RAIL: A modular framework for Reinforcement-learning-based Adversarial Imitation Learning[⋆]

Eddy Hudson[1][0000−0002−8350−3919], Garrett Warnell[1,2][0000−0003−3846−8787], and Peter Stone[1,3][0000−0002−6795−420X]

[1] The University of Texas at Austin, TX 78712, USA
[2] US Army Research Laboratory
[3] Sony AI

Abstract. While Adversarial Imitation Learning (AIL) algorithms have recently led to state-of-the-art results on various imitation learning benchmarks, it is unclear as to what impact various design decisions have on performance. To this end, we present here an organizing, modular framework called Reinforcement-learning-based Adversarial Imitation Learning (RAIL) that encompasses and generalizes a popular subclass of existing AIL approaches. Using the view espoused by RAIL, we create two new IfO (Imitation from Observation) algorithms, which we term SAIfO: SAC-based Adversarial Imitation from Observation and SILEM (Skeletal Feature Compensation for Imitation Learning with Embodiment Mismatch). We go into greater depth about SILEM in a separate technical report [11]. In this paper, we focus on SAIfO, evaluating it on a suite of locomotion tasks from OpenAI Gym, and showing that it outperforms contemporaneous RAIL algorithms that perform IfO.

Keywords: Reinforcement Learning · Imitation Learning · Adversarial Imitation Learning

1 Introduction

In the past decade, the field of deep learning has been punctuated by revolutionary results in computer vision, natural language processing (NLP), and reinforcement learning (RL). Key to this explosive growth has been an implicit modularization that has allowed researchers to work on improving individual modules independently and in parallel. For example, in computer vision, an

[⋆] This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CPS-1739964, IIS-1724157, NRI-1925082), ONR (N00014-18-2243), FLI (RFP2-000), ARO (W911NF-19-2-0333), DARPA, Lockheed Martin, GM, and Bosch. Peter Stone serves as the Executive Director of Sony AI America and receives financial compensation for this work. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

approach to tackle the ImageNet challenge [16] can be said to comprise the following two main modules among a host of other components: an optimization algorithm, and the deep network architecture. Research into optimization algorithms has yielded algorithms such as AdaGrad [4] and Adam [12], while research into deep network architectures has resulted in architectures such as convolutional networks and ResNet [9]. Progress in NLP can similarly be characterized by modules corresponding to optimization algorithm and network architecture. Research into network architectures for NLP has been taking place independent of the optimization algorithm used, and has resulted in success stories from seq2seq [19] to the Transformer [21].

Recently, there has been rapid progress on imitation learning (IL), especially with respect to a class of new algorithms termed adversarial imitation learning (AIL). Starting from GAIL [10] and GAIfO [20], to DAC [13] and OPOLO [22], the sample complexity of AIL approaches have been rapidly declining. However, progress in this space has been ad hoc: AIL research lacks the type of organizing and modular characterization that has led to the steady and substantial improvement enjoyed by the other communities mentioned above.

In this work, we aim to provide a modular characterization for a sub-class of AIL algorithms that employs RL. We term this class of algorithms Reinforcement-learning-based Adversarial Imitation Learning (RAIL). We characterize RAIL techniques using a modular framework, with modules corresponding to the RL backbone and the input format to the discriminator. As evidence that the modularization of RAIL can accelerate progress in it, we explore the resulting design space and discover a new RAIL variant for imitation from observation that we call SAC-based Adversarial Imitation from Observation (SAIfO), where SAC [7] is a recently proposed RL algorithm. We evaluate SAIfO on a suite of locomotion tasks from OpenAI Gym [3], and show that it outperforms recent IfO algorithms that fall under the RAIL umbrella.

2 Background

The ultimate goal in imitation learning is to learn a controller that solves a sequential decision making problem. Such problems are typically formulated in the context of a Markov decision process (MDP), i.e., a tuple $\mathcal{M} =< \mathcal{S}, \mathcal{A}, T, R, \gamma >$, where $\mathcal{S}$ denotes an agent's state space, $\mathcal{A}$ denotes the agent's action space, $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ denotes the environment model which maps state-action pairs to a distribution over the agent's next state, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a reward function that provides a scalar-valued reward signal for state-action-next-state tuples, and $\gamma \in [0, 1]$ is a discount factor that specifies how the agent should weight short- vs. long-term rewards. Solutions to sequential decision making problems are often specified by reactive policies $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, which specify agent behavior by providing a mapping from the agent's current state to a distribution over the actions it can take. Machine learning solutions to problems described by an MDP typically search for policies that can maximize the agent's expected sum of future rewards.

The IL problem is typically formulated using an MDP *without* a specified reward function, i.e., $\mathcal{M} \setminus R$. Instead of reward, the agent is provided with *demonstration* trajectories—typically assumed to have been generated by an expert—that specifies the desired behavior, i.e., $\tau_E = (s_0, a_0, s_1, a_1, ...)$. Imitation from observation (IfO) is a sub-problem of IL in which the agent does not have access to the actions taken during the demonstration trajectories, i.e., $\tau_E = (s_0, s_1, ...)$. Techniques designed to solve the IL problem seek to use observed demonstrations to find policies that an imitating agent can use to imitate the demonstrator.

Adversarial imitation learning (AIL) is a particular way to perform IL that has recently come to the fore (Figure 1). AIL leverages the adversarial training technique popularized by GANs [6], in that both involve the same min-max game with discriminators and generator networks. The discriminator, D, is trained to distinguish between the demonstration trajectories and trajectories generated by the imitator. In particular, the goal of updating D is to drive $\mathbb{E}_{o \sim \tau_E}[D(o)]$ toward 1 and $\mathbb{E}_{o \sim \tau}[D(o)]$ toward 0, where o is a segment of the trajectory, τ represents trajectories recently generated by the imitator, and τ_E is a dataset of demonstration trajectories. In the seminal AIL algorithm GAIL [10] and more recent AIL algorithms ASAF [2] and ValueDICE [14], $o = (s_t, a_t)$, whereas in GAIfO [20] and OPOLO [22], $o = (s_t, s_{t+1})$. The generator, which in AIL algorithms is the imitator's policy π, is trained to induce behavior that elicits large output from D, i.e., to "fool" D into thinking that the imitator's trajectories came from the demonstrator. By iteratively updating D and π as described, AIL approaches are able to find imitator policies that successfully mimic the demonstrated behavior.

3 RAIL: Reinforcement-learning-based Adversarial Imitation Learning

In this work, we focus on a specific slice of AIL that we term RAIL: Reinforcement-learning-based Adversarial Imitation Learning. As suggested by the name, we include in RAIL all AIL algorithms that optimize the policy by using an RL algorithm where the reward is given by the discriminator. Thus, GAIL and GAIfO are RAIL algorithms, as are OPOLO and DAC [13]. On the other hand, ValueDICE is not a RAIL algorithm since it optimizes the policy by backpropagating directly into the discriminator, and ASAF is not a RAIL algorithm since it optimizes the policy by simply copying over the weight values from the discriminator.

We characterize RAIL techniques using a modular framework consisting of the following two modules (color coded according to the corresponding design decisions in Figure 1):

- **the RL backbone**
- **the discriminator's input representation**

The first module is the RL algorithm used by a RAIL algorithm to optimize the learner's policy. Early RAIL algorithms like GAIL or GAIfO, used on-policy

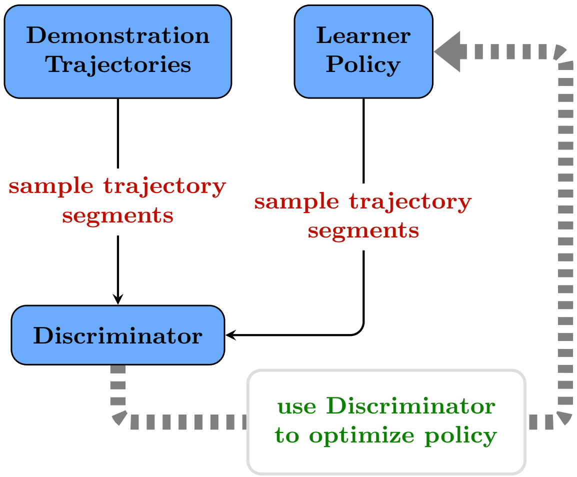

Demonstration Trajectories
Learner Policy
sample trajectory segments
sample trajectory segments
Discriminator
use Discriminator to optimize policy

Fig. 1. Schematic diagram explaining the work-flow in AIL algorithms

RL algorithms such as TRPO [17] or PPO [18]. More recent RAIL algorithms OPOLO and DAC [13] radically reduced their sample complexity by using off-policy RL algorithms such as TD3 [5] or AlgaeDICE [15].

The second module is the input representation used by the discriminator. This module controls whether the discriminator gets access to state-action pairs (s_t, a_t), state-next state pairs (s_t, s_{t+1}), or any arbitrary subsequence of the trajectory. For example, (s_t, s_{t+3}). It is worth pointing out here that most RAIL algorithms either use (s_t, a_t), or (s_t, s_{t+1}) as the input to the discriminator. This last module that we just introduced is an especially powerful one as it allows an RL researcher to seamlessly create an IfO algorithm out of any RAIL algorithm by changing the input representation to the discriminator. For example, by changing (s_t, a_t) to (s_t, s_{t+1}), as was done to create GAIfO from GAIL (Table 1).

Table 1. Recent RAIL algorithms arranged according to our modular characterization of RAIL. Off policy algorithms are denoted by green-colored names. T is the affine transform that is learned by SILEM.

RAIL Algorithm	**RL Backbone**	**Discriminator Input**
GAIL	TRPO [17]	(s_t, a_t)
DAC	TD3 [5]	(s_t, a_t)
GAIfO	TRPO [17]	(s_t, s_{t+1})
OPOLO	AlgaeDICE [15]	(s_t, s_{t+1})
DACfO [22]	TD3 [5]	(s_t, s_{t+1})
SAIfO	SAC [8]	(s_t, s_{t+1})
SILEM [11]	PPO [18]	$T(s_t, s_{t+1}, s_{t+2}, s_{t+3})$

Apart from the two modules defined above, RAIL algorithms also typically incorporate miscellaneous tricks to improve performance such as absorbing states in DAC, and regularization with a dynamics model in OPOLO. We disregard them in our modular characterization due to questions that have arisen regarding their value. The authors of the OPOLO paper observe that the regularization only adds marginal improvements in performance [22], and our own experience has indicated that absorbing states are not necessary for state of the art performance.

4 SAIfO

Given our modular characterization of RAIL algorithms, a natural way to try to obtain a state-of-the-art RAIL algorithm would be to try to maximize the performance of each module independently of one another. Inspired by a recent IfO algorithm, OPOLO [22], we develop a state-of-the-art IfO algorithm by maximizing the performance of the RL backbone. We could also attempt to improve performance by choosing a different form of input to the discriminator that does not include action information, but we leave that to future work.

We maximize performance of the RL backbone by choosing a state-of-the-art off-policy RL algorithm, Soft Actor Critic (SAC) [8], and we term the resulting RAIL algorithm SAC-based Adversarial Imitation from Observation (SAIfO).

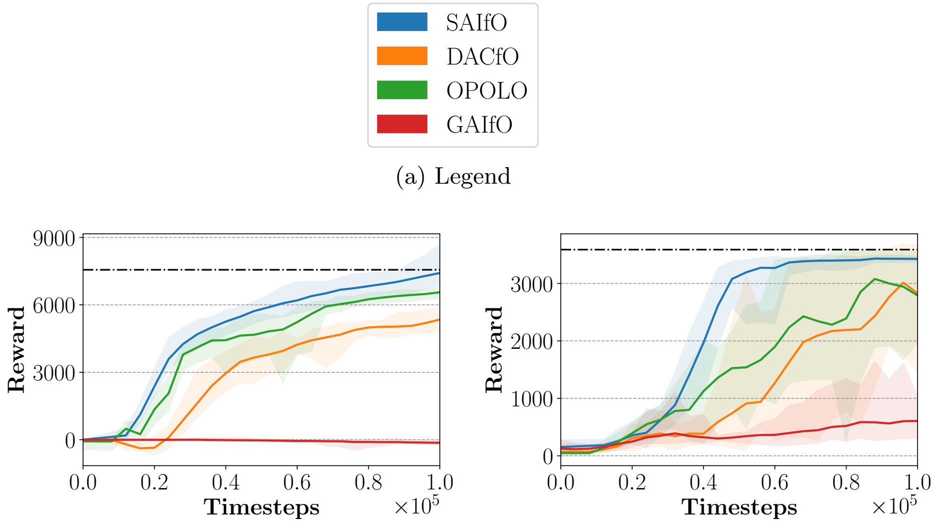
5 Experiments and Results

We run experiments on locomotion tasks provided by OpenAI gym in the Mujoco environment. Our implementation of SAIfO is built on the SAC implementation provided by Spinning Up OpenAI [1]. Every other algorithm we use is based on the implementation provided by the authors of OPOLO [22]. The appendix contains details on the hyperparameters we use in each algorithm along with the neural network architectures we use.

Our main goal with the experiments is to show that our proposed RAIL algorithm, SAIfO, outperforms other RAIL algorithms that perform IfO. We compare SAIfO against GAIfO, OPOLO and DACfO, and find that it indeed does better than the three prior RAIL algorithms (Figure 2). Note that our OPOLO and DACfO results are much better than those reported by Zhu et al. [22] since we strongly optimized all three of our baselines to provide a fair comparison with SAIfO. Refer to the appendix for specific details on how we optimized their implementations of GAIfO, OPOLO and DACfO.

6 SILEM

SAIfO leverages our modular characterization of RAIL to merely accelerate IfO. In SILEM (Skeletal Feature Compensation for Imitation Learning with Embodiment Mismatch) [11], we go a step further and enable IfO to work even in the



(a) Legend

(b) HalfCheetah-v2

(c) Hopper-v2

Fig. 2. Results comparing SAIfO to RAIL algorithms that perform IfO. The horizontal black lines show expert level performance in each domain. The x-axis shows the number of interactions with the environment. Results shown are the mean, minimum and maximum over 10 independent trials.

presence of embodiment mismatch. In SILEM, we process the discriminator's input using a learnable affine transform before inputting it to the discriminator. We train the affine transform to compensate for embodiment mismatch between the learner's trajectories and demonstration trajectories. In Hudson et. al. [11], we evaluate SILEM on a suite of challenging environments involving learning from human demonstrations, and we show that SILEM far outperforms the alternatives.

7 Conclusion

In this work, we introduced a modular characterization of a sub-area in Adversarial Imitation Learning that we term RAIL: Reinforcement-learning-based Adversarial Imitation Learning. We describe the modules apparent in RAIL and show how recent AIL algorithms fit within the modular framework. Using the modularity of the framework as a basis, we develop a new off-policy IfO algorithm (SAIfO) that is faster than prior RAIL algorithms that perform IfO, thus also showing the benefits of the modular characterization.

While developing SAIfO, we only focused on improving performance of the RL backbone. Exploring different formats of the discriminator input is an interesting avenue of future work. For example, rather than using (s_t, s_{t+1}) as an input, using (s_t, s_{t+4}) or $(s_t, s_{t+1} - s_t)$. We also wish to perform more experiments to figure out why SAC makes a much better RL backbone than the alternatives, TD3 and AlgaeDICE. We suspect that the *stochastic update* afforded by the kernel trick plays an important role [8].

Lastly, we also showcased how our modular characterization of RAIL even allowed us to develop an algorithm capable of performing IfO in the presence of embodiment mismatch. We did this by outlining a recent algorithm we created called SILEM [11].

References

1. Achiam, J.: Spinning Up in Deep Reinforcement Learning (2018)
2. Barde, P., Roy, J., Jeon, W., Pineau, J., Pal, C., Nowrouzezahrai, D.: Adversarial soft advantage fitting: Imitation learning without policy optimization. In: Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020), https://proceedings.neurips.cc/paper/2020/hash/9161ab7a1b61012c4c303f10b4c16b2c-Abstract.html
3. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym (2016)
4. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research **12**(7) (2011)
5. Fujimoto, S., van Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 1582–1591. PMLR (2018), http://proceedings.mlr.press/v80/fujimoto18a.html
6. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C., Bengio, Y.: Generative adversarial networks. CoRR **abs/1406.2661** (2014), http://arxiv.org/abs/1406.2661
7. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International Conference on Machine Learning. pp. 1861–1870. PMLR (2018)
8. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 1856–1865. PMLR (2018), http://proceedings.mlr.press/v80/haarnoja18b.html
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
10. Ho, J., Ermon, S.: Generative adversarial imitation learning. In: Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain. pp. 4565–4573 (2016), https://proceedings.neurips.cc/paper/2016/hash/cc7e2b878868cbae992d1fb743995d8f-Abstract.html

11. Hudson, E., Warnell, G., Torabi, F., Stone, P.: Skeletal feature compensation for imitation learning with embodiment mismatch (2021)
12. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6980
13. Kostrikov, I., Agrawal, K.K., Dwibedi, D., Levine, S., Tompson, J.: Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019), https://openreview.net/forum?id=Hk4fpoA5Km
14. Kostrikov, I., Nachum, O., Tompson, J.: Imitation learning via off-policy distribution matching. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net (2020), https://openreview.net/forum?id=Hyg-JC4FDr
15. Nachum, O., Dai, B., Kostrikov, I., Chow, Y., Li, L., Schuurmans, D.: Algaedice: Policy gradient from arbitrary experience. CoRR **abs/1912.02074** (2019), http://arxiv.org/abs/1912.02074
16. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M.S., Berg, A.C., Li, F.: Imagenet large scale visual recognition challenge. CoRR **abs/1409.0575** (2014), http://arxiv.org/abs/1409.0575
17. Schulman, J., Levine, S., Abbeel, P., Jordan, M.I., Moritz, P.: Trust region policy optimization. In: Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015. JMLR Workshop and Conference Proceedings, vol. 37, pp. 1889–1897. JMLR.org (2015), http://proceedings.mlr.press/v37/schulman15.html
18. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR **abs/1707.06347** (2017), http://arxiv.org/abs/1707.06347
19. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada. pp. 3104–3112 (2014), https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html
20. Torabi, F., Warnell, G., Stone, P.: Generative adversarial imitation from observation. CoRR **abs/1807.06158** (2018), http://arxiv.org/abs/1807.06158
21. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA. pp. 5998–6008 (2017), https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html
22. Zhu, Z., Lin, K., Dai, B., Zhou, J.: Off-policy imitation learning from observations. In: Advances in Neural Information Processing Systems. vol. 33, pp. 12402–12413. Curran Associates, Inc. (2020), https://proceedings.neurips.cc/paper/2020/file/92977ae4d2ba21425a59afb269c2a14e-Paper.pdf

8 Appendix

8.1 Ensuring a fair comparison with baselines

We spent a considerable amount of time optimizing our baselines to ensure a fair comparison. We first added a *warmup* phase to the code provided by the authors of OPOLO where we trained the discriminator and Q function for a configurable number of iterations before using them to train the policy with DACfO or OPOLO. We also applied multiple grid searches to find the best configuration of hyperparameters for our three baselines. We found hyperparameter configurations that resulted in a better performance than the default hyperparameters provided by the OPOLO authors.

8.2 Hyperparameters used for each algorithm

For every experiment, we used a grid search to find the best set of hyper-parameters. Specifically, for every configuration of hyper-parameters in the experiment, we ran 10 independent trials. Within each trial, we trained a policy, measured the average reward obtained by that policy over the last 10 tests (with each test run in intervals of 4000 interactions with the environment), and assigned the resulting number as the score for that particular trial. The score for each configuration of hyper-parameters is the average score over the 10 trials corresponding to that configuration. The best configuration of hyper-parameters is then that which maximizes this score. We usually ran multiple successive grid searches to more strongly optimize performance instead of just running one single grid search for experiment.

SAIfO We list below the hyperparameter values we used for our implementation of SAIfO. We only list hyperparameter values that we either introduced or changed from the default value.

Table 2. Hyperparameter settings for HalfCheetah-v2

Hyperparameter name	Value
Discriminator learning rate	6×10^{-5}
Entropy coefficient for discriminator	0
Batch size (SAC)	400
No. steps of warm up for Q function	5000
No. steps of warm up for Discriminator	500
Update discriminator every X iterations	100
SAC alpha	0.02
Gradient penalty for Discriminator	0.0006
Update policy X times every Y iterations	674, 100

Table 3. Hyperparameter settings for Hopper-v2

Hyperparameter name	**Value**
Discriminator learning rate	1×10^{-4}
Entropy coefficient for discriminator	0
Batch size (SAC)	400
No. steps of warm up for Q function	5000
No. steps of warm up for Discriminator	1000
Update discriminator every X iterations	10
SAC alpha	0.15
Gradient penalty for Discriminator	0.002
Update policy X times every Y iterations	674, 100

DACfO We list below the hyperparameter values we used for the implementation of DACfO by the OPOLO authors. We only list hyperparameter values that we either introduced or changed from the default value.

Table 4. Hyperparameter settings for HalfCheetah-v2

Hyperparameter name	**Value**
No. steps of warm up for Q function	5000
No. steps of warm up for Discriminator	200
Update discriminator every X iterations	15
Update policy X times every Y iterations	7000, 1000

Table 5. Hyperparameter settings for Hopper-v2

Hyperparameter name	**Value**
No. steps of warm up for Q function	5000
No. steps of warm up for Discriminator	100
Update discriminator every X iterations	300
Update policy X times every Y iterations	700, 100
Batch size	50

OPOLO We list below the hyperparameter values we used for the implementation of OPOLO by the OPOLO authors. We only list hyperparameter values that we either introduced or changed from the default value.

Table 6. Hyperparameter settings for HalfCheetah-v2

Hyperparameter name	**Value**
No. steps of warm up for Q function	5000
No. steps of warm up for Discriminator	200
Update policy X times every Y iterations	2000, 1000

Table 7. Hyperparameter settings for Hopper-v2

Hyperparameter name	**Value**
No. steps of warm up for Q function	1000
No. steps of warm up for Discriminator	100
Update policy X times every Y iterations	700, 100
Batch size	400
Update Discriminator every X iterations	200

GAIfO We list below the hyperparameter values we used for the implementation of GAIfO by the OPOLO authors. We only list hyperparameter values that we either introduced or changed from the default value.

Table 8. Hyperparameter settings for HalfCheetah-v2

Hyperparameter name	**Value**
No. steps of Conjugate gradient descent	10
No. training steps for value function	5

8.3 Neural network architectures

For every algorithm but SAIfO, we used the default network architecture provided by the authors of OPOLO. For SAIfO, all the deep networks that we used

Table 9. Hyperparameter settings for Hopper-v2

Hyperparameter name	**Value**
No. steps of Conjugate gradient descent	7
No. training steps for value function	3

were (multi layer perceptrons) MLPs with two hidden layers and tanh nonlinearities. The discriminator contained 128 units in each hidden layer, while all the other MLPs contained 256 units each.