGAIL is a combination of the above techniques. We start by generating expert trajectories.Next, the current policy is queried to generate non-expert trajectories. These are modified to add a reward by querying the GAIL discriminator to see how effective they are in fooling the discriminator that they were generated by an expert. The second portion of the reward is generated based on the original goal, with the reward being 1 for reaching the goal and 0 otherwise. These are then augmented by HER in order to add more indicative rewards, by treating some intermediate states as the goal and rewarding accordingly. Treating the goals in this way allows even failed attempts to be indicative by saving them as successful attempts at reaching a different goal. This set of trajectories is then fed into the policy algorithm (in this case DDPG) in order to train a more effective policy. Next, the discriminator itself is trained according to the GAN loss above. Finally, the amount of the reward generated by GAIL is annealed, and then the cycle repeats after a test of the current quality of the policy. Annealing the GAIL reward allows for the policies generated to eventually outperform the expert.

3 Results

We reproduced the results of the Figure 3 from original paper for Fetch Pick and Place environment.

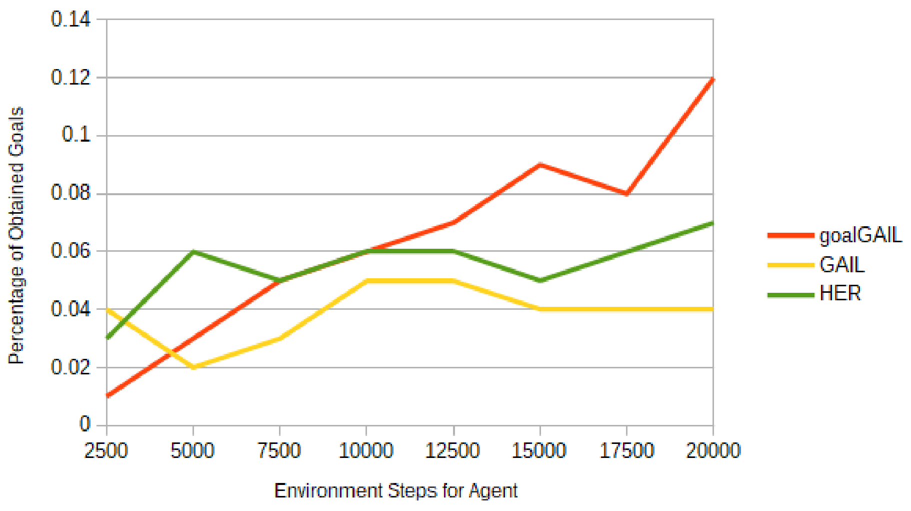


Figure 2: Results obtained from a goalGAIL policy, HER-only, and GAIL-only

With the time and resources available we were only able to run one session of 20,000 environment steps for each of the three agent configurations. This is clearly less than the original paper's graph, and insufficient to show the later stages of GAIL capping out on progress. However, the initial growth of goalGAIL is suitably strong even so. Similarly for the original chart for this environment, GAIL and HER are roughly similar, although HER does start with a lead over GAIL which is not seen in the original result. However this does show promise for the ability of goalGAIL to learn quickly.

4 Conclusion

Due to the lengthy time involved in training with the resources available, we were only able to run each algorithm to 20,000 timesteps. We interpreted the timesteps in the original paper as the number of times that the policy trained in the environment, however due to the expert batch size being smaller than the agents (through the efficiency of relabeling) this would only increase the number of steps by about a third. We originally achieved very poor results. Upon examining the code however, we modified how often our trajectories were being modified by HER and expert relabeling. However, we still did not achieve the same results for the baselines. GAIL appeared to have great difficulty improving its initial results, while HER was able to achieve a greater value fairly early on. We would have expected GAIL to outperform HER initially and for HER to slowly catch up, however this might have appeared farther on in training. Overall though, goalGAIL did perform more strongly than either of the baselines.

References

[1] Ding, Yiming, et al. "Goal-conditioned imitation learning." Advances in Neural Information Processing Systems. 2019.

[2] Andrychowicz, Marcin, et al. "Hindsight experience replay." Advances in Neural Information Processing Systems. 2017.

[3]H Vanseijen, R Sutton. "A Deeper Look at Planning as Learning from Replay." International conference on machine learning, 2015 - jmlr.org

[4] Ng, Andrew Y., Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping."

[5] Ho, Jonathan, and Stefano Ermon. "Generative adversarial imitation learning." Advances in neural information processing systems. 2016. ICML. Vol. 99. 1999.

[6] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).