Generative Adversarial Imitation Learning Benchmarking and Investigative Explorations

Ziyi Liu
Department of Computer Science and Technology
University of Cambridge
zl413@cam.ac.uk

Abstract

Generative Adversarial Imitation learning (GAIL), derived from inverse reinforcement learning (IRL), has shown both theoretic soundness and empirical prowess in the selected experiments from the original paper. We set out to reproduce some of the experimental results to reiterate the validity in part as well as to lay out the caveats in the original conclusions with detailed analysis, since we are unable to recover similar outcomes in some cases. An extension to novel environments was conducted to further test the capabilities of GAIL as opposed to the baseline methods. We also endeavored to improve GAIL performance by adjusting the regularization and architectural hyperparameters. With all the challenges and issues in mind from the experimental and exploratory work, we proposed several valuable research tracks for a more comprehensive future work, training resources permitting.

1 Introduction

The reward function is fundamental to reinforcement learning techniques, such that when agents act in accordance with some policy, the action will be rewarded or penalized based on how desirable the produced action is. This approach works well in relatively simple environments, where the reward function can be straightforwardly formulated towards the goal. However, in more complex settings where the agent with convoluted controls navigates through very complicated environments, such as autonomous driving [1], designing such a comprehensive reward function manually may become too laborious, if possible at all. However, in certain scenarios, obtaining expert demonstrations is much more intuitive and cost-effective, such as a human driving an automobile in the city or playing a multiplayer battle arena game.

Therefore, in imitation learning, we can bypass the challenge of manually defining a reward function by learning from the expert trajectories. There are several approaches towards this goal, and the most prominent ones include behavioral cloning [2], inverse reinforcement learning [3], and generative adversarial imitation learning [4].

Typically, behavioral cloning (BC) takes a "direct mapping" approach that trains the policy from the state to the control output, such that the problem can be viewed as a supervised regression problem [5]. Inverse reinforcement learning (IRL) [6], however, uses a model-based approach that first learns the reward function, based on which an outer loop of RL is applied to learn the "expert policy". And it has been shown that IRL methods have triumphed in many control tasks. Both methods are elaborated in section 2.

However, IRL does not scale well with comparatively large environments, as the RL-IRL loop quickly becomes very costly. The original GAIL paper [4], taking on the same theoretical basis as IRL, demonstrated that the full IRL can be reformulated as an occupancy measure matching problem

[4], and the task is now converted to looking for a policy that minimizes the difference between the occupancy measures of the learner and the expert.

In [4], GAIL was proposed as a generalization to the apprenticeship learning algorithms, as the latter bears a strong analogy to the occupancy measure matching problem, but is restricted to linear cost function classes. GAIL lifted this constraint with a new cost regularizer, where the cost function can be effectively viewed as the discriminator neural network in an adversarial context. With the policy parameterised by trainable weights θ, a generator-discriminator pipeline is established, with a strong resemblance to the original GAN [7].

The experiments in [4] showed that in most environments rendered by OpenAI Gym [8], GAIL performs significantly better than BC, Random, and other baseline methods; it is also capable of exceeding the expert policy performance in certain configurations, demonstrating its superiority as an imitation learning method.

However, as concise as [4] is, several issues have been left un-addressed, such as the analogy between GAN and GAIL. Furthermore, although 9 tasks have been selected, almost all of them are trained in the same environment MuJoCo [9] with similar control inputs, while the remaining few are also set in very simple environments.

Hence, the report will cover some key points as follows:

1. Providing further summaries and clarifications of certain theoretical topics not fully expounded in the original paper (Section 2)
2. Replicating some of the experiment results published in the original GAIL paper and expanding on the experimental procedures and challenges with analysis (Section 4)
3. Applying GAIL to novel tasks and analyze the empirical results (Section 5)
4. Benchmarking GAIL performances against baseline methods with extensive comparisons (Section 4, 5)
5. Proposing further research directions and experiments with some exploratory ground work (Section 6, 7)

2 Related Work

Behavioral Cloning Assuming the expert's demonstration is available throughout training, we can view this as solving an optimization problem[10]:

$$\hat{\pi}^* = argmin_\pi \sum_{\gamma \in \Gamma} \sum_{s \in \gamma} \mathcal{L}(\pi(s), \pi^*(s))$$

where π^* is the expert policy, and $\pi^*(s)$ the expert action at state s. The double summation suggests that we are covering all states in all collected trajectories $\gamma \in \Gamma$. This straightforward method, however, performs poorly as the environment becomes more complicated. Since the learning procedure relies solely on trajectories collected beforehand, it is most likely that expert demonstrations are not uniformly sampled across the state-action space. Thus, during inference, the BC model will not be able to generalize well in states unseen to the model in training, resulting in compounding errors.

DAgger DAgger is a variation of BC algorithms, where it trains the policy iteratively with an increasing amount of aggregated data [11]. The algorithm is summarised in fig. 2. The β parameter could control how much "expert data" as opposed to agent-explored trajectories, and using a geometric sequence for beta in iterations, the expert can be gradually phased out. Since the original work conducted all of its experiments on tasks that require image inputs, this paper will include a more diverse set of experiments to further benchmark the DAgger model.

Inverse Reinforcement Learning IRL can be seen as the reverse of RL process, where given state-action pairs (s_t, a_t), we want to infer the reward (cost) function $c(s, a)$. [4] adopted the procedures from maximum entropy IRL[3], the full procedure for which is shown in fig. 1. Using the γ-discounted causal entropy $H(\pi) = \mathbb{E}_\pi[-\log \pi(a|s)]$, the formalism of this optimization problem can be written as

$$max_{c\in\mathcal{C}}\Big(min_{\pi\in\Pi} - H(\pi) + \mathbb{E}_{\pi}[c(s,a)]\Big) - \mathbb{E}_{\pi_E}[c(s,a)],$$

which solves for an optimal cost function. The intuition behind this expression is that the cost function produces lower cost for expert policy π_E (the negative term on the right), and higher cost for other policies (positive expectation term). Upon obtaining the expert cost function, we can run a RL procedure with

$$RL(c) = argmin_{\pi\in\Pi} - H(\pi) + \mathbb{E}_{\pi}[c(s,a)]$$

that finds a policy which minimizes the cost function.

```
Algorithm 1 Maximum Entropy Deep IRL
Input: $\mu_D^a, f, S, A, T, \gamma$
Output: optimal weights $\theta^*$
 1: $\theta^1 = \text{initialise\_weights}()$
    Iterative model refinement
 2: for n = 1 : N do
 3:     $r^n = \text{nn\_forward}(f, \theta^n)$
        Solution of MDP with current reward
 4:     $\pi^n = \text{approx\_value\_iteration}(r^n, S, A, T, \gamma)$
 5:     $\mathbb{E}[\mu^n] = \text{propagate\_policy}(\pi^n, S, A, T)$
        Determine Maximum Entropy loss and gradients
 6:     $\mathcal{L}_D^n = \log(\pi^n) \times \mu_D^a$
 7:     $\frac{\partial \mathcal{L}_D^n}{\partial r^n} = \mu_D - \mathbb{E}[\mu^n]$
        Compute network gradients
 8:     $\frac{\partial \mathcal{L}_D^n}{\partial \theta_D^n} = \text{nn\_backprop}(f, \theta^n, \frac{\partial \mathcal{L}_D^n}{\partial r^n})$
 9:     $\theta^{n+1} = \text{update\_weights}(\theta^n, \frac{\partial \mathcal{L}_D^n}{\partial \theta_D^n})$
10: end for
```

Figure 1: IRL Algorithm

```
Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in $\Pi$.
for $i = 1$ to $N$ do
    Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
    Sample $T$-step trajectories using $\pi_i$.
    Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by $\pi_i$
    and actions given by expert.
    Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
    Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation.
```

Figure 2: DAgger Algorithm

GAN Generative Adversarial Network (GAN) [7] has been widely applied to generation tasks and produced promising results for many. It consists of two main components - The discriminator and the generator. The discriminator is a classifier that distinguishes between authentic data from those produced by the generator. With an example of image generation, the goal of the generator is to minimize the negative log-likelihood of the image generated so that it will resemble the training set images as much as possible, while that of the discriminator is to minimize the cross-entropy of classification to be better at differentiating the real images from generated images (see fig. 3). This approach, as illustrated later, provides a clear foundation for the origin and development of GAIL model.

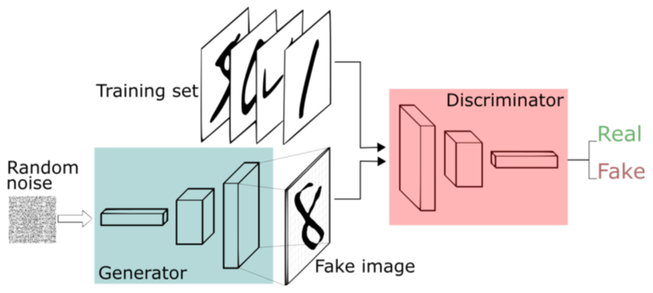


Figure 3: GAN architecture

GAIL The main subject of this report, GAIL, is founded on a seminal proposition:

$$RL \circ IRL_\psi(\pi_E) = \arg\min_{\pi \in \Pi} - H(\pi) + \psi^*(\rho_\pi - \rho_{\pi_E}),$$

where ρ denotes the occupancy measure, which is the distribution of state-action pairs the agent meets when exploring the environment with a policy. Therefore, the problem can be translated such that obtaining the optimal policy for RL-IRL approach is the same as matching ρ to ρ_E. This conversion is fundamental for the derivation of GAIL, as the ρ-matching problem can be then extended to the GAIL formulation with a change of cost regularizer function. The paper shows that for a linear class of cost functions $\mathcal{C}$ and cost regularizers $\delta_{\mathcal{C}}$, the ρ-matching problem $\min_\pi d_\psi(\rho_\pi, \rho_E) - H(\pi)$ can be written in the format of apprenticeship learning [12]:

$$\min_\pi -H(\pi) + \max_{c \in \mathcal{C}} \mathbb{E}_\pi[c(s,a)] - \mathbb{E}_{\pi_E}[c(s,a)].$$

However, with an empirically chosen cost regularizer, we can show the optimization expression now becomes

$$\min_\pi \max_D \mathbb{E}_\pi[\log(D(s,a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))] - \lambda H(\pi),$$

drawing a clear analogy with GAN. Specifically, the policy to learn π_θ is parameterised by θ and trained with a policy update algorithm, and it's used to generate the trajectories that the discriminator D needs to distinguish from the expert trajectories. The full algorithm is shown in fig. 4.

```
1: Input: Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters $\theta_0, w_0$
2: for $i = 0, 1, 2, \dots$ do
3:     Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4:     Update the discriminator parameters from $w_i$ to $w_{i+1}$ with the gradient
           $\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))]$    (17)
5:     Take a policy step from $\theta_i$ to $\theta_{i+1}$, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$.
       Specifically, take a KL-constrained natural gradient step with
           $\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta)$,
           where $Q(\bar{s}, \bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}]$    (18)
6: end for
```

Figure 4: GAIL algorithm

3 Experimental Setup

Since the original paper [4] provided no official codes nor detailed training configurations for its experiments, we adapted our imitation learning algorithms based on two existing repositories for all replicated and new experiments. The first codebase `PYTORCH-RL` is an implementation of GAIL in continuous context with different policy gradient methods (PPO [13], TRPO [14], and A2C), whereas the second repo `IMITATION` [15] implements many of the baseline methods in addition to GAIL used for benchmarking in this report as well.

All experiments in this report were conducted with the OpenAI gym framework [8]. In addition to reproducing a number of them in classical control and MuJoCo tasks used in the original paper, we also extended the evaluation to other simulations and scenarios. Notably, we also made use of Box2D environments (BipedalWalker, LunarLander) and Atari games [8]. The Box2D tasks are similar to MuJoCo tasks with comparable state space dimensions but more intuitive goals, while the Atari tasks are based on graphic inputs rather than well-defined state inputs.

For GAIL and most baseline methods, a three-step procedure is adopted to conduct full experiments, detailed below:

Step 1: Train an expert parameterised by a neural network with policy optimisation algorithms (PPO or TRPO in this report) with access to the intrinsic reward function of the environment

Step 2: Roll out a dataset of expert trajectories with the trained expert

Step 3: Train the learner with respect to the expert demonstrations from step 2

Since we have no information on the original hyperparameters used for training, we assume the default provided in the implementation for all replication experiments. Subsequently, we tune them for further exploratory experiments. In the `IMITATION` implementation, the policy generation network is borrowed from `StableBaseline` library [16], and the default policy net is a MLP with two linear layers of dimension 64 and activation function $tanh$, while the reward net (the "discriminator") has a similar structure but with dimension 32 in each layer and ReLU activations instead.

As extensive as this report purports to be, there are a number of limitations in the following evaluations. And these constraints will be analyzed and explained thoroughly with analysis towards the end of section 4.

4 Reproduction and Analysis of GAIL Results

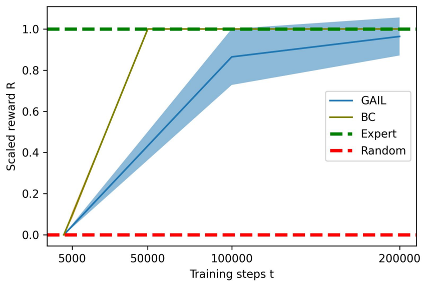
In the original paper, 3 classical control tasks and 6 MuJoCo tasks were provided for GAIL evaluation, but very little explanation was provided. Due to the limited resources, we reproduced a subset of these and laid out detailed analysis regarding training, evaluation performance, and benchmarking.

Classical Control Environments

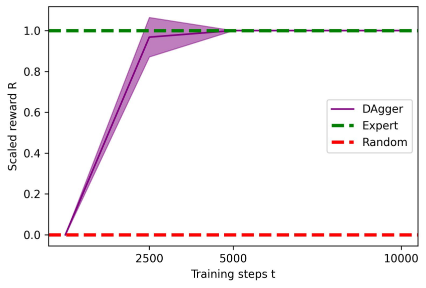
For classical tasks, we adapted `IMITATION` framework for the standard 3-step training for both GAIL and the two baseline methods. We set up our analysis with two environments - `CartPole` and `Acrobot`, where the benchmarking plots and environment examples are shown in figs. 5.

In both cases, we first train the expert with PPO to proficiency after 50000 timesteps. Since we also aim to explore whether imitation learning methods are capable of outperforming the expert, it is unnnecessary to train the expert with excessive resources, as long as the separation between the random baseline and the expert reward is sufficiently large. For example, although the Acrobot raw expert reward is only -90.5, it is still significantly and consistently higher than the random reward of -500.0, and therefore it is a valid expert policy. In all subsequent plots, we scale the trained policy rewards with respect to the expert(scaled-reward=1) and random policy(scaled-reward=0). We also benchmark GAIL performances against behavioral cloning and DAgger algorithms. However, as DAgger trains with an increasingly larger dataset acquired iteratively, the "timestep" measure does not effectively reflect the actual training the procedure undergoes. Therefore, we plot DAgger rewards separately where the "training step" is redefined as the number of distinct instances rather than the total number seen. Nevertheless, we can still draw reasonable comparisons between GAIL and DAgger by observing the learning curve and sampling efficiency.

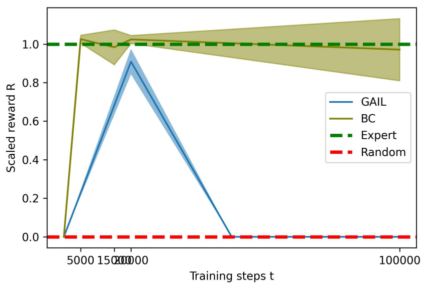
In the CartPole experiment, we can demonstrate the correctness of the full setup by observing the convergence to the highest possible reward in all three methods. However, unlike the original paper, BC approach quickly reached the expert proficiency, while GAIL took more than 4 times total training timesteps to achieve the same result. DAgger also attained the expert level very fast with around 5000 samples. We can, therefore, see that behavioral cloning-based methods are very efficient with low-complexity environments. Similarly, we can observe the same trend in the Acrobot-v1 experiment(fig. 5c, 5d). Despite some fluctuations after reaching expert reward, both BC methods slightly exceeded the expert very rapidly, while GAIL took twice as long. Furthermore, we can see



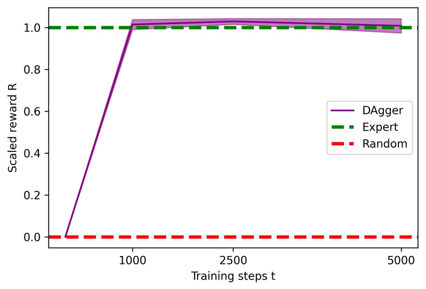
(a) CartPole GAIL performance benchmarks



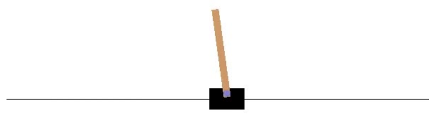
(b) CartPole DAgger performance benchmarks



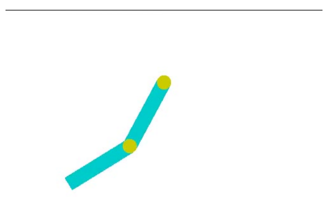
(c) Acrobot GAIL performance benchmarks



(d) Acrobot DAgger performance benchmarks



(e) CartPole environment illustration



(f) Acrobot environment illustration

Figure 5: Classical control tasks results

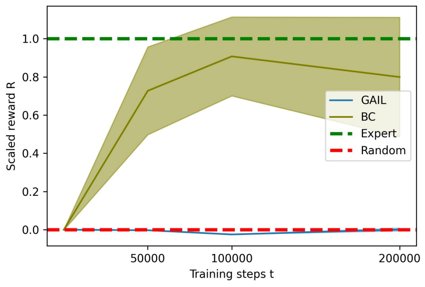
that our GAIL configuration is very unstable after reaching the expert level, where the test rewards quickly defaulted back to the random baseline. This will be a recurring theme in the later experiments as well, showing that the choice of hyperparameters in GAIL is particularly important.

Since both experiments only assume discrete action spaces, we then moved on to reproducing some results in MuJoCo environments with both continuous observation and action spaces for better generalisation.

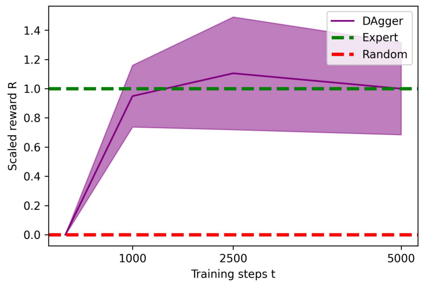
Mujoco Environments

Out of the 5 MuJoCo tasks outlined in the original paper, we selected `HalfCheetah` and `Hopper` to further test the efficacy of the GAIL algorithm (fig. 6). One extra experiment `Swimmer` was included as well.

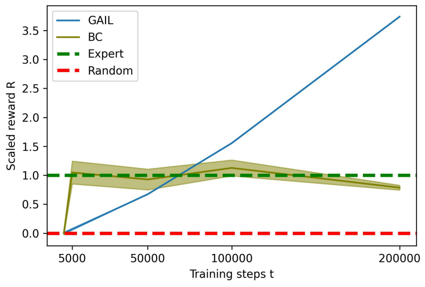
In the `HalfCheetah` experiment, GAIL did not manage to learn from the demonstrations at all, unlike BC which neared the expert. In the original paper, however, GAIL performance exceeded the expert by $\sim$5%. This disparity again shows that GAIL is potentially very sensitive to both the initial conditions and the training configurations. This was reiterated in the second experiment with `Hopper`,



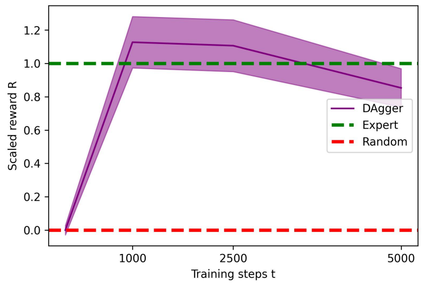
(a) HalfCheetah GAIL performance benchmarks



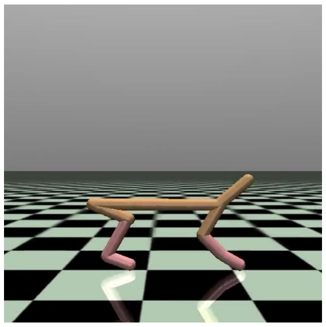
(b) HalfCheetah DAgger performance benchmarks



(c) Hopper GAIL performance benchmarks



(d) Hopper DAgger performance benchmarks


(e) HalfCheetah environment illustration

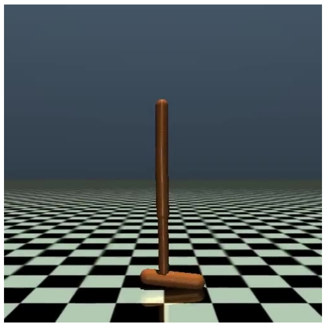
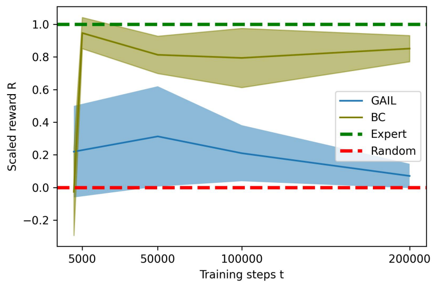

(f) Hopper environment illustration

Figure 6: Mujoco control tasks results

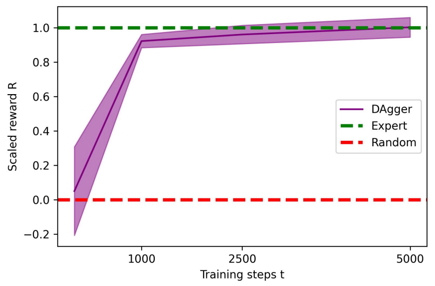
in which the final performance at timestep=200000 was around 3 times better than the expert result. One explanation for this immense boost in performance compared with the previous experiment is that the expert is not trained near the full proficiency, so that there is ample room for potential improvement, which was seen in GAIL, and this also exhibits the capability for GAIL to learn beyond the expert, as stated in [4]. In contrast, DAgger performed predictably well in both experiments, as the evaluation rewards exceeded the expert by $\sim$10% in both cases.

We then show the results for `Swimmer-v2` in figs. 7a, 7b. Although GAIL improved from the random baseline, its performance was markedly below that of both BC-based methods.

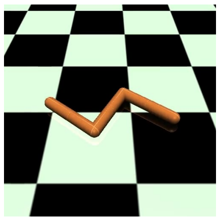
In summary, although in most cases we can show the ability of GAIL to learn from expert demonstrations, this report does not echo the results shown in the original paper in many sampled environments, where GAIL either under-performs or overshoots, comparatively. The baseline method BC, on the other hand, manifested better results than those given in [4] constantly. Surprisingly, DAgger, of all methods employed, topped almost all of the experiments insofar. Listed below are a few comments and explanations for the inconsistencies observed:



(a) Swimmer GAIL performance benchmarks



(b) Swimmer DAgger performance benchmarks



(c) Swimmer environment illustration

Figure 7: Extended Mujoco control tasks results

1. Framework difference: as [4] did not provide official codes or configurations, we had to conduct the experiments with adapted codebases and train with different sets of parameters. This alone may have a strong impact on the final results.
2. Version difference: since OpenAI Gym updated most of the environments used above, the older versions were not longer supported, and this inevitably leads to irregularities.
3. Expert Proficiency: as mentioned above, the experts are not trained till levelling off for many tasks, which could lead to significant disruption of learning curves, especially in the case of `HalfCheetah` and `Hopper`.
4. RAM limits: in some experiments, the number of trajectories sampled is much lower than that in original experiments due to insufficient RAM for expert transition storage.
5. Performance dips are seen in many plots after an extended period of training for both GAIL and BC & DAgger. The slow and gradual drops are possibly due to overfitting to the rollout training data, while the sharp decreases can be ascribed to the instabilities of PPO policy update algorithm near a local optimum. This can be further examined in future research by investigating the training reward feedback.

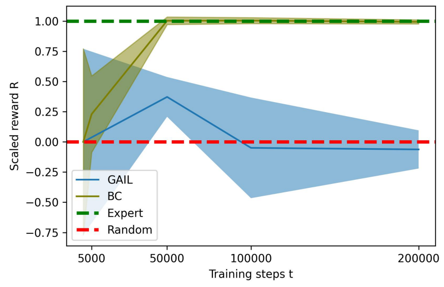
5 Extending GAIL to New Tasks

Following the attempts at recovering experimental results and accounting for the disparities, we then extended the current training and evaluation framework to completely new environments with an aim of displaying the versatility of GAIL and BC-based methods.

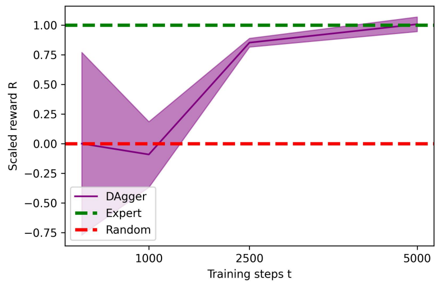
Box2D Environments

We selected `LunarLander-v2` and `BipedalWalker-v3` for Box2D environments. `LunarLander` results are obtained with the same setup described above, while `BipedalWalker` experimentation was carried out with the other framework `PYTORCH-RL`, the studies of which are investigated separately. `LunarLander` results are shown in fig. 8:

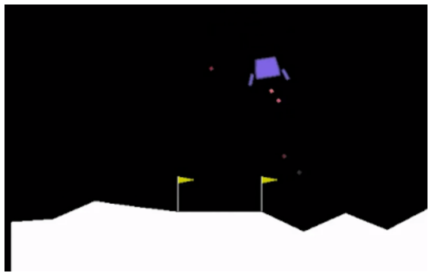
The results are very similar to what we saw in `Swimmer`, such that GAIL learned till a certain point and returned to random baseline, while the other methods converged to the expert level. In fact, both environments had an observation space of dimension 8, and thus the complexity is also similar.



(a) LunarLander GAIL performance benchmarks



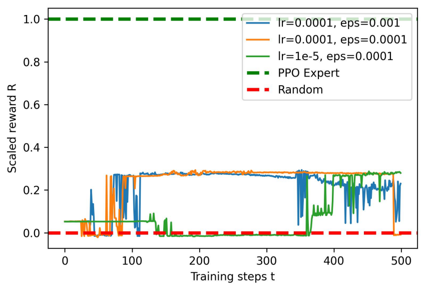
(b) LunarLander DAgger performance benchmarks



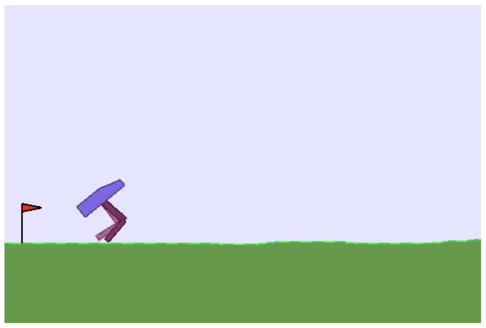
(c) LunarLander environment illustration

Figure 8: Box2D LunarLander control tasks results

For `BipedalWalker-v3`, we adopted `PYTORCH-RL` framework in order to obtain a good expert policy. In each iteration, we generated new samples with our agent, and trained the discriminator and generator respectively against the expert demonstrations. We plot the evaluation rewards vs. the number of iterations in fig. 9.



(a) LunarLander GAIL performance benchmarks



(b) BipedalWalker environment illustration

Figure 9: Box2D BipedalWalker control tasks results

For all three sets of hyperparameters, we can conclude that GAIL is indeed learning from the expert rollouts, yet none managed to come close to the expert. By watching the clip generated with our learned policy, we can see that very briefly after the initiation, the agent was stuck in a stable but stationary position without going forward. Hence, the agent is most likely trapped in a local minimum. Though changing learning-rate and clip-range did not help the agent escape the local minimum, we saw very different degrees of instability and learning curves, yet again showing that GAIL is very sensitive to the range of hyperparameters.

Atari Environments

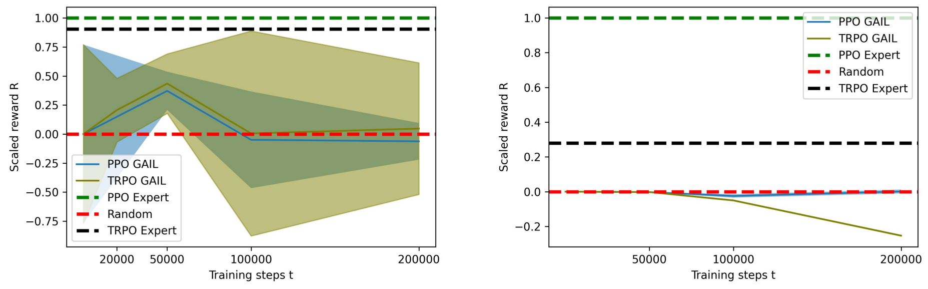
Finally, we attempted to extend the framework to Atari games, with an example of `Pong-v0`. As the MLP policy no longer works in step 1 of training the expert because the inputs are images, we switched to a CNN policy instead, and reached an average score of 3.3 after 200000 timesteps with the random baseline at -21.0, where the training time was 2.5h. The number of episodes was sharply reduced during rollout collection phase to avoid insufficient RAM error. However, despite the efforts in reducing the batch-size and training data size, a RAM overflow error was reported when we run the GAIL learning step. A few other experiments were also tried (`Breakout`, `Carnival`), but the same outcome ensued. This could, therefore, be an interesting topic to look into in future studies with better training resources.

Nevertheless, the Behavioral cloning algorithm did not crash during training, but the final result was only -20.5, slightly above the random policy. We therefore postulate that BC is no longer effective with much more complicated environments where the inputs are not simple independent states, as expected.

6 Improving GAIL

In this section, we adjusted a subset of GAIL hyperparameters and architectures as an endeavor to understand the reaction of the model to such tuning and ultimately look for means of improving GAIL under various contexts. Specifically, we chose environments in which GAIL performed rather poorly.

Firstly, a different policy gradient method, TRPO, was undertaken in place of PPO for `LunarLander` and `HalfCheetah` tasks, and the results are plotted in fig. 10. TRPO was the original approach used in [4], and it is benchmarked with the PPO method we have assumed in all experiments above. In both figures, we can show TRPO either resembled the PPO outcome closely but produced much higher variance or it showed even worse results than random policies in the long term.



(a) LunarLander GAIL performance with TRPO (b) HalfCheetah GAIL performance with TRPO

Figure 10: Replacing PPO with TRPO gradient method.

Secondly, since we have observed strong instabilities when the learned policy attained some certain reward level, we varied the clip-range ϵ which imposes a range of $[1-\epsilon, 1+\epsilon]$ on the action update probability ratio, acting as the regularization for range of exploration - the smaller ϵ is, the stronger the restriction of moving away from the current policy. As shown in fig. 11, GAIL failed to learn in all three ranges of clipping, but the original config of $\epsilon = 0.2$ manifested a higher degree of variability that led to a dip below the random baseline, while the other two got stuck closely to random policy. This indicates a strong potential utility of scheduling clip-range as the rewards gets higher to avoid the sudden crash of learning [17], which can be further investigated.

Finally, we explored the effect of policy architecture on reward results. We continued to use the FCNN architecture (originally 2 layers of dim 64 with tanh activation) but we modified the number of layers and activation function and obtained the results below (fig. 12). Other than our original implementation, all changes led to negative feedback after training, despite the early success in rising above random baseline. Hence, we show that MLP structure might be insufficient at capturing the expert actions in this particular case, such that simply scaling the network would not work at all. Further research in different architectures (e.g. RNNs) can help identify network features more conducive to particular tasks.

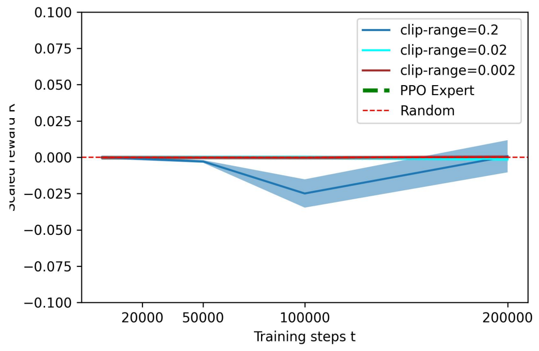


Figure 11: Varying clip-range ϵ for HalfCheetah GAIL training.

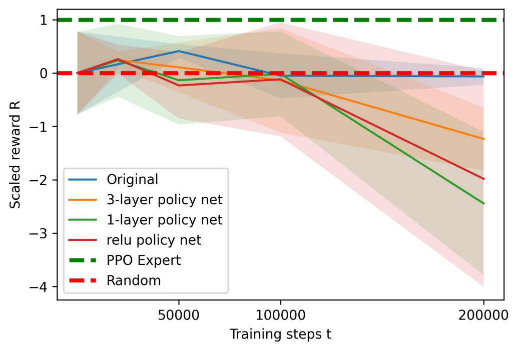


Figure 12: Varying architectural hyperparameters for LunarLander GAIL training.

7 Discussion and Future Work

Having encountered a few issues and intriguing phenomena in the experiments above, we thereby propose several inspiring directions for future research and contributions:

- Instead of training a genuine expert for every task, different levels of expertise can be examined and evaluated in terms of the final learning results with an objective of recovering the genuine expertise through imitation. Intuitively, the "expert" may potentially expedite the learning of the agent without the effort for obtaining a true expert a priori.
- As shown in most experiments above, GAIL is very sensitive to initial environment conditions and training hyperparameters, so that a framework of hyperparameter-tuning can be constructed for better ad hoc performance.
- In addition to TRPO and PPO, many other policy gradient methods more suitable to the specific tasks can be leveraged. For instance, the stable-baseline library [16] also provides implementations for A2C, ARS, Maskable PPO, etc., which can be experimented as the GAIL policy learner as well.
- Inspections of reward distribution and stability with respect to training time can be carried out in order to identify the relevant parameters that can lead to the most erratic behaviour. While we have already set up plots to visualise the reward distribution after training for a given number of timesteps (shown in Appendix A), a more granular approach can be utilised by visualising the actions taken by the policy for certain states during training and observe the cause for sudden dips and consistent drops in performance.
- As seen in DAgger experiments, the algorithm was evaluated on games with whole frames as the observation inputs, and we can certainly evaluate GAIL with the Atari tasks on open AI once the RAM limit is lifted with better hardware. These assessments can also testify

the superiority of GAIL in complex environments over BC methods claimed by the original paper [4].

- As expounded in section 6, substantial analysis on architectural improvement and regularizations can be done with a systematic methodology to pin down the optimal default networks and regularizations for GAIL training. Our tentative experiments by changing the size, activation, and regularization coefficient of the model have indicated strong response in evaluation rewards.
- One main advantage GAIL has over IRL methods is that model-free approaches saves the effort of retrieving a reward function and are thus much less costly. There are, nevertheless, other adversarial methods that combines the benefits in both worlds, such that the reward function can still be recovered using a framework resembling that of GAIL. AIRL[18] falls in this category, and future research can benchmark AIRL against GAIL and other baselines.

In conclusion, our empirical reproduction and extended work showed partial success in verifying some key ideas stated in the GAIL paper, while the inconsistent results led us to propose a variety of further experimentation plans for more systematic and comprehensive corroborations of the theoretic validity of GAIL.

Word Count: 3911

References

[1] Tanmay Vilas Samak, Chinmay Vilas Samak, and Sivanathan Kandhasamy. Robust behavioral cloning for autonomous vehicles using end-to-end imitation learning. *SAE International Journal of Connected and Automated Vehicles*, 4(3), Aug 2021.

[2] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4950–4957. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[3] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Deep inverse reinforcement learning. *CoRR*, abs/1507.04888, 2015.

[4] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016.

[5] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J. Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics*, 7(1–2):1–179, 2018.

[6] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, page 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[9] 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.

[10] Imitation learning lecture in principles of robot autonomy ii, February 2022.

[11] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010.

[12] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 1, New York, NY, USA, 2004. Association for Computing Machinery.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[14] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[15] Steven Wang, Sam Toyer, Adam Gleave, and Scott Emmons. The `imitation` library for imitation learning and inverse reinforcement learning. https://github.com/HumanCompatibleAI/imitation, 2020.

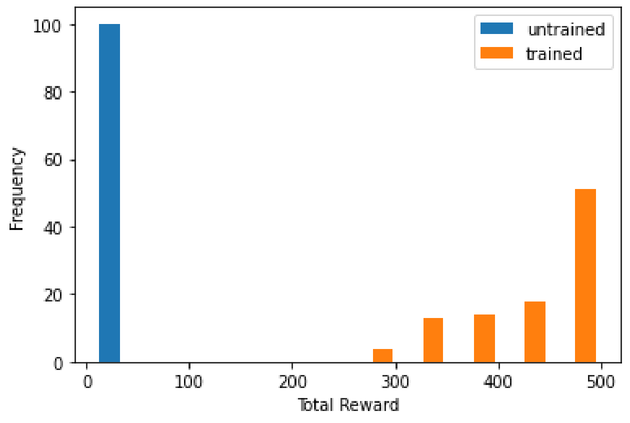
[16] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[17] Monika Farsang and Luca Szegletes. Decaying clipping range in proximal policy optimization. *2021 IEEE 15th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, May 2021.

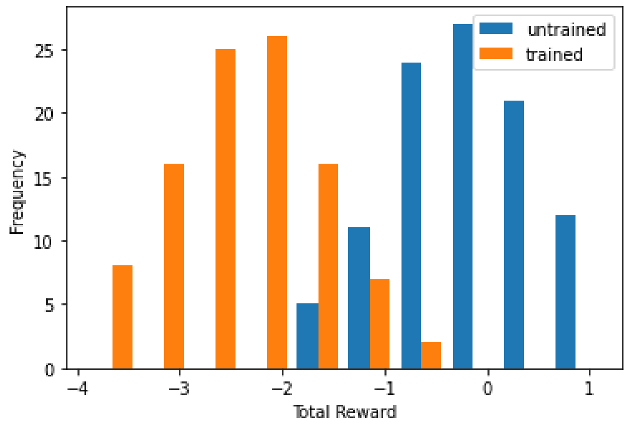
[18] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *CoRR*, abs/1710.11248, 2017.

A Appendix

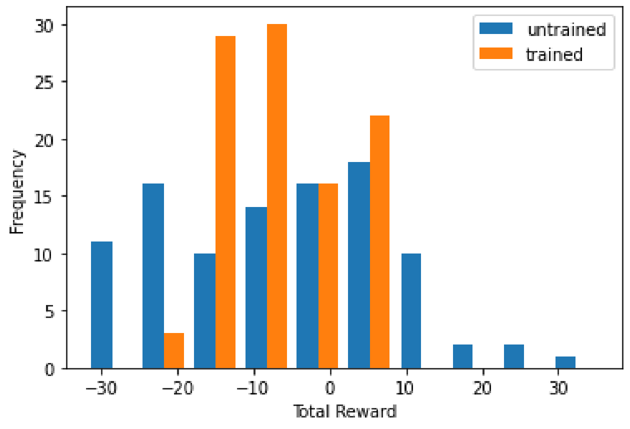
We show some examples of reward distribution plots for both the untrained and trained GAIL learners, and 100 episodes are recorded for each.



(a) CartPole GAIL evaluation reward distribution of trained and untrained agents after 100000 timesteps



(b) HalfCheetah GAIL evaluation reward distribution of trained and untrained agents after 50000 timesteps



(c) Swimmer GAIL evaluation reward distribution of trained and untrained agents after 100000 timesteps

Figure 13: Reward distribution plot examples