ARC - Actor Residual Critic for Adversarial Imitation Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract: Adversarial Imitation Learning (AIL) is a class of popular state-of-the-art Imitation Learning algorithms commonly used in robotics. In AIL, an artificial adversary's misclassification is used as a reward signal that is optimized by any standard Reinforcement Learning (RL) algorithm. Unlike most RL settings, the reward in AIL is *differentiable* but current model-free RL algorithms do not make use of this property to train a policy. In contrast, we leverage the differentiability property of the AIL reward function and formulate a class of Actor Residual Critic (ARC) RL algorithms. ARC algorithms draw a parallel to the standard Actor-Critic (AC) algorithms in RL literature and uses a residual critic, C function (instead of the standard Q function) to approximate only the discounted future return (excluding the immediate reward). ARC algorithms have similar convergence properties as the standard AC algorithms with the additional advantage that the gradient through the immediate reward is exact. For the discrete (tabular) case with finite states, actions, and known dynamics, we prove that policy iteration with C function converges to an optimal policy. In the continuous case with function approximation and unknown dynamics, we experimentally show that ARC aided AIL outperforms standard AIL in simulated continuous-control and real robotic manipulation tasks. ARC algorithms are simple to implement and can be incorporated into any existing AIL implementation with an AC algorithm.

Keywords: Adversarial Imitation Learning (AIL), Actor-Critic (AC), Actor Residual Critic (ARC)

1 Introduction

Although Reinforcement Learning (RL) allows us to train agents to perform complex tasks without manually designing controllers [1, 2, 3], it is often tedious to hand-craft a dense reward function that captures the task objective in robotic tasks [4, 5, 6]. Imitation Learning (IL) or Learning from Demonstration (LfD) is a popular choice in such situations [4, 5, 6, 7]. Common approaches to IL are Behavior Cloning (BC) [8] and Inverse Reinforcement Learning (IRL) [9].

Within IRL, recent Adversarial Imitation Learning (AIL) algorithms have shown state-of-the-art performance, especially in continuous control tasks which make them relevant to real-world robotics problems. AIL methods cast the IL problem as an adversarial game between a policy and a learned adversary (discriminator). The adversary aims to classify between agent and expert trajectories and the policy is trained using the adversary's mis-classification as the reward function. This encourages the policy to imitate the expert. Popular AIL algorithms include Generative Adversarial Imitation Learning (GAIL) [10], Adversarial Inverse Reinforcement Learning (AIRL) [11] and f-MAX [12].

The agent in AIL is trained with any standard RL algorithm. There are two popular categories of RL algorithms: (i) on-policy algorithms such as TRPO [13], PPO [2], GAE [14] based on the policy gradient theorem [15, 16]; and (ii) off-policy Actor-Critic (AC) algorithms such as DDPG [17], TD3 [18], SAC [3] that compute the policy gradient through a critic (Q function). These standard RL algorithms were designed for arbitrary scalar reward functions; and they compute an *approximate* gradient for updating the policy. Practical on-policy algorithms based on the policy gradient theorem use several approximations to the true gradient [13, 2, 14] and off-policy AC

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

algorithms first approximate policy return with a critic (Q function) and subsequently compute the gradient through this critic [17, 18, 3]. Even if the Q function is approximated very accurately, the error in its gradient can be arbitrarily large, Appendix A.1.

Our insight is that the reward function in AIL has a special property, it is *differentiable* which means we can compute the exact gradient through the reward function instead of approximating it. As we will see in section 3, naively computing the gradient through reward function would lead to a short-sighted sub-optimal policy. To address this issue, we formulate a class of Actor Residual Critic (ARC) RL algorithms that use a residual critic, C function (instead of the standard Q function) to approximate only the discounted future return (excluding immediate reward).

The contribution of this paper is the introduction of ARC, which can be easily incorporated to replace the AC algorithm in any existing AIL algorithm for continuous-control and helps boost the asymptotic performance by computing the exact gradient through the reward function.

2 Related Work

Algorithm	Minimized f-Divergence Name	Minimized f-Divergence Expression	$r(s,a)$
GAIL [10]	Jensen-Shannon	$\frac{1}{2}\left\{\mathbb{E}_{\rho^{\text{exp}}} \log \frac{2\rho^{\text{exp}}}{\rho^{\text{exp}}+\rho^{\pi}} + \mathbb{E}_{\rho^{\pi}} \log \frac{2\rho^{\pi}}{\rho^{\text{exp}}+\rho^{\pi}}\right\}$	$\log D(s,a)$
AIRL [11], f-MAX-RKL [12]	Reverse KL	$\mathbb{E}_{\rho^{\pi}} \log \frac{\rho^{\pi}}{\rho^{\text{exp}}}$	$\log \frac{D(s,a)}{1-D(s,a)}$

Table 1: Popular AIL algorithms, f-divergence metrics they minimize and their reward functions.

The simplest approach to imitation learning is Behavior Cloning [8] where an agent policy directly regresses on expert actions (but not states) using supervised learning. This leads to distribution shift and poor performance at test time [19, 10]. Methods such as DAgger [19] and Dart [20] eliminate this issue but assume an interactive access to an expert policy, which is often impractical.

Inverse Reinforcement Learning (IRL) approaches recover a reward function which can be used to train an agent using RL [9, 21] and have been more successful than BC. Within IRL, recent Adversarial Imitation Learning (AIL) methods inspired by Generative Adversarial Networks (GANs) [22] have been extremely successful. GAIL [10] showed state-of-the-art results in imitation learning tasks following which several extensions have been proposed [23, 24]. AIRL [11] imitates an expert as well as recovers a robust reward function. [25] and [12] presented a unifying view on AIL methods by showing that they minimize different divergence metrics between expert and agent state-action distributions but are otherwise similar. [12] also presented a generalized AIL method f-MAX which can minimize any specified f-divergence metric [26] between expert and agent state-action distributions thereby imitating the expert. Choosing different divergence metrics leads to different AIL algorithms, e.g. choosing Jensen-Shannon divergence leads to GAIL [10]. [27] proposed a method that automatically learns a f-divergence metric to minimize. Our proposed Actor Residual Critic (ARC) can be augmented with any of these AIL algorithms to leverage the reward gradient.

Some recent methods have leveraged the differentiable property of reward in certain scenarios but they have used this property in very different settings. [28] used the gradient of the reward to improve the reward function but not to optimize the policy. We on the other hand explicitly use the gradient of the reward to optimize the policy. [29] used the gradient through the reward to optimize the policy but operated in the model-based setting. Differentiable cost (negative reward) has also been leveraged in control literature for a long time to compute a policy, e.g. in LQR [30] and its extensions; but they assume access to a known dynamics model. We on the other hand present a model-free method with unknown dynamics that uses the gradient of the reward to optimize the policy with the help of a new class of RL algorithms called Actor Residual Critic (ARC).

3 Background

Objective Our goal is to imitate an expert from one or more demonstrated trajectories (state-action sequences) in a continuous-control task (state and action spaces are continuous). Given any Adversarial Imitation Learning (AIL) algorithm that uses an off-policy Actor-Critic algorithm RL algo-

rithm, we wish to use our insight on the availability of a differentiable reward function to improve the imitation learning algorithm.

Notation The environment is modeled as a Markov Decision Process (MDP) represented as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition dynamics $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$, reward function $r(s, a)$, initial state distribution $\rho_0(s)$, and discount factor γ. $\pi(.|s)$, $\pi^{\text{exp}}(.|s)$ denote policies and $\rho^{\pi}, \rho^{\text{exp}} : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ denote state-action occupancy distributions for agent and expert respectively. $\mathcal{T} = \{s_1, a_1, s_2, a_2, \ldots, s_T, a_T\}$ denotes a trajectory or episode and (s, a, s', a') denotes a continuous segment in a trajectory. A discriminator or adversary $D(s, a)$ tries to determine whether the particular (s, a) pair belongs to an expert trajectory or agent trajectory, i.e. $D(s, a) = P(\text{expert}|s, a)$. The optimal discriminator is $D(s, a) = \frac{\rho^{\text{exp}}(s,a)}{\rho^{\text{exp}}(s,a)+\rho^{\pi}(s,a)}$ [22].

Adversarial Imitation Learning (AIL) In AIL, the discriminator and agent are alternately trained. The discriminator is trained to maximize the likelihood of correctly classifying expert and agent data using supervised learning, (1) and the agent is trained to maximize the expected discounted return, (2).

$$\max_{D} \left\{ \mathbb{E}_{s,a\sim\rho^{\text{exp}}}[\log D(s, a)] + \mathbb{E}_{s,a\sim\rho^{\pi}} \left[\log(1 - D(s, a))\right] \right\} \tag{1}$$

$$\max_{\pi} \left\{ \mathbb{E}_{s,a\sim\rho_0,\pi,\mathcal{P}} \sum_{t\geq 0} \gamma^t r(s_t, a_t) \right\} \tag{2}$$

Here, reward $r_{\psi}(s, a) = h(D_{\psi}(s, a))$ is a function of the discriminator which varies between different AIL algorithms. Different AIL algorithms minimize different f-divergence metrics between expert and agent state-action distribution. Defining a f-divergence metric instantiates different reward functions [12]. Some popular divergence choices are Jensen-Shannon in GAIL [10] and Reverse Kullback-Leibler in f-MAX-RKL [12] and AIRL [11] as shown in Table 1.

Any RL algorithm could be used to optimize (2) and popular choices are off-policy Actor-Critic algorithms such as DDPG [17], TD3 [18], SAC [3] and on-policy algorithms such as TRPO [13], PPO [2], GAE [14] which are based on the policy gradient theorem [15, 16]. We focus on off-policy Actor-Critic algorithms as they are usually more sample efficient and stable than on-policy policy gradient algorithms [18, 3].

Continuous-control using off-policy Actor-Critic The objective in off-policy RL algorithms is to maximize expected Q function of the policy, Q^{π} averaged over the state distribution of a dataset $\mathcal{D}$ (typically past states stored in buffer) and the action distribution of the policy π [31]:

$$\max_{\pi} \mathbb{E}_{s\sim\mathcal{D},a\sim\pi} Q^{\pi}(s, a) \tag{3}$$

$$\text{where, } Q^{\pi}(s, a) = \mathbb{E}_{s,a\sim\rho_0,\pi,\mathcal{P}} \left[\sum_{k\geq 0} \gamma^k r_{t+k} \middle| s_t = s, a_t = a \right] \tag{4}$$

The critic and the policy denoted by Q, π respectively are approximated by function approximators such as neural networks with parameters ϕ and θ respectively. There is an additional target $Q_{\phi_{\text{targ}}}$ function parameterized by ϕ_{targ}. There are two alternating optimization steps:

1. Policy evaluation: Fit critic (Q_{ϕ} function) by minimizing Bellman Backup error.

$$\min_{\phi} \mathbb{E}_{s,a,s'\sim\mathcal{D}} \left\{ Q_{\phi}(s, a) - y(s, a) \right\}^2 \tag{5}$$

$$\text{where, } y(s, a) = r(s, a) + \gamma Q_{\phi_{\text{targ}}}(s', a') \text{ and } a' \sim \pi_{\theta}(.|s') \tag{6}$$

   Q_{ϕ} is updated with gradient descent without passing gradient through the target $y(s, a)$.

2. Policy improvement: Update policy with gradient ascent over RL objective.

$$\mathbb{E}_{s\sim\mathcal{D}} \left[\nabla_{\theta} Q_{\phi}(s, a \sim \pi_{\theta}(.|s)) \right] \tag{7}$$

All off-policy Actor Critic algorithms follow the core idea above ((5) and (7)) along with additional details such as the use of a deterministic policy and target network in DDPG [17], double Q networks and delayed updates in TD3 [18], entropy regularization and reparameterization trick in SAC [3].

Naive-Diff and why it won't work Realizing that the reward in AIL is differentiable, we can formulate a Naive-Diff RL algorithm that updates the policy by differentiating the RL objective (2) with respect to the policy parameters θ.

$$\mathbb{E}_{\mathcal{T}\sim\mathcal{D}}\left[\nabla_\theta r(s_1,a_1)+\gamma\nabla_\theta r(s_2,a_2)+\gamma^2\nabla_\theta r(s_3,a_3)+\dots\right] \tag{8}$$

$\mathcal{T}=\{s_1,a_1,s_2,a_2\dots\}$ is a sampled trajectory in $\mathcal{D}$. Using standard autodiff packages such as Pytorch [32] or Tensorflow [33] to naively compute the gradients in (8) would produce incorrect gradients. Apart from the immediate reward $r(s_1,a_1)$, all the terms depend on the transition dynamics of the environment $\mathcal{P}(s_{t+1}|s_t,a_t)$, which is unknown and we cannot differentiate through it. So, autodiff will calculate the gradient of only immediate reward correctly and calculate the rest as 0's. This will produce a short-sighted sub-optimal policy that maximizes only the immediate reward.

4 Method

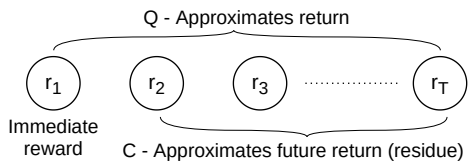


Figure 1: Visual illustration of approximating reward via Q function or C function.

The main lesson we learnt from Naive-Diff is that while we can obtain the gradient of immediate reward, we cannot directly obtain the gradient of future return due to unknown environment dynamics. This directly motivates our formulation of Actor Residual Critic (ARC). Standard Actor Critic algorithms use Q function to approximate the return as described in Eq. 4. However, since we can directly obtain the gradient of the reward, we needn't approximate it with a Q function. We, therefore, propose to use C function to approximate only the future return, leaving out the immediate reward. This is the core idea behind Actor Residual Critic (ARC) and is highlighted in Fig. 1. The word "Residual" refers to the amount of return that remains after subtracting the immediate reward from the return. As we will see in Section 4.3, segregating the immediate reward from future return will allow ARC algorithms to leverage the exact gradient of the reward. We now formally describe Residual Critic (C function) and its relation to the standard critic (Q function) in RL literature.

4.1 Definition of Residual Critic (C function)

The Q function under a policy π, $Q^\pi(s,a)$, is defined as the expected discounted return from state s taking action a, (9). The C function under a policy π, $C^\pi(s,a)$, is defined as the expected discounted future return, excluding the immediate reward (10). Note that the summation in (10) starts from 1 instead of 0. Q function can be expressed in terms of C function as shown in (11).

$$Q^\pi(s,a)=\mathbb{E}_{s,a\sim\rho_0,\pi,\mathcal{P}}\left[\sum_{k\geq 0}\gamma^k r_{t+k}\middle| s_t=s,a_t=a\right] \tag{9}$$

$$C^\pi(s,a)=\mathbb{E}_{s,a\sim\rho_0,\pi,\mathcal{P}}\left[\sum_{k\geq 1}\gamma^k r_{t+k}\middle| s_t=s,a_t=a\right] \tag{10}$$

$$Q^\pi(s,a)=r(s,a)+C^\pi(s,a) \tag{11}$$

4.2 Policy Iteration using C function

Algorithm 1: Policy Iteration with C function

Initialize $C^0(s,a)\forall s,a$;
while π *not converged* **do**
  // Policy evaluation
  for *n=1,2,... until* C_k *converges* **do**
    $C^{n+1}(s,a)\leftarrow\gamma\sum_{s'}P(s'|s,a)\sum_{a'}\pi(a'|s')\left(r(s',a')+C^n(s',a')\right)\quad\forall s,a$
  // Policy improvement
  $\pi(s,a)\leftarrow\begin{cases}1, & \text{if } a=\text{argmax}_{a'}\left(r(s,a')+C(s,a')\right)\\ 0, & \text{otherwise}\end{cases}\quad\forall s,a$

Theorem 1. *The policy iteration algorithm defined by Algorithm 1 converges to the optimal C^* function and an optimal policy π^*.*

Proof. Appendix B.4. □

Using C function, we can formulate a Policy Iteration algorithm as shown in Algorithm 1, which is guaranteed to converge to an optimal policy (Theorem 1), similar to the case of Policy Iteration with Q or V function [16]. Other properties of C function and proofs are presented in Appendix B.

4.3 Continuous-control using Actor Residual Critic

We can easily extend the policy iteration algorithm with C function (Algorithm 1) for continuous-control tasks using function approximators instead of discrete C values and a discrete policy (similar to the case of Q function [16]). We call any RL algorithm that uses a policy, π and a residual critic, C function as an Actor Residual Critic (ARC) algorithm. Using the specific details of different existing Actor Critic algorithms, we can formulate analogous ARC algorithms. For example, using a deterministic policy and target network as in [17] we can get ARC-DDPG. Using double C networks (instead of Q networks) and delayed updates as in [18] we can get ARC-TD3. Using entropy regularization and reparameterization trick as in [3] we can get ARC-SAC or SARC (Soft Actor Residual Critic).

4.4 ARC aided Adversarial Imitation Learning

To incorporate ARC in any Adversarial Imitation Learning algorithm, we simply replace the Actor Critic RL algorithm with an ARC RL algorithm without altering anything else in the pipeline. For example, we can replace SAC [3] with SARC to get SARC-AIL as shown in Algorithm 2. Implementation-wise this is extremely simple and doesn't require any additional functional parts in the algorithm. The same neural network that approximated Q function can be now be used to approximate C function.

Algorithm 2: SARC-AIL: Soft Actor Residual Critic Adversarial Imitation Learning

Intialization: Environment (env), Discriminator parameters ψ, Policy parameters θ, C-function parameters ϕ_1, ϕ_2, dataset of expert demonstrations $\mathcal{D}^{\text{exp}}$, replay buffer $\mathcal{D}$, Target parameters $\phi_{\text{targ1}} \leftarrow \phi_1$, $\phi_{\text{targ2}} \leftarrow \phi_2$, Entropy regularization coefficient α;

while *Max no. of environment interactions is not reached* **do**

- $a \sim \pi_\theta(.|s)$;
- $s', r, d =$ env.step(a); $d = 1$ if s' is terminal state, 0 otherwise
- Store (s, a, s', d) in replay buffer $\mathcal{D}$;
- **if** *Update interval reached* **then**
  - **for** *no. of update steps* **do**
    - Sample batch $B = (s, a, s', d) \sim \mathcal{D}$;
    - Sample batch of expert demonstrations $B^{\text{exp}} = (s, a) \sim \mathcal{D}^{\text{exp}}$;
    - Update Discriminator parameters (ψ) with gradient ascent.
      $$\nabla_\psi \Big\{ \textstyle\sum_{(s,a)\in B^{\text{exp}}} [\log D_\psi(s,a)] + \sum_{(s,a,s',d)\in B} [\log(1 - D_\psi(s,a))] \Big\};$$
    - Compute C targets $\forall (s, a, s', d) \in B$
      $$y(s,a,d) = \gamma \left(r_\psi(s', \tilde{a}') + \min_{i=1,2} C_{\phi_{\text{targ}i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s') \right), \quad \tilde{a}' \sim \pi_\theta(.|s'), r_\psi(s', \tilde{a}') = h(D_\psi(s,', \tilde{a}'))$$
    - Update C-functions parameters (ϕ_1, ϕ_2) with gradient descent.
      $$\nabla_{\phi_i} \frac{1}{|B|} \textstyle\sum_{(s,a,s',d)\in B} \left(C_{\phi_i}(s,a) - y(s,a,d) \right)^2, \quad \text{for } i = 1, 2$$
    - Update policy parameters (θ) with gradient ascent.
      $$\nabla_\theta \frac{1}{|B|} \textstyle\sum_{s\in B} \left(r_\psi(s, \tilde{a}) + \min_{i=1,2} C_{\phi_i}(s, \tilde{a}) - \alpha \log \pi_\theta(\tilde{a}|s) \right), \quad \tilde{a} \sim \pi_\theta(.|s), r_\psi(s, \tilde{a}) = h(D_\psi(s, \tilde{a}))$$
    - Update target networks.
      $$\phi_{\text{targ}i} \leftarrow \zeta \phi_{\text{targ}i} + (1 - \zeta)\phi_i, \quad \text{for } i = 1, 2; \quad \zeta \text{ controls polyak averaging}$$

4.5 Why choose ARC over Actor-Critic in Adversarial Imitation Learning?

The advantage of using an ARC algorithm over an Actor-Critic (AC) algorithm is that we can leverage the exact gradient of the reward. Standard AC algorithms use Q_ϕ to approximate the immediate reward + future return and then compute the gradient of the policy parameters through the Q_ϕ function (12). This is an approximate gradient with no bound on the error in gradient, since the Q_ϕ function is an estimated value, Appendix A.1. On the other hand, ARC algorithms segregate the immediate reward (which is known in Adversarial Imitation Learning) from the future return (which needs to be estimated). ARC algorithms then compute the gradient of policy parameters through the immediate reward (which is exact) and the C function (which is approximate) separately (13).

$$\text{Standard AC} \quad \mathbb{E}_{s\sim\mathcal{D}}\big[\nabla_\theta Q_\phi(s,a)\big], \qquad a \sim \pi_\theta(.|s) \tag{12}$$

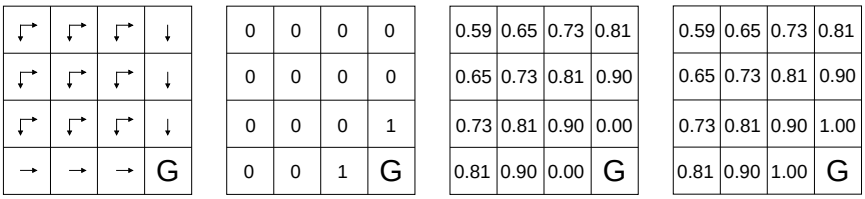
$$\text{ARC (Our)} \quad \mathbb{E}_{s\sim\mathcal{D}}\big[\nabla_\theta r(s,a) + \nabla_\theta C_\phi(s,a)\big], \qquad a \sim \pi_\theta(.|s) \tag{13}$$

Because ARC uses the exact gradient of the immediate reward, it produces a more accurate gradient as explained in Appendix A.2 and experimentally verified in Appendix D.2.

5 Results

In Theorem 1, we proved that Policy Iteration with C function converges to an optimal policy. In Fig. 2, we experimentally validate this on an example grid world. The complete details are presented in Appendix E.1. In the following sections (5.2, 5.3 and 5.4) we show the effectiveness of ARC aided AIL in Mujoco continuous-control tasks, and simulated and real robotic manipulation tasks. In Appendix D.2, we experimentally illustrate that ARC produces more accurate gradients than AC using a simple 1D driving environment.

5.1 Policy Iteration on a Grid World



(a) π^* (b) r^* (c) C^* (d) $Q^* = r^* + C^*$

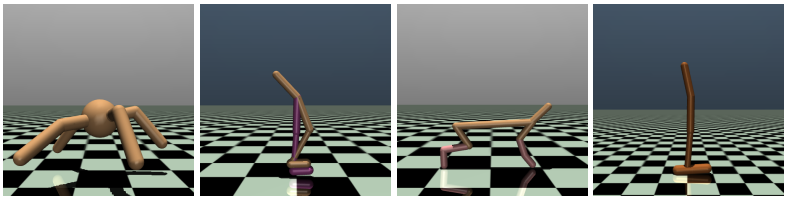
Figure 2: On a Grid World, the results of running two Policy Iteration (PI) algorithms - PI with C function (Algorithm 1) and the standard PI with Q function (Appendix C.1 Algorithm 3). Both algorithms converge in 7 policy improvement steps to the same optimal policy π^* as shown in a. The optimal policy gets the immediate reward shown shown b. The C values c at the convergence of PI with C function and the Q values d at the convergence of PI with Q function are consistent with their relation $Q^* = r^* + C^*$ (11). Additional details are presented in Appendix E.1.

5.2 Imitation Learning in Mujoco continuous-control tasks

We used 4 Mujoco continuous-control environments from OpenAI Gym [34], as shown in Fig. 3. We evaluated the benefit of using ARC with two popular Adversarial Imitation Learning (AIL) algorithms, f-MAX-RKL [12] and GAIL [10]. For each of these algorithms, we evaluated the performance of standard AIL algorithms (f-MAX-RKL, GAIL), ARC aided AIL algorithms (ARC-f-MAX-RKL, ARC-GAIL) and Naive-Diff algorithm described in Section 3 (Naive-Diff-f-MAX-RKL, Naive-Diff-GAIL). We also evaluated the performance of Behavior Cloning (BC). Further experimental details are presented in Appendix E. Fig. 4 shows the training plots and Table 2 shows the final performance of the different algorithms. Across all environments and across both the AIL algorithms, incorporating ARC shows consistent improvement over standard AIL algorithms (Table 2). BC suffers from distribution shift at test time [19, 10] and performs very poorly. As we predicted in Section 3, Naive-Diff algorithms don't perform well as naively using autodiff doesn't compute the gradients correctly.

5.3 Imitation Learning in robotic manipulation tasks

We used simplified 2D versions of FetchReach (Fig. 5a) and FetchPush (Fig. 5b) robotic manipulation tasks from OpenAI Gym [34] which have a simulated Fetch robot, [35]. In the FetchReach task, the robot needs to take it's end-effector to the goal (virtual red sphere) as quickly as possible.



(a) Ant-v2 (b) Walker-v2 (c) HalfCheetah-v2 (d) Hopper-v2

Figure 3: OpenAI Gym's [34] Mujoco continuous-control environments used for evaluation.

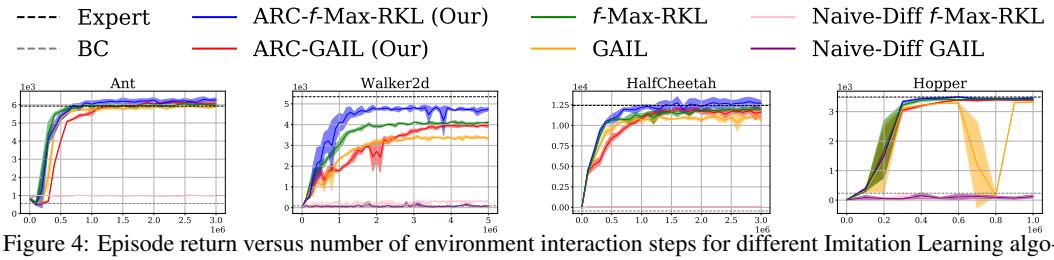


Figure 4: Episode return versus number of environment interaction steps for different Imitation Learning algorithms on Mujoco continuous-control environments.

Method	Ant	Walker2d	HalfCheetah	Hopper
Expert return	5926.18 $\pm$ 124.56	5344.21 $\pm$ 84.45	12427.49 $\pm$ 486.38	3592.63 $\pm$ 19.21
ARC-f-Max-RKL (Our)	**6275.08** $\pm$ 112.61	**4741.28** $\pm$ 94.49	**12698.38** $\pm$ 330.67	**3453.28** $\pm$ 37.95
f-Max-RKL	5983.96 $\pm$ 80.94	4107.55 $\pm$ 61.16	12003.33 $\pm$ 183.04	3410.09 $\pm$ 12.6
Naive-Diff f-Max-RKL	997.65 $\pm$ 3.31	320.48 $\pm$ 7.7	-0.48 $\pm$ 0.03	146.43 $\pm$ 34.83
ARC-GAIL (Our)	**6048.67** $\pm$ 126.97	**3941.39** $\pm$ 83.72	**11615.09** $\pm$ 555.05	**3385.28** $\pm$ 15.92
GAIL	5895.09 $\pm$ 10.44	3318.8 $\pm$ 48.55	10814.69 $\pm$ 323.91	3322.05 $\pm$ 8.31
Naive-Diff GAIL	995.97 $\pm$ 2.13	55.63 $\pm$ 29.04	-0.46 $\pm$ 0.02	126.3 $\pm$ 51.55
BC	559.76 $\pm$ 129.82	87.51 $\pm$ 120.39	-444.5 $\pm$ 44.67	233.83 $\pm$ 84.82

Table 2: Policy return on Mujoco environments using different Adversarial Imitation Learning algorithms. Each algorithms is run with 5 random seeds and each seed is evaluated for 20 episodes.

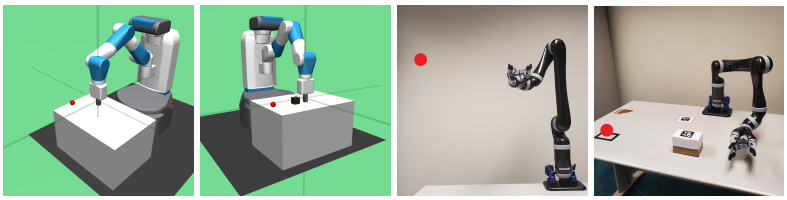
In the FetchPush task, the robot's needs to push the block to the goal as quickly as possible. Further details are presented in Appendix E.3.

Fig. 6a shows the training plots and Table 3 under the heading 'Simulation' shows the final performance of the different algorithms. In both the FetchReach and FetchPush tasks, ARC aided AIL algorithms consistently outperformed the standard AIL algorithms. Fig. 6b shows the magnitude of the 2nd action dimension vs. time-step in one episode for different algorithms. The expert initially executed large actions when the end-effector/block was far away from the goal. As the end-effector/block approached the goal, the expert executed small actions. ARC aided AIL algorithms (ARC-f-Max-RKL and ARC-GAIL) showed a similar trend while standard AIL algorithms (f-Max-RKL and GAIL) learnt a nearly constant action. Thus, ARC aided AIL algorithms were able to better imitate the expert than standard AIL algorithms.

5.4 Sim-to-real transfer of robotic manipulation policies

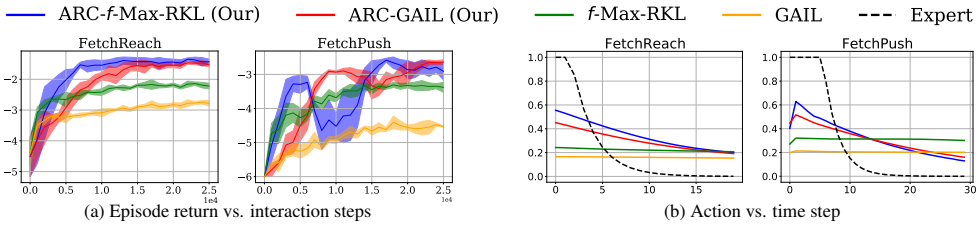
For testing the sim-to-real transfer of the different trained AIL manipulation policies, we setup JacoReach (Fig. 5c) and JacoPush (Fig. 5d) tasks with a Kinova Jaco Gen 2 arm, similar to the FetchReach and FetchPush tasks in the previous section. The details are presented in Appendix E.4.

Table 3 under the heading 'Real Robot' shows the performance of the different AIL algorithms in the real robotic manipulation tasks. The real robot evaluations showed a similar trend as in the simulated tasks. ARC aided AIL consistently outperformed the standard AIL algorithms. Appendix D Fig. 9 visualizes the policies in the JacoPush task showing that ARC aided AIL algorithms were able to push the block closer to the goal as compared to the standard AIL algorithms. Supplementary slide shows videos of the same.



(a) FetchReach (b) FetchPush (c) JacoReach (d) JacoPush

Figure 5: Simulated and real robotic manipulation tasks used for evaluation. Simplified 2D versions of the FetchReach a and FetchPush b tasks from OpenAI Gym, [34] with a Fetch robot, [35]. Corresponding JacoReach c and JacoPush d tasks with a real Kinova Jaco Gen 2 arm, [36].



(a) Episode return vs. interaction steps (b) Action vs. time step

Figure 6: a Episode return vs. number of environment interaction steps for different Adversarial Imitation Learning algorithms on FetchPush and FetchReach tasks. b Magnitude of the 2nd action dimension versus time step in a single episode for different algorithms.

Method	Simulation		Real Robot	
	FetchReach	FetchPush	JacoReach	JacoPush
Expert return	-0.58 $\pm$ 0	-1.18 $\pm$ 0.04	-0.14 $\pm$ 0.01	-0.77 $\pm$ 0.01
ARC-f-Max-RKL (Our)	**-1.43** $\pm$ 0.08	**-2.91** $\pm$ 0.25	**-0.38** $\pm$ 0.02	**-1.25** $\pm$ 0.06
f-Max-RKL	-2.22 $\pm$ 0.09	-3.38 $\pm$ 0.15	-0.8 $\pm$ 0.05	-2.03 $\pm$ 0.06
ARC-GAIL (Our)	**-1.53** $\pm$ 0.06	**-2.64** $\pm$ 0.07	**-0.46** $\pm$ 0.01	**-1.56** $\pm$ 0.08
GAIL	-2.78 $\pm$ 0.09	-4.53 $\pm$ 0.01	-1.05 $\pm$ 0.06	-2.35 $\pm$ 0.06

Table 3: Policy return on simulated (FetchReach, FetchPush) and real (JacoReach, JacoPush) robotic manipulation tasks using different AIL algorithms. The reward at each time step is negative distance between end-effector & goal for reach tasks and block & goal for push tasks. The reward in the real and simulated tasks are on different scales due to implementation details described in Appendix E.4.

6 Limitations

Three main limitations in our work are: (1) While many AIL algorithms can be trained using expert 'states' only, ARC-AIL can only be trained with 'state-action' pairs. We believe this isn't very limiting because we typically have access to both states and actions in expert demonstrations. (2) ARC-AIL can only work with continuous action space. Again, most real world robotic tasks have or can be modified to have a continuous action space. (3) We haven't explored how the agent-adversary interaction in AIL affects the accuracy of the reward gradient and leave that for future work.

7 Conclusion

We highlighted that the reward in popular Adversarial Imitation Learning (AIL) algorithms are differentiable but this property has not been leveraged by existing model-free RL algorithms to train a policy. We also showed that naively differentiating the policy through this reward function does not perform well. To solve this issue, we proposed a class of Actor Residual Critic (ARC) RL algorithms that use a C function as an alternative to standard Actor Critic (AC) algorithms which use a Q function. An ARC algorithm can replace the AC algorithm in any existing AIL algorithm. We formally proved that Policy Iteration using C function converges to an optimum policy in tabular environments. For continuous-control tasks, using ARC can compute the exact gradient of the policy through the reward function which helps improve the performance of the AIL algorithms in simulated continuous-control and simulated & real robotic manipulation tasks. Future work can explore the applicability of ARC algorithm to other scenarios which have a differentiable reward function.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[3] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[4] C. G. Atkeson and S. Schaal. Robot learning from demonstration. In *ICML*, volume 97, pages 12–20. Citeseer, 1997.

[5] S. Schaal. Learning from demonstration. In *Advances in neural information processing systems*, pages 1040–1046, 1997.

[6] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[7] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[8] M. Bain and C. Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129, 1995.

[9] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[10] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[11] J. Fu, K. Luo, and S. Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

[12] S. K. S. Ghasemipour, R. Zemel, and S. Gu. A divergence minimization perspective on imitation learning methods. In *Conference on Robot Learning*, pages 1259–1277. PMLR, 2020.

[13] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[14] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[15] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[16] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[18] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[19] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[20] M. Laskey, J. Lee, W. Hsieh, R. Liaw, J. Mahler, R. Fox, and K. Goldberg. Iterative noise injection for scalable imitation learning. In *1st conference on robot learning (CoRL),(ed., Sergey Levine and Vincent Vanhoucke and Ken Goldberg), Mountain View, CA, USA*, pages 13–15, 2017.

[21] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[22] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets.

[23] Y. Li, J. Song, and S. Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In *NIPS*, 2017.

[24] R. Jena, C. Liu, and K. Sycara. Augmenting gail with bc for sample efficient imitation learning. In *Conference on Robot Learning*, pages 80–90. PMLR, 2021.

[25] L. Ke, M. Barnes, W. Sun, G. Lee, S. Choudhury, and S. Srinivasa. Imitation learning as f-divergence minimization. *arXiv preprint arXiv:1905.12888*, 2019.

[26] J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

[27] X. Zhang, Y. Li, Z. Zhang, and Z.-L. Zhang. f-gail: Learning f-divergence for generative adversarial imitation learning. *arXiv preprint arXiv:2010.01207*, 2020.

[28] T. Ni, H. Sikchi, Y. Wang, T. Gupta, L. Lee, and B. Eysenbach. f-irl: Inverse reinforcement learning via state marginal matching. In *Conference on Robot Learning*, pages 529–551. PMLR, 2021.

[29] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2019.

[30] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[31] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[33] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[34] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[35] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich. Fetch and freight: Standard platforms for service robot applications.

[36] A. Campeau-Lecours, H. Lamontagne, S. Latour, P. Fauteux, V. Maheu, F. Boucher, C. Deguire, and L.-J. C. L'Ecuyer. Kinova modular robot arms for service robotics applications. In *Rapid Automation: Concepts, Methodologies, Tools, and Applications*, pages 693–719. IGI global, 2019.