A Pragmatic Look at Deep Imitation Learning

Kai Arulkumaran¹ and Dan Ogawa Lillrank^{2,1}

¹ ARAYA Inc., 1-12-32 Akasaka, Minato-ku, Tokyo 107-6024, Japan ² AIST, 2-3-26 Aomi, Koto City, Tokyo 135-0064, Japan {kai_arulkumaran,dan_ogawa}@araya.org

Abstract. The introduction of the generative adversarial imitation learning (GAIL) algorithm has spurred the development of scalable imitation learning approaches using deep neural networks. The GAIL objective can be thought of as 1) matching the expert policy's state distribution; 2) penalising the learned policy's state distribution; and 3) maximising entropy. While theoretically motivated, in practice GAIL can be difficult to apply, not least due to the instabilities of adversarial training. In this paper, we take a pragmatic look at GAIL and related imitation learning algorithms. We implement and automatically tune a range of algorithms in a unified experimental setup, presenting a fair evaluation between the competing methods. From our results, our primary recommendation is to consider non-adversarial methods. Furthermore, we discuss the common components of imitation learning objectives, and present promising avenues for future research.

Keywords: imitation learning · inverse reinforcement learning.

1 Introduction

In the field of robotics, learning from demonstration [2] is widely used to construct control policies. In the field of machine learning, this is commonly known as imitation learning (IL) [21]. Given expert³ trajectories, one can formulate rich supervisory signals, which may even be preferred over the trial-and-error approach of reinforcement learning (RL).

In the same way that deep learning (DL) has enabled scaling RL to highdimensional state and action spaces, it has also lead to a host of new IL methods. In robotics, this may simply be using DL with behavioural cloning (BC) [35], one of the simplest IL methods, to train a vision-and-proprioception-guided policy [47]. As another example, one may use DL to recover an expert's cost function, and then use optimal control on this estimate [10]. These works rely on relatively simple IL algorithms and robotics domain knowledge; in contrast, more "sophisticated" approaches have been developed on simulated environments. We posit that properly benchmarking such algorithms will represent a step forwards towards being able to use them in more realistic domains.

³ Although IL can include suboptimal and/or partially observed data, in this paper we focus on the most basic/common setting.

2 Kai Arulkumaran and Dan Ogawa Lillrank

A seminal work is generative adversarial imitation learning (GAIL) [20], which builds upon the real vs. fake discrimination task for generative models introduced by Goodfellow et al. [15]. Considering the training process, GAIL involves training a policy to 1) match the expert's state-action distribution; 2) avoid state-actions from previous iterations of the learned policy; and 3) maximise entropy. These principles underlie most IL algorithms, which then differ in how these are achieved.

However, while many DL-based IL algorithms have been introduced in recent years, it is not common knowledge which might be the best fit for a given problem. Results in deep RL have been brought into question [19], and it is often the case that implementation details matter more than the algorithmic contributions of novel algorithms [1]. It is now the time for this to be investigated in the context of IL. Recent, complementary work has investigated the use of proxies for the real reward function in tuning IL algorithms [22], as well as a large-scale investigation of hyperparameters for adversarial IL algorithms [33].

In this work, similarly to Orsini et al. [33], we investigate a range of adversarial IL methods [20,12,26,14] on standard simulated robot environments [5]. However, while they focus on adversarial learning, we also examine other DL-based methods [25,45,4], unifying them from the perspective of the three previouslymentioned IL objectives (Subsection 2.7). Comparing the algorithms using a unified codebase and an equal hyperparameter optimisation budget (Section 3), we find that 1) BC remains competitive in low-dimensional state spaces; 2) generative moment matching imitation learning (GMMIL) [25] performs the best out of all newer methods tested; and 3) all other algorithms perform similarly when averaged over all tested environments. Given these results, we advocate trialling non-adversarial IL methods, and present time and memory complexity tradeoffs (Appendix A.3) for further consideration. Our code is made available⁴ to facilitate further research.

2 Background

2.1 Imitation Learning

The goal of IL is to train a parameterised policy, $\hat{\pi}(a|s;\theta)$, mapping states s to a distribution over actions a, to mimic an expert policy $\pi^*(a|s)$, given either the expert policy itself, or more commonly, a fixed dataset $\xi^* = \{\tau_1, \ldots, \tau_N\}$, of trajectories $\tau = \{s_0, a_0, s_1, a_1, \ldots, s_T, a_T\}$ generated by the expert, where N denotes the number of expert trajectories provided.

A common assumption within IL is that both the expert and our agent inhabit a Markov decision process (MDP), defined by the tuple $(S, \mathcal{A}, \mathcal{T}, \mathcal{R}, p_0, \gamma)$: S and \mathcal{A} are the state and action spaces, $\mathcal{T} : S \times \mathcal{A} \to S$ is the state transition dynamics, $\mathcal{R} : S \times \mathcal{A} \to \mathbb{R}$ is the reward function, $p_0(s)$ is the initial state distribution, and $\gamma \in [0, 1]$ is the discount factor (used to weight immediate vs. future rewards). The expert policy is optimal in the sense that $\pi^* = \operatorname{argmax}_{\pi \in II} \mathbb{E}_{\tau \sim \pi}[R_0]$,

⁴ https://github.com/Kaixhin/imitation-learning.

where the return at timestep t, R_t , is the discounted sum of rewards following a policy from state s_t until the end of the episode at timestep T: $R_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k+1}$. While in RL [43] the goal is to interact with the environment in order to find π^* , in IL we do not have access to \mathcal{R} , and must instead find π^* assuming that we have access to optimal trajectories. All following methods, unless specified otherwise, can be implemented using neural networks, providing flexible function approximation that can scale to large state and/or action spaces.

2.2 Reduction to Supervised Learning

The simplest method, BC [35], reduces IL to a supervised learning problem. Using a^* to denote the expert's actions, BC can be formulated as minimising the 1-step deviation from the expert trajectories:

$$\operatorname{argmin}_{\theta} \mathbb{E}_{s,a^* \sim \xi^*} [\mathcal{L}(a^*, \hat{\pi}(a|s; \theta))], \tag{1}$$

where \mathcal{L} can be, as in maximum likelihood estimation, the negative log likelihood.

BC is very simple, and benefits from a fixed objective over a stationary data distribution. However, as $\hat{\pi}$ is only trained on $s \sim \xi^*$, it can fail catastrophically when it diverges from the states covered by π^* .

2.3 Inverse Reinforcement Learning

Inverse reinforcement learning (IRL) overcomes this by using RL to train $\hat{\pi}$ to mimic π^* . The procedure consists of iterating between the following two steps:

1. Construct a reward function $\hat{\mathcal{R}}(s, a; \phi)$ using ξ^* , and optionally $\tau \sim \hat{\pi}$

2. Train $\hat{\pi}$ using RL

RL is more complicated than the typical supervised learning setting. In particular, as the policy evolves, the data distribution changes. In the case of IRL, $\hat{\mathcal{R}}$ changing over time can introduce further non-stationarity.

The basic objective of IRL can be stated as:

$$\operatorname{argmax}_{\theta} \mathbb{E}_{\tau \sim \hat{\pi}(s,a;\theta)}[\hat{R}_0] \quad \text{such that} \quad \pi^* = \operatorname{argmax}_{\pi \in \Pi} \mathbb{E}_{\tau \sim \pi}[\hat{R}_0], \tag{2}$$

where \hat{R} is the return with respect to the learned reward function $\hat{\mathcal{R}}$. However, this is underspecified [32]; e.g., any policy is trivially optimal for $\hat{\mathcal{R}} = 0$. IRL algorithms therefore incorporate one or several of the following three properties.

Firstly, one can match the state-action distribution under π^* , known as the expert's occupancy measure $\rho_{\pi^*} = \mathbb{E}_{\tau \sim \pi^*} \left[\sum_{t=0}^T \gamma^t \mathbb{1}_{s,a} \right]$ [44,20], or, alternatively, feature expectations [32]. This is achieved using the learned reward function, and is hence dependent on the expressivity of $\hat{\mathcal{R}}$. In particular, the constant function is underspecified and allows an infinite set of solutions. Secondly, one can "penalise" following trajectories taken by (previous iterations of) $\rho_{\hat{\pi}}$ [32]. This allows $\hat{\mathcal{R}}$ to focus on relevant parts of the state-action space, but implicitly

4

assumes that current/past versions of $\hat{\pi}$ are suboptimal. Thirdly, one can use the maximum entropy principle [23] to find a unique best solution out of the set of solutions that match the expert's occupancy measure/feature expectations [48]. Using the Lagrangian multiplier λ , and denoting H as the entropy, this results in the following modified RL objective: $\operatorname{argmax}_{\theta} \mathbb{E}_{\tau \sim \pi(s,a;\theta)}[R_0] + \lambda H[\pi(s,a;\theta)]$. Entropy regularisation is a classic technique in RL [46].

2.4 Adversarial Imitation Learning

The theory behind GAIL [20] is that of maximum entropy occupancy measure matching. In generative adversarial network training [15], the "generator" is trained to output samples that fool the "discriminator" $D: S \times A \to (0, 1)$, whilst the discriminator is trained to discriminate between samples from the generator and the data distribution. This is a minimax game, in which the equilibrium solution corresponds to minimising the Jensen-Shannon divergence between the generated and real distributions. In GAIL, $\hat{\pi}$ plays the role of the generator, and the discriminator is trained on state-action pairs from $\hat{\pi}$ and $\pi^*: \min_G \max_D \mathbb{E}_{s,a \sim \pi^*}[\log(D(s,a))] + \mathbb{E}_{s,a \sim \hat{\pi}}[\log(1 - D(s,a))]$. Under this formulation, higher values indicate how "expert" D believes its input to be.

There are several options for constructing \mathcal{R} from D. Prominent examples include those introduced in GAIL, adversarial inverse reinforcement learning (AIRL) [12] (corresponding to the reverse Kullback-Leibler (KL) divergence $D_{\text{KL}}(\rho_{\hat{\pi}} || \rho_{\pi^*})$), and forward KL AIRL (FAIRL) [14] (Table 1). As discussed by Kostrikov et al. [26] and empirically investigated by Jena et al. [24], there is a potential reward bias in these functions. They note that positive $\hat{\mathcal{R}}$, i.e., $-\log(1 - D(s, a))$, biases agents towards survival, whereas negative $\hat{\mathcal{R}}$, i.e., $\log(D(s, a))$ biases agents towards early termination. This bias means that even constant reward functions can outperform either of these depending on the type of the environment. In line with both their theoretical and empirical findings, we hence use $\log(D(s, a)) - \log(1 - D(s, a))$ for our implementations of GAIL and AIRL, while keeping FAIRL's original $\hat{\mathcal{R}}$. We recommend the original works for discussions on the properties of various reward functions [26,24,14].

Table 1: Adversarial imitation reward functions [14].

	$\hat{\mathcal{R}}$	Positive (bounded)	Negative (bounded)
GAIL	$\log D(s,a)$	X (-)	✓ (X)
AIRL	$h(s, a) = \log(D(s, a)) - \log(1 - D(s, a))$	✓ (X)	✓(X)
FAIR	$-h(s,a) \cdot e^{h(s,a)}$	√ (√)	✓(X)

While GAIL implicitly returns a reward function, if trained to optimality then D will return 0.5 for state-action pairs from both $\hat{\pi}$ and π^* . Finn et al. [9] propose changing the form of the D to $\frac{\exp(f(\tau))}{\exp(f(\tau))+\hat{\pi}(\tau)}$, allowing the optimal reward function to be recovered as f(+const). AIRL makes a practical algorithm from this by

changing D to operate over state-action pairs, as in GAIL, and also further disentangling the recovered reward function f as the sum of a reward approximator g(s, a) and a reward shaping [31] term h(s): $f(s, a, s') = g(s, a) + \gamma h(s') - h(s)$, where s' is the successor state. We use this form of D and the original AIRL reward function in our implementations.

2.5 Distance-based Imitation Learning

One of the disadvantages of adversarial training is the requirement for the discriminator, which is also undergoing training as part of the minimax game, to provide a useful training signal to the generator. One solution is to replace the discriminator with a nonparametric model [28,7]. Specifically, distribution matching can be achieved by minimising the maximum mean discrepancy (MMD) [16] defined over a reproducing kernel Hilbert space (RKHS). Given distributions, P and Q, and a mapping $\psi : \mathcal{X} \to \mathcal{H}$ from features $X \in \mathcal{X}$ to an RKHS \mathcal{H} , the MMD is the distance between the mean embeddings of the features: MMD(P, Q) = $\|\mathbb{E}_{x\sim P}[\psi(x)] - \mathbb{E}_{y\sim Q}[\psi(y)]\|_{\mathcal{H}}$. Using a kernel function⁵ k, one can calculate MMD²(P, Q) = $\mathbb{E}_{x,x'\sim P}k(x,x') + \mathbb{E}_{y,y'\sim Q}k(y,y') - 2\mathbb{E}_{x\sim P,y\sim Q}k(x,y)$. GMMIL [25] extends this principle to the IL setting. Dropping terms that

are constant with respect to $\hat{\pi}$, GMMIL has the reward function:

$$\hat{\mathcal{R}} = \frac{1}{M} \sum_{i=1}^{M} k((s,a), (s_i^*, a_i^*)) - \frac{1}{N} \sum_{j=1}^{N} k((s,a), (s_j, a_j)),$$
(3)

where M and N are the number of state-action pairs from π^* and $\hat{\pi}$, respectively.

Two disadvantages of GMMIL are that 1) the "discriminator" cannot learn relevant features, and 2) it has O(MN) complexity. Random expert distillation (RED) [45] solves these issues by building upon random network distillation (RND) [6]. In RND, a predictor network $f_{\phi} : S \times A \to \mathbb{R}^{K}$ is trained to minimise the mean squared error (MSE) against a fixed, randomly initialised network $f_{\phi} : S \times A \to \mathbb{R}^{K}$. Empirically, the MSE indicates how out-of-distribution new data is. RED utilises a Gaussian function over the MSE, resulting in

$$\hat{\mathcal{R}} = \exp(-\sigma \| f_{\phi}(s,a) - f_{\bar{\phi}}(s,a) \|_2^2), \tag{4}$$

where σ is a bandwidth hyperparameter (Appendix A.1).

2.6 Uncertainty-based Imitation Learning

Similarly to RED, disagreement-regularised IL (DRIL) [4] constructs a reward function based on the disagreement between models trained on the expert data. However, unlike the other methods which operate over the joint distribution of state-action pairs, DRIL builds simply upon BC, operating over p(a|s). DRIL first trains an ensemble of E different policies using the BC objective (Eqn. 1)

⁵ Kernels are similarity functions. A discussion is given in Appendix A.1.

Kai Arulkumaran and Dan Ogawa Lillrank

on the expert data, and then uses a function of the (negative of the) variance between the policies to estimate a reward for the agent:

$$\hat{\mathcal{R}} = -C_{\mathrm{U}}^{\mathrm{clip}}(s, a) = \begin{cases} +1 & \text{if } \operatorname{Var}_{\pi \in \Pi_E}[\pi(a|s)] \le q\\ -1 & \text{otherwise,} \end{cases}$$
(5)

where the q is a top quantile of the uncertainty cost computed over the expert dataset. Deep ensembles are known to produce reasonable uncertainty estimates (i.e., variance in outputs) on out-of-distribution data [27]. In our implementation⁶, we approximate the ensemble using sampling with dropout [42], as this was shown to perform comparatively to using independent models [4].

2.7 Comparison

6

Tab. 2 gives an overview of some important properties of IL algorithms. We believe this framework helps frame promising avenues for future research in IL algorithms. For example, could RED be improved by updating $\hat{\mathcal{R}}$, in the same way as adversarial IL algorithms? Conversely, would restricting the updates of $\hat{\mathcal{R}}$ improve adversarial IL algorithms? In this paper, we test one hypothesis, which is the importance of "penalising" $\rho_{\hat{\pi}}$. Concretely, the second term of GMMIL's reward function (Eqn. 3) penalises self-similarity in $\rho_{\hat{\pi}}$. In our experiments (Section 3), we see that whether this term is helpful or not is environment-dependent; this suggests that automatically tuning coefficients of the two terms in GMMIL's reward function could result in an improved IL algorithm.

Table 2: Properties of IL algorithms. We do not include entropy maximisation, as this is essentially a property of the RL algorithm used. * These algorithms operate over p(a|s), rather than p(s, a). [†] While GMMIL "penalises" $\rho_{\hat{\pi}}$, we experiment with making this term optional.

•	-		
	Match ρ_{π^*}	Penalise $\rho_{\hat{\pi}}$	Fixed $\hat{\mathcal{R}}$
BC	✓*	×	-
GAIL/AIRL/FAIRL	1	1	X
GMMIL	1	\checkmark^{\dagger}	1
RED	1	×	1
DRIL	✓*	×	1

⁶ The original DRIL implementation interleaves IRL with BC training; however, we only use the reward function for a clear comparison against other algorithms.

 $\overline{7}$

3 Experiments

3.1 Setup

We base our experiments on the PyBullet version⁷ of the datasets for deep datadriven RL (D4RL) [11]. Although this benchmark was developed for offline RL, it can be used for IL by ignoring rewards. The benchmark contains data for 4 standard simulated robotics environments: Ant, HalfCheetah, Hopper, and Walker2D. We use the highest performance data⁸, subsampled⁹ by 20 [20].

A common choice of on-policy RL algorithm to use within these methods [20,12,25,45] is trust region policy optimisation [39]; we opt instead to use proximal policy optimisation (PPO) [40], which also approximately enforces a trust region on policy updates, but is simpler and achieves comparable results. Theoretically, one can use more sample-efficient off-policy RL algorithms with importance sampling (IS), although empirically this still works well without IS [26]; we leave investigating this for future work. Kostrikov et al. [26] also introduce two further ways to improve the performance of deep IL algorithms. Firstly, they add a terminal state indicator to allow learning non-zero rewards for absorbing states; empirically they found that this is important for sparse reward/goalbased tasks like target reaching, so we leave this addition out given our set of environments. Secondly, they add a gradient penalty on the discriminator [17] to prevent overfitting. In our experiments we use the R_1 gradient penalty, which is motivated by improving the convergence of adversarial training [30]: $R_1(\phi) = \mathbb{E}_{\xi^*}[\|\nabla D(s, a; \phi)\|_2^2]$. A way to alleviate data shift in online learning is to use experience replay [29], so we add this as an option when relevant.¹⁰

We re-iterate that a primary aim of our work is to compare the approximate reward functions of the different proposed algorithms, under a unified experimental setup, given an equal¹¹ hyperparameter tuning budget. In summary, important changes against the original algorithms are: 1) we use the AIRL reward function with GAIL; 2) we use the R_1 gradient penalty with all adversarial methods; 3) we add an option to disable the self-similarity term in GMMIL; and 4) we do not interleave BC training with DRIL. Hyperparameters and their optimisation are detailed in Tab. A1; optimised hyperparameters are shown in Fig. A1.

⁷ https://github.com/takuseno/d4rl-pybullet is based on the free and opensource PyBullet physics simulator. The dataset and any results are not directly comparable to the MuJoCo version.

⁸ In D4RL-PyBullet these are the "medium" datasets, collected by a soft-actor critic agent [18] during training, consisting of 10⁶ transitions.

⁹ Kostrikov et al. [26] note that BC is competitive with GAIL when provided large datasets. Subsampling limits dataset size, but we agree with Hussenot et al. [22] that there is little justification from a practical perspective.

¹⁰ In our implementation, D is trained on the entire replay's worth of data, hence the number of training updates is proportional to the size of the replay.

¹¹ In terms of function evaluations, not time.

8 Kai Arulkumaran and Dan Ogawa Lillrank

3.2 Results

Fig. 1 and Tab. 3 shows the performance of the different algorithms, evaluated (but not trained) against the ground truth reward function. In terms of performance, GMMIL performs best overall, although BC is competitive on the lower-dimensional environments. The relative ranking of the rest of the algorithms changes over the different environments, and it is unclear that any are particularly better than the others.



Fig. 1: PyBullet evaluation results. Mean \pm standard error, over 5 random seeds. Each evaluation consists of 50 episodes with a deterministic policy.

indom seeds. Evaluation consists of 50 episodes with a deterministic policy.				
	Ant	HalfCheetah	Hopper	Walker2D
PPO	936.01 ± 551.29	1299.46 ± 300.27	1114.07 ± 331.77	535.84 ± 246.42
Dataset	570.80 ± 104.82	787.35 ± 104.31	1078.36 ± 325.52	1106.68 ± 417.79
BC	629.17 ± 19.13	509.31 ± 185.67	1005.64 ± 12.18	220.00 ± 23.69
GAIL	420.82 ± 182.67	-863.37 ± 638.37	12.99 ± 1.30	280.53 ± 211.39
AIRL	270.02 ± 59.43	24.08 ± 511.16	444.78 ± 204.99	322.07 ± 210.98
FAIRL	498.85 ± 95.15	-1411.01 ± 150.52	497.46 ± 322.36	519.13 ± 99.80
GMMIL	591.37 ± 79.67	225.82 ± 545.62	1192.84 ± 68.12	645.05 ± 66.60
RED	402.95 ± 163.81	-1373.72 ± 89.30	641.32 ± 158.37	548.43 ± 124.35
DRIL	413.50 ± 109.34	-1416.37 ± 48.11	761.93 ± 96.10	590.94 ± 87.64

Table 3: PyBullet final evaluation results. Mean \pm standard deviation, over 5 random seeds. Evaluation consists of 50 episodes with a deterministic policy.

However, GMMIL has the worst time (Tab. A2) and memory complexity (Tab. A3); in particular, with commonly used settings GMMIL uses over 10GB of

RAM, an order of magnitude more than all other methods. Apart from GMMIL, non-adversarial algorithms have the best time complexity.

With reference to our hypothesis in Subsection 2.7, GMMIL's self-similarity term was disabled in half of the environments at the end of hyperparameter optimisation. This means that this term may be beneficial, detrimental, or have little effect depending on the environment. Whether the term is removed or not is not correlated with the optimal entropy regularisation cost, so it is not the case that the latter needs to compensate for the loss of the self-similiarity term.

We note that PPO's performance was significantly lower than the dataset's, and many IL algorithms reached this performance. This underscores the need for using strong RL algorithms within IRL.

4 Discussion

In this paper, we took a pragmatic look at deep IL methods, finding common principles between them (Subsection 2.7), and performing as fair a comparison as we could, using the algorithms as described in the original works. In our experimental setup, we found that the only algorithm that consistently worked well was GMMIL, and it was unclear if any of the others would be a reasonable second choice. On the other hand, GMMIL may require a relatively large amount of memory, making it unfavourable depending on hardware availability. Given the performance and complexity of adversarial IL algorithms, it is therefore worth considering alternatives first.

As highlighted by the results on Walker2D, using more performant RL algorithms is an important avenue for future work. Another valuable direction is to investigate the importance of the source of the dataset. The contemporaneous work of Orsini et al. [33] discusses how human expert data can largely differ from "artificial" expert data, to the extent that the form of the optimum learned reward function should be changed. As they focus on adversarial methods, it would be instructive to see how the other methods we examined would perform on such data. Our code, developed for benchmarking, should facilitate such research.

References

- Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., et al.: What Matters in Onpolicy Reinforcement Learning? A Large-scale Empirical Study. arXiv:2006.05990 (2020)
- Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. RAS 57(5), 469–483 (2009)
- Balandat, M., Karrer, B., Jiang, D., Daulton, S., Letham, B., Wilson, A.G., Bakshy, E.: BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. NeurIPS 33 (2020)
- Brantley, K., Sun, W., Henaff, M.: Disagreement-regularized Imitation Learning. In: ICLR (2019)

- 10 Kai Arulkumaran and Dan Ogawa Lillrank
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym. arXiv:1606.01540 (2016)
- Burda, Y., Edwards, H., Storkey, A., Klimov, O.: Exploration by Random Network Distillation. In: ICLR (2018)
- Dziugaite, G.K., Roy, D.M., Ghahramani, Z.: Training Generative Neural Networks via Maximum Mean Discrepancy Optimization. In: UAI. pp. 258–267 (2015)
- Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., Madry, A.: Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO. arXiv:2005.12729 (2020)
- Finn, C., Christiano, P., Abbeel, P., Levine, S.: A Connection Between Generative Adversarial Networks, Inverse Reinforcement Learning, and Energy-based Models. arXiv:1611.03852 (2016)
- Finn, C., Levine, S., Abbeel, P.: Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization. In: ICML. pp. 49–58 (2016)
- Fu, J., Kumar, A., Nachum, O., Tucker, G., Levine, S.: D4RL: Datasets for Deep Data-driven Reinforcement Learning. arXiv:2004.07219 (2020)
- Fu, J., Luo, K., Levine, S.: Learning Robust Rewards with Adversarial Inverse Reinforcement Learning. In: ICLR (2018)
- Fukumizu, K., Gretton, A., Sun, X., Schölkopf, B.: Kernel Measures of Conditional Dependence. In: NeurIPS. vol. 20, pp. 489–496 (2007)
- Ghasemipour, S.K.S., Zemel, R., Gu, S.: A Divergence Minimization Perspective on Imitation Learning Methods. In: CoRL (2020)
- Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative Adversarial Networks. In: NeurIPS (2014)
- Gretton, A., Borgwardt, K.M., Rasch, M.J., Schölkopf, B., Smola, A.: A Kernel Two-sample Test. JMLR 13(1), 723–773 (2012)
- Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.: Improved Training of Wasserstein GANs. In: NeurIPS (2017)
- Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft Actor-critic: Off-policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In: ICML. pp. 1861–1870 (2018)
- Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D.: Deep Reinforcement Learning that Matters. In: AAAI (2018)
- 20. Ho, J., Ermon, S.: Generative Adversarial Imitation Learning. In: NeurIPS (2016)
- Hussein, A., Gaber, M.M., Elyan, E., Jayne, C.: Imitation Learning: A Survey of Learning Methods. ACM CSUR 50(2), 1–35 (2017)
- Hussenot, L., Andrychowicz, M., Vincent, D., Dadashi, R., Raichuk, A., Stafiniak, L., Girgin, S., Marinier, R., Momchev, N., Ramos, S., et al.: Hyperparameter Selection for Imitation Learning. In: ICML (2021)
- Jaynes, E.T.: Information Theory and Statistical Mechanics. Physical Review 106(4), 620 (1957)
- Jena, R., Agrawal, S., Sycara, K.: Addressing Reward Bias in Adversarial Imitation Learning with Neutral Reward Functions. arXiv:2009.09467 (2020)
- Kim, K.E., Park, H.S.: Imitation Learning via Kernel Mean Embedding. In: AAAI (2018)
- Kostrikov, I., Agrawal, K.K., Dwibedi, D., Levine, S., Tompson, J.: Discriminatoractor-critic: Addressing Sample Inefficiency and Reward Bias in Adversarial Imitation Learning. In: ICLR (2018)
- 27. Lakshminarayanan, B., Pritzel, A., Blundell, C.: Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In: NeurIPS. pp. 6405–6416 (2017)

- Li, Y., Swersky, K., Zemel, R.: Generative Moment Matching Networks. In: ICML. pp. 1718–1727 (2015)
- Lin, L.J.: Self-improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. Machine learning 8(3-4), 293–321 (1992)
- Mescheder, L., Geiger, A., Nowozin, S.: Which Training Methods for GANs do Actually Converge? In: ICML (2018)
- Ng, A.Y., Harada, D., Russell, S.: Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In: ICML. vol. 99, pp. 278–287 (1999)
- Ng, A.Y., Russell, S.J., et al.: Algorithms for Inverse Reinforcement Learning. In: ICML. vol. 1, p. 2 (2000)
- 33. Orsini, M., Raichuk, A., Hussenot, L., Vincent, D., Dadashi, R., Girgin, S., Geist, M., Bachem, O., Pietquin, O., Andrychowicz, M.: What Matters for Adversarial Imitation Learning? arXiv:2106.00672 (2021)
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. NeurIPS 32, 8026–8037 (2019)
- Pomerleau, D.A.: ALVINN: An Autonomous Land Vehicle in a Neural Network. In: NeurIPS (1989)
- Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., Dormann, N.: Stable Baselines3. https://github.com/DLR-RM/stable-baselines3 (2019)
- Ramdas, A., Reddi, S.J., Póczos, B., Singh, A., Wasserman, L.: On the Decreasing Power of Kernel and Distance Based Nonparametric Hypothesis Tests in High Dimensions. In: AAAI (2015)
- Schölkopf, B., Smola, A.J., Bach, F., et al.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press (2002)
- Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust Region Policy Optimization. In: ICML. pp. 1889–1897 (2015)
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. arXiv:1707.06347 (2017)
- Smola, A., Gretton, A., Song, L., Schölkopf, B.: A Hilbert Space Embedding for Distributions. In: ALT. pp. 13–31 (2007)
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR 15(1), 1929–1958 (2014)
- Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (2018)
- Syed, U., Bowling, M., Schapire, R.E.: Apprenticeship Learning using Linear Programming. In: ICML. pp. 1032–1039 (2008)
- 45. Wang, R., Ciliberto, C., Amadori, P.V., Demiris, Y.: Random Expert Distillation: Imitation Learning via Expert Policy Support Estimation. In: ICML (2019)
- Williams, R.J., Peng, J.: Function Optimization using Connectionist Reinforcement Learning Algorithms. Connection Science 3(3), 241–268 (1991)
- Zhang, T., McCarthy, Z., Jow, O., Lee, D., Chen, X., Goldberg, K., Abbeel, P.: Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In: ICRA. pp. 5628–5635 (2018)
- Ziebart, B.D., Maas, A.L., Bagnell, J.A., Dey, A.K.: Maximum Entropy Inverse Reinforcement Learning. In: AAAI (2008)

A Appendix

A.1 Kernels

Kernel functions $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ are commonly used in nonparametric models, where the output of the model is a function of the similarity between query points and (most commonly) a subset of the training set. Kernels can also be used represent probability distributions as an element of RKHS [41], enabling the comparison and manipulation of probability distributions. Some advantages of this approach to modelling distributions include not having to make parametric assumptions or perform intermediate density estimation. For characteristic kernels [13], where the mean embedding is injective, MMD(P, Q) = 0 if and only if P and Q are the same distribution.

The original implementation of GMMIL uses the Gaussian kernel, which is one of the most widely-used characteristic kernels. Following prior work on generative moment matching [28], GMMIL uses two kernels. The kernel bandwidth parameters, σ_1 and σ_2 , are set using the median heuristic [38,37]. σ_1 is set as "the median of the pairwise squared- ℓ_2 distances among the data points from the expert policy and the initial policy", and σ_2 is set as the median of the pairwise distances between the expert data points [25].

For RED, Wang et al. [45] "choose σ such that r(s, a) from expert demonstrations are mostly close to 1." However it is unclear how to achieve this for a given dataset, as analytically the only choice is always $\sigma = 0$, in which case the reward is a constant 1 for all state-action pairs. Therefore, for our experiments we opted instead to use the median heuristic, calculated over the expert dataset.

A.2 Hyperparameters

Table A1: Hyperparameters for PyBullet, with PPO hyperparameters based primarily on prior works [1,8,36]. Brackets indicate hyperparameter search space, which we optimise over using Bayesian optimisation [3]; each algorithm is given a hyperparameter budget of 20 evaluations, with the average cumulative return of the last 5 evaluations used as the optimisation objective. * is specific to BC, RED and DRIL. ⁺ is specific to GAIL, AIRL, FAIRL, RED and DRIL. [†] is specific to GAIL, AIRL and FAIRL. All experiments are run using a single (non-batched) environment instance.

Value
2×10^{6}
20
2
256
Tanh
Orthogonal
0.01
Gaussian
State-independent
e^{-2}
0.99
0.9
Yes
$[3 \times 10^{-5}, 3 \times 10^{-4}]$
[1024, 2048, 4096]
0.5
0.25
[5, 10, 20]
0.5
$[0, 10^{-3}, 10^{-2}]$
[5, 15, 25]
$[3 \times 10^{-5}, 3 \times 10^{-4}]$
[5, 15, 25]
[1, 3, 5]
[0.1, 0.5, 1]



14 Kai Arulkumaran and Dan Ogawa Lillrank

Fig. A1: Best hyperparameters for each algorithm, across every environment. "Imitation epochs" denotes pretraining for BC, RED and DRIL, and training for GAIL, AIRL and FAIRL. Although some algorithm hyperparameters are the same across all environments, in general there does not appear to be any noticeable trends.

A.3 Time/Memory Complexity

Table A2: Time taken (s) for pretraining and training each algorithm on the Hopper environment for 10^6 steps; mean \pm standard deviation, over 5 runs. Standardised hyperparameters for all algorithms, based on common values selected during hyperparameter optimisation, include: rollout buffer length of 2048; PPO iterations of 10; imitation pretraining epochs of 25; adversarial training epochs of 5; imitation replay size of 3. All experiments were run on an Ubuntu 18.04 machine with an Intel i7-9700K CPU @ 3.60GHz and 64 GB RAM @ 2666 MHz, using PyTorch 1.7.1 [34].

	Pretraining	Training
BC	18 ± 1	-
GAIL	-	$2,653\pm9$
AIRL	-	$2,819\pm16$
FAIRL	-	$2,618\pm6$
GMMIL	-	$3,524\pm 8$
RED	21 ± 2	$2,021\pm5$
DRIL	30 ± 2	$1,982 \pm 5$

15

Table A3: Max memory taken (MB) for training each algorithm on the Hopper environment; mean \pm standard deviation, over 5 runs. Standardised hyperparameters for all algorithms include: rollout buffer length of 2048; imitation replay size of 3. All experiments were run on an Ubuntu 18.04 machine using PyTorch 1.7.1 [34].

	Training
BC	435 ± 2
GAIL	569 ± 33
AIRL	623 ± 12
FAIR	L 580 ± 14
GMM	$111 14, 539 \pm 10 $
RED	1041 ± 16
DRIL	604 ± 3