

ARC - Actor Residual Critic for Adversarial Imitation Learning

Ankur Deka^{†*}, Changliu Liu[†], Katia Sycara[†]

[†]Robotics Institute, Carnegie Mellon University ^{*}Intel Labs
adeka@alumni.cmu.edu, {cliu6, katia}@cs.cmu.edu

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. The reward in AIL is also *shaped* since it comes from an adversary. We leverage the differentiability property of the shaped 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. Video and link to code are available at: sites.google.com/view/actor-residual-critic.

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

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 2 special properties: (i) it is *differentiable* which means we can compute the exact gradient through the reward function instead of approximating it and (ii) it is dense/shaped as it comes from an adversary. 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 shaped reward function.

2 Related Work

Algorithm	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. If we have access to a differentiable dynamics model, we can directly obtain the gradient of the expected return (policy objective) w.r.t. the policy parameters, Appendix E.5. Since we can directly obtain the objective’s gradient, we do not necessarily need to use either a critic (Q) as in standard Actor Critic (AC) algorithms or a residual critic (C) as in our proposed Actor Residual Critic (ARC) algorithms. 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 Adver-

serial Imitation Learning (AIL) algorithm that uses an off-policy Actor-Critic algorithm RL algorithm, 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 . $\pi(\cdot|s)$, $\pi^{\text{exp}}(\cdot|s)$ 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. $\mathcal{T} = \{s_1, a_1, s_2, a_2, \dots, s_T, a_T\}$ 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).

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

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

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]:

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

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

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.

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

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

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.

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

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 and shaped, 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 .

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

$\mathcal{T} = \{s_1, a_1, s_2, a_2, \dots\}$ 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

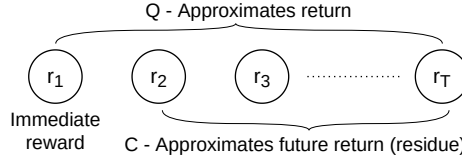


Figure 1: Visual illustration of approximating reward via Q function or C function.

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 shaped reward. We now formally describe Residual Critic (C function) and its relation to the standard critic (Q function).

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).

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

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

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

4.2 Policy Iteration using C function

Algorithm 1: Policy Iteration with C function

```

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

```

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

Initialization: Environment (env), Discriminator parameters ψ , Policy parameters θ , C -function parameters ϕ_1, ϕ_2 , dataset of expert demonstrations \mathcal{D}^{exp} , replay buffer \mathcal{D} , Target parameters $\phi_{\text{targ}1} \leftarrow \phi_1, \phi_{\text{targ}2} \leftarrow \phi_2$, Entropy regularization coefficient α ;

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

```

     $a \sim \pi_\theta(\cdot|s)$ ;
     $s', r, d = \text{env.step}(a)$ ;  $d = 1$  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 ( $\psi$ ) with gradient ascent.
             $\nabla_\psi \left\{ \sum_{(s,a) \in B^{\text{exp}}} [\log D_\psi(s, a)] + \sum_{(s,a,s',d) \in B} [\log(1 - D_\psi(s, a))] \right\}$ ;
            Compute  $C$  targets  $\forall (s, a, s', d) \in B$ 
             $y(s, a, d) = \gamma (r_\psi(s', \tilde{a}') + \min_{i=1,2} C_{\phi_{\text{targ}i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s'))$ ,  $\tilde{a}' \sim \pi_\theta(\cdot|s')$ ,  $r_\psi(s', \tilde{a}') = h(D_\psi(s', \tilde{a}'))$ 
            Update  $C$ -functions parameters ( $\phi_1, \phi_2$ ) with gradient descent.
             $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,s',d) \in B} (C_{\phi_i}(s, a) - y(s, a, d))^2$ , for  $i = 1, 2$ 
            Update policy parameters ( $\theta$ ) with gradient ascent.
             $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left( r_\psi(s, \tilde{a}) + \min_{i=1,2} C_{\phi_i}(s, \tilde{a}) - \alpha \log \pi_\theta(\tilde{a}|s) \right)$ ,  $\tilde{a} \sim \pi_\theta(\cdot|s)$ ,  $r_\psi(s, \tilde{a}) = h(D_\psi(s, \tilde{a}))$ 
            Update target networks.
             $\phi_{\text{targ}i} \leftarrow \zeta \phi_{\text{targ}i} + (1 - \zeta) \phi_i$ , for  $i = 1, 2$ ;  $\zeta$  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).

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

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

In Appendix A.2, we derive the conditions under which ARC is likely to outperform AC by performing a (Signal to Noise Ratio) SNR analysis similar to [34]. Intuitively, favourable conditions

for ARC are (i) Error in gradient due to function approximation being similar or smaller for C as compared to Q (ii) the gradient of the immediate reward not having a high negative correlation with the gradient of C ($\mathbb{E}[\nabla_a r(s, a) \nabla_a C(s, a)]$ is not highly negative). Under these conditions, ARC would produce a higher SNR estimate of the gradient to train the policy. We believe that AIL is likely to present favourable conditions for ARC since the reward is shaped.

ARC would under-perform AC if the error in gradient due to function approximation of C network is significantly higher than that of Q network. In the general RL setting, immediate reward might be misleading (i.e. $\mathbb{E}[\nabla_a r(s, a) \nabla_a C(s, a)]$ might be negative) which might hurt the performance of ARC. However, we propose using ARC for AIL where the adversary reward measures how closely the agent imitates the expert. In AIL, the adversary reward is dense/shaped making ARC likely to be useful in this scenario, as experimentally verified in the following section.

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. The results are discussed in more detail in Appendix F.

5.1 Policy Iteration on a Grid World

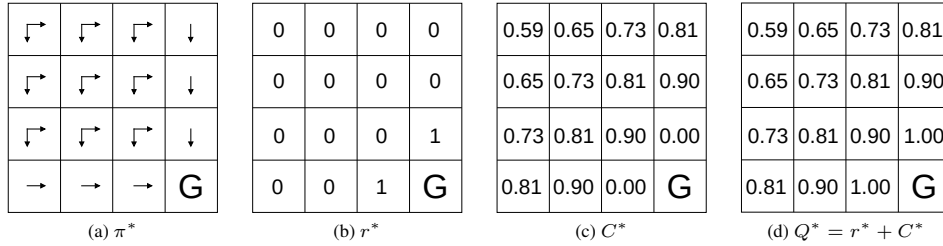


Figure 2: On a Grid World, the results of running two Policy Iteration (PI) algorithms - PI with C function (Algorithm 1) and the standard PI with Q function (Appendix C.1 Algorithm 3). Both algorithms converge in 7 policy improvement steps to the same optimal policy π^* as shown in a. The optimal policy gets the immediate reward shown in 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). Details are in E.1.

5.2 Imitation Learning in Mujoco continuous-control tasks

We used 4 Mujoco continuous-control environments from OpenAI Gym [35], as shown in Fig. 3. Expert trajectories were obtained by training a policy with SAC [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). For standard AIL algorithms (GAIL and f -MAX-RKL) and BC, we used the implementation of [28]. Further experimental details are presented in Appendix E.

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 [35] which have a simulated Fetch robot, [36]. In the FetchReach task, the robot needs to take it's end-effector to the goal (virtual red sphere) as quickly as possible. In the FetchPush task, the robot's needs to push the block to the goal as quickly as possible. We used hand-coded proportional controller to generate expert trajectories for these tasks. Further details are presented in Appendix E.3.

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.

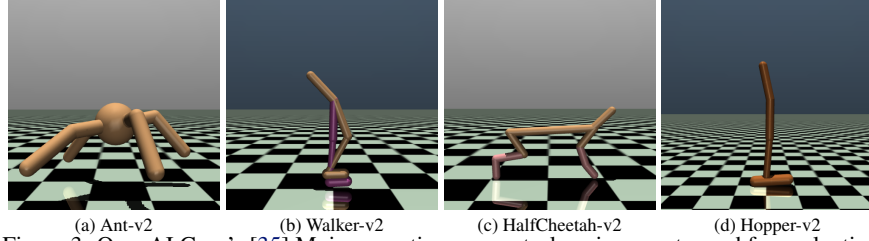


Figure 3: OpenAI Gym’s [35] Mujoco continuous-control environments used for evaluation.

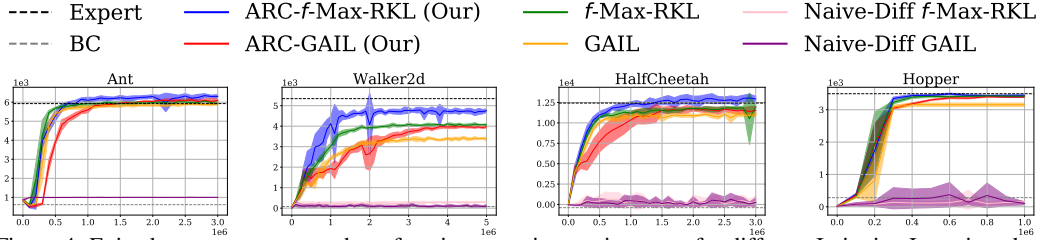


Figure 4: Episode return versus number of environment interaction steps for different Imitation Learning algorithms on Mujoco continuous-control environments.

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)	6306.25 \pm 95.91	4753.63 \pm 88.89	12930.51 \pm 340.02	3433.45 \pm 49.48
f -Max-RKL	5949.81 \pm 98.75	4069.14 \pm 52.14	11970.47 \pm 145.65	3417.29 \pm 19.8
Naive-Diff f -Max-RKL	998.27 \pm 3.63	294.36 \pm 31.38	357.05 \pm 732.39	154.57 \pm 34.7
ARC-GAIL (Our)	6090.19 \pm 99.72	3971.25 \pm 70.11	11527.76 \pm 537.13	3392.45 \pm 10.32
GAIL	5907.98 \pm 44.12	3373.26 \pm 98.18	11075.31 \pm 255.69	3153.84 \pm 53.61
Naive-Diff GAIL	998.17 \pm 2.22	99.26 \pm 76.11	277.12 \pm 523.77	105.3 \pm 48.01
BC	615.71 \pm 109.9	81.04 \pm 119.68	-392.78 \pm 74.12	282.44 \pm 110.7

Table 2: Policy return on Mujoco environments using different Imitation Learning algorithms. Each algorithm is run with 10 random seeds. Each seed is evaluated for 20 episodes.

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 2^{nd} 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. Since we didn’t tune hyper-parameters for these tasks (both our methods and the baselines, details in Appendix E.3), it is likely that the performances would improve with further parameter tuning. Without fine-tuning hyper-parameters for these tasks, ARC algorithms showed higher performance than the baselines. This shows that ARC algorithms are parameter robust and applicable to real robot tasks without much fine tuning.

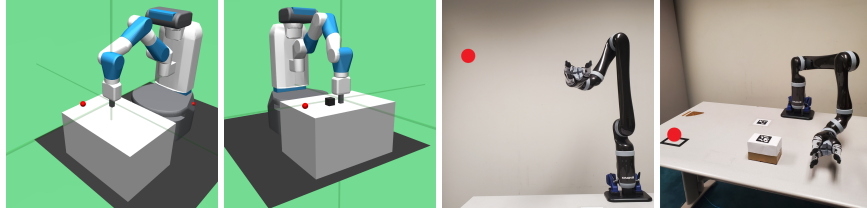


Figure 5: Simulated and real robotic manipulation tasks used for evaluation. Simplified 2D versions of the FetchReach **a** and FetchPush **b** tasks from OpenAI Gym, [35] with a Fetch robot, [36]. Corresponding JacoReach **c** and JacoPush **d** tasks with a real Kinova Jaco Gen 2 arm, [37].

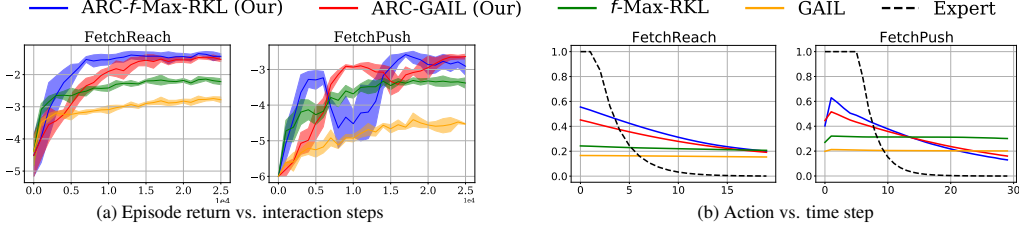


Figure 6: **a** Episode return vs. number of environment interaction steps for different Adversarial Imitation Learning algorithms on FetchPush and FetchReach tasks. **b** Magnitude of the 2^{nd} action dimension versus time step in a single episode for different algorithms.

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- <i>f</i> -Max-RKL (Our)	-1.43 \pm 0.08	-2.91 \pm 0.25	-0.38 \pm 0.02	-1.25 \pm 0.06
<i>f</i> -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’ (s, a) pairs. There are several scenarios where obtaining (s, a) pairs is challenging (e.g. kinesthetic teaching). In such scenarios, ARC is not directly applicable. People often use tricks to mitigate this issue and using (s, a) pairs to train a policy is a popular choice [38, 39, 40, 41, 42]. (2) ARC-AIL can only work with continuous action space. 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. Further, they are usually shaped. 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.

Acknowledgments

We are thankful to Swaminathan Gurumurthy and Tejus Gupta for several insightful discussions on the idea. We are also thankful to Rohit Jena, Advait Gadhihar for their feedback on the manuscript and Dana Hughes, Sushmita Das for their support with some logistics of the project.

Finally, we are thankful to the reviewers for their constructive feedback through the rebuttal period, which we believe helped strengthen the paper.

This work has been supported by the following grants: Darpa HR001120C0036, AFRL/AFOSR FA9550-18-1-0251 and ARL W911NF-19-2-0146.

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] J. W. Roberts and R. Tedrake. Signal-to-noise ratio analysis of policy gradient algorithms.
- [35] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

- [36] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich. Fetch and freight: Standard platforms for service robot applications.
- [37] 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.
- [38] S. Young, D. Gandhi, S. Tulsiani, A. Gupta, P. Abbeel, and L. Pinto. Visual imitation made easy. In *Conference on Robot Learning (CoRL)*, 2020.
- [39] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine. Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784*, 2020.
- [40] Y. Lu, K. Hausman, Y. Chebotar, M. Yan, E. Jang, A. Herzog, T. Xiao, A. Irpan, M. Khansari, D. Kalashnikov, et al. Aw-opt: Learning robotic skills with imitation and reinforcement at scale. In *CoRL*, 2021.
- [41] O. Scheel, L. Bergamini, M. Wolczyk, B. Osinowski, and P. Ondruska. Urban driver: Learning to drive from real-world demonstrations using policy gradients. In *5th Annual Conference on Robot Learning*, 2021.
- [42] R. Hoque, A. Balakrishna, E. Novoseller, A. Wilcox, D. S. Brown, and K. Goldberg. Thriftydagger: Budget-aware novelty and risk gating for interactive imitation learning. In *Conference on Robot Learning*, pages 598–608. PMLR, 2022.

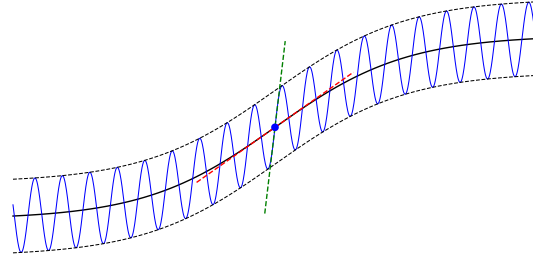
A Accuracy of gradient

A.1 Error in gradient of an approximate function

Theorem 2. *The error in gradient of an approximation of a differentiable function can be arbitrarily large even if the function approximation is accurate (but not exact). Formally, for any differentiable function $f(x) : A \rightarrow B$, any small value of $\epsilon > 0$ and any large value of $D > 0$, we can have an approximation $\hat{f}(x)$ s.t. the following conditions are satisfied:*

$$\left| \hat{f}(x) - f(x) \right| \leq \epsilon \quad \forall x \in A \quad (\text{Accurate approximation}) \quad (14)$$

$$\left| \nabla_x \hat{f}(x) - \nabla_x f(x) \right| \geq D \quad \text{for some } x \in A \quad (\text{Arbitrarily large error in gradient}) \quad (15)$$



— $f(x)$ - - - $f(x) \pm \epsilon$ — $\hat{f}(x)$ - - - $\nabla f(x_0)$ - - - $\nabla \hat{f}(x_0)$ • x_0

Figure 7: Illustrative example of a function approximation with accurate approximation, E.q.14 but large error in gradient, E.q.15.

Proof. For any differentiable $f(x)$, $\epsilon > 0$ and $D > 0$, we can construct many examples of $\hat{f}(x)$ that satisfy the conditions in Eq. 14 and 15. Here we show just one example that satisfies the 2 conditions. Let x_0 be any point $x_0 \in A$. We can choose $\hat{f}(x) = f(x) + \epsilon \sin(b(x - x_0))$, where $b = \frac{2D}{\epsilon}$. This is shown pictorially in Fig. 7.

The error in function approximation is:

$$\begin{aligned} \left| \hat{f}(x) - f(x) \right| &= \left| \epsilon \sin(b(x - x_0)) \right| = \epsilon |\sin(b(x - x_0))| \\ &\leq \epsilon \quad \because \sin(x) \in [-1, 1], \forall x \in \mathbb{R} \end{aligned}$$

Thus, $\hat{f}(x)$ satisfies Eq. 14 and approximates $f(x)$ accurately.

The error in gradient at x_0 is:

$$\begin{aligned} \left| \nabla_x \hat{f}(x) - \nabla_x f(x) \right| \Big|_{x=x_0} &= \left| \nabla_x f(x) + \epsilon b \cos(b(x - x_0)) - \nabla_x f(x) \right| \Big|_{x=x_0} \\ &= \epsilon b |\cos(b(x_0 - x_0))| \\ &= \epsilon \frac{2D}{\epsilon} |\cos(0)| \quad \because b = \frac{2D}{\epsilon} \text{ and } \cos(0) = 1 \\ &= 2D > D \end{aligned}$$

Thus, $\hat{f}(x)$ satisfies Eq.15, i.e. the error in gradient can be arbitrarily large even if function approximation is accurate. We can also see visually from Fig. 7 that although $\left| \hat{f}(x) - f(x) \right| < \epsilon$, there is a large difference between $\nabla \hat{f}(x_0)$ and $\nabla f(x_0)$. \square

A.2 Decomposition in ARC leads to more accurate gradient for AIL

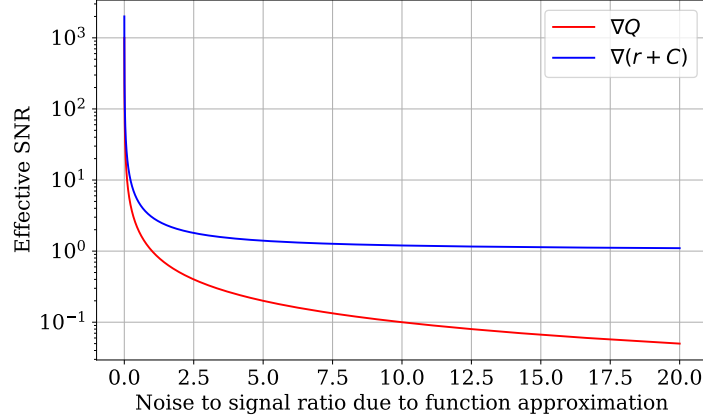


Figure 8: Signal to Noise Ratio (SNR) in Q gradient approximation as noise to signal ratio due to function approximation increases. Higher SNR is better. Using our proposed decomposition, $Q = r + C$, the effective SNR is higher than that without decomposition, when there is large noise due to function approximation.

From Theorem 2, there is no bound on the error in gradient of an approximate function. Let \hat{Q} and \hat{C} denote the approximated Q and C values respectively. In the worst case, the gradients $\nabla_a \hat{Q}(s, a)$ and $\nabla_a \hat{C}(s, a)$ can both be completely wrong and act like random noise. Even in that case, the gradient obtained using our proposed decomposition ($Q = r + C$) would be useful because $\nabla_a r(s, a)$ is exact and hence $\nabla_a((s, a) + \hat{C}(s, a))$ would have useful information.

It is possible that the immediate “environment reward” is misleading which might hurt ARC. However, the “adversary reward” is a measure of closeness between agent and expert actions. It naturally is never misleading as long as we have a reasonably trained adversary. If we have an initial bad action that the expert takes to obtain a high reward later on, then the initial bad action will have a corresponding high adversary reward.

In practice, we can expect both $\nabla_a \hat{Q}(s, a)$ and $\nabla_a \hat{C}(s, a)$ to have some finite noise. Signal to Noise (SNR) of a noisy signal is defined as the ratio of the magnitudes of true (signal strength) and noisy components in the noisy signal. If a signal $\hat{f} = f + \epsilon$ has a true signal component f and a noisy component ϵ , then the SNR is $\frac{\mathbb{E}f^2}{\mathbb{E}\epsilon^2}$. Higher SNR is better. SNR has been used in the past to analyze policy gradient algorithms [34].

Let us consider the case of a 1D environment which makes $\nabla_a r(s, a)$, $\nabla_a \hat{C}(s, a)$ and $\nabla_a \hat{Q}(s, a)$ scalars.

1. “Signal strength of $\nabla_a r(s, a)$ ” = $\mathbb{E}[(\nabla_a r(s, a))^2] = S_r$ (say). Noise strength = 0
2. $\nabla_a \hat{C}(s, a) = \nabla_a C(s, a) + \epsilon_c$ (i.e. True signal + noise)
3. “Signal strength of $\nabla_a \hat{C}(s, a)$ ” = $\mathbb{E}[(\nabla_a C(s, a))^2] = S_c$ (say)
4. Let SNR of $\nabla_a \hat{C}(s, a) = \text{snr}_c$ (say).
5. Noise strength = $\mathbb{E}[\epsilon_c^2] = S_n$ (say)
6. By definition of SNR, $\text{snr}_c = \frac{S_c}{S_n} \implies S_n = \frac{S_c}{\text{snr}_c}$
- 7.

$$\text{Final signal} = \nabla_a r(s, a) + \nabla_a \hat{C}(s, a) \quad (16)$$

$$= \nabla_a r(s, a) + \nabla_a C(s, a) + \epsilon_c \quad (17)$$

$$= (\nabla_a r(s, a) + \nabla_a C(s, a)) + \epsilon_c \quad (18)$$

$$= \text{net true signal} + \text{net noise} \quad (19)$$

8.

$$\text{Net signal strength} = \mathbb{E}[(\nabla_a r(s, a) + \nabla_a C(s, a))^2] \quad (20)$$

$$= \mathbb{E}[(\nabla_a r(s, a))^2] + \mathbb{E}[(\nabla_a C(s, a))^2] + 2\mathbb{E}[\nabla_a r(s, a)\nabla_a C(s, a)] \quad (21)$$

$$= S_r + S_c + 2S_{r,C} \quad (22)$$

9.

$$\text{Net SNR} = \frac{\text{Net signal strength}}{\text{Net noise strength}} \quad (23)$$

$$= \frac{S_r + S_c + 2S_{r,C}}{S_n} \quad (24)$$

$$= \frac{S_r + S_c + 2S_{r,C}}{\frac{S_c}{\text{snr}_c}} \quad (25)$$

$$= \text{snr}_c \left(\frac{S_r}{S_c} + 1 + \frac{2S_{r,c}}{S_c} \right) \quad (26)$$

$$(27)$$

Let the SNR in $\nabla_a \hat{Q}(s, a)$ be snr_Q . Now, let's find when the net SNR in $\nabla_a \hat{C}(s, a)$ is higher than snr_Q , i.e. when does the decomposition lead to higher SNR.

$$\text{Net SNR in } \nabla_a \hat{C}(s, a) \geq \text{snr}_Q \quad (28)$$

$$\implies \text{snr}_c \left(\frac{S_r}{S_c} + 1 + \frac{2S_{r,c}}{S_c} \right) \geq \text{snr}_Q \quad (29)$$

$$\implies \text{snr}_c \geq \frac{1}{\left(\frac{S_r}{S_c} + 1 + \frac{2S_{r,c}}{S_c} \right)} \text{snr}_Q \quad (30)$$

Thus, Net SNR in $\nabla_a \hat{C}(s, a)$ is higher than snr_Q if Eq. 30 holds true.

Consider 3 cases:

- Case 1: $S_{r,c} \geq 0$

What does this mean?:

$S_{r,c} = \mathbb{E}[\nabla_a r(s, a)\nabla_a C(s, a)]$, this means $\nabla_a r(s, a)$ and $\nabla_a C(s, a)$ are positively correlated.

Implication:

$\frac{S_r}{S_c} \geq 0$ since it is a ratio of signal strengths. Thus $\frac{S_r}{S_c} + 1 + \frac{2S_{r,c}}{S_c} \geq 1$ since we are adding non-negative terms to 1. Thus, $\frac{1}{\frac{S_r}{S_c} + 1 + \frac{2S_{r,c}}{S_c}} \leq 1$. Let's call $\frac{1}{\frac{S_r}{S_c} + 1 + \frac{2S_{r,c}}{S_c}} = \text{fraction}$.

Thus, Eq. 30 reduces to $\text{snr}_c \geq \text{fraction} \times \text{snr}_Q$.

In other words, even if snr_C is a certain fraction of snr_Q , the net SNR due to decomposition is higher than that without decomposition.

- Case 2: $-\frac{S_r}{2} \leq S_{r,c} < 0$

What does this mean?:

This means $\nabla_a r(s, a)$ and $\nabla_a C(s, a)$ are slightly negatively correlated.

Implication: In this case,

$$\frac{S_r}{S_c} + 1 + \frac{2S_{r,c}}{S_c} \geq \frac{S_r}{S_c} + 1 + \left(\frac{2}{S_c}\right)\left(\frac{-S_r}{2}\right) \quad (31)$$

$$\implies \frac{S_r}{S_c} + 1 + \frac{2S_{r,c}}{S_c} \geq \frac{S_r}{S_c} + 1 - \frac{S_r}{S_c} \quad (32)$$

$$\implies \frac{S_r}{S_c} + 1 + \frac{2S_{r,c}}{S_c} \geq 1 \quad (33)$$

Just like in case 1, we get the denominator in in Eq. 30 is a fraction. This in turn leads to the same conclusion that even if snr_C is a certain fraction of snr_Q , the net SNR due to decomposition is higher than that without decomposition.

- Case 3: $S_{r,c} < -\frac{S_r}{2}$

What does this mean?:

This means $\nabla_a r(s, a)$ and $\nabla_a C(s, a)$ are highly negatively correlated.

Implication: In this case, we get the denominator in Eq. 30 is > 1 . Decomposition would only help if $\text{snr}_c > \text{snr}_Q$ by the same factor.

What determines relative values of snr_c and snr_Q in AIL?

snr_c and snr_Q arise from noise in gradient due to function approximation. In other words, if Q and C both are similarly difficult to approximate, then we can expect snr_c and snr_Q to have similar values. In AIL, the adversary reward is dense/shaped which is why snr_c is likely to be greater or at least similar to snr_Q .

When is the decomposition likely to help in AIL?

As long as snr_c is similar to higher than snr_Q and the gradients of the reward and C are not highly negatively correlated (in expectation), the decomposition is likely to help.

In Fig. 8 show how this looks visually for the special case where $\text{snr}_c = \text{snr}_c$ and that signal strength of $\nabla_a r(s, a)$ is equal to the signal strength of $\nabla_a C(s, a)$.

When would the decomposition hurt? Two factors that can hurt ARC are:

1. If snr_c is significantly lower than snr_Q
2. If $S_{r,c}$ is highly negative

Appendix D.2 experimentally verifies that the decomposition in ARC produces more accurate gradient than AC using a simple 1D driving environment.

B Properties of C function

We show some useful properties of the C function. We define the optimal C function, C^* as $C^*(s, a) = \max_{\pi} C^{\pi}(s, a)$. There exists a unique optimal C function for any MDP as described in Appendix B.1 Lemma 1. We can derive the Bellman equation for C^{π} (Appendix B.2 Lemma 2), similar to the Bellman equations for traditional action value function Q^{π} [16]. Using the recursive Bellman equation, we can define a Bellman backup operation for policy evaluation which converges to the true C^{π} function (Theorem 3). Using the convergence of policy evaluation, we can arrive at the Policy Iteration algorithm using C function 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 function or V function [16]. For comparison, the standard Policy Iteration with Q function algorithm is described in Appendix C.1 Algorithm 3.

B.1 Unique optimality of C function

Lemma 1. *There exists a unique optimum C^* for any MDP.*

Proof. The unique optimality of C function can be derived from the optimality of the Q function [16]. The optimum Q function, Q^* is defined as:

$$\begin{aligned} Q^*(s, a) &= \max_{\pi} Q^{\pi}(s, a) \\ &= \max_{\pi} [r(s, a) + C^{\pi}(s, a)] \\ &= r(s, a) + \max_{\pi} C^{\pi}(s, a) \\ &= r(s, a) + C^*(s, a) \end{aligned} \tag{34}$$

$$\therefore C^*(s, a) = Q^*(s, a) - r(s, a) \tag{35}$$

Since Q^* is unique [16], (35) implies C^* must be unique. \square

B.2 Bellman backup for C function

Lemma 2. *The recursive Bellman equation for C^π is as follows*

$$C^\pi(s, a) = \gamma \sum_{s'} P(s'|s, a) \sum_{a'} \pi(a'|s') (r(s', a') + C^\pi(s', a'))$$

Proof. The derivation is similar to that of state value function V^π presented in [16]. We start deriving the Bellman backup equation for C^π function by expressing current $C(s_t, a_t)$ in terms of future $C(s_{t+1}, a_{t+1})$. In the following, the expectation is over the policy π and the transition dynamics \mathcal{P} and is omitted for ease of notation.

$$C^\pi(s_t, a_t) = \mathbb{E} \sum_{k \geq 1} \gamma^k r_{t+k} \quad (36)$$

$$= \mathbb{E} \left(\gamma r_{t+1} + \sum_{k \geq 2} \gamma^k r_{t+k} \right) \quad (37)$$

$$= \gamma \left(\mathbb{E}[r_{t+1}] + \mathbb{E} \sum_{k \geq 1} \gamma^k r_{t+1+k} \right) \quad (38)$$

$$= \gamma \left(\mathbb{E}[r_{t+1}] + \mathbb{E} \sum_{k \geq 1} \gamma^k r_{t+1+k} \right) \quad (39)$$

$$= \gamma \mathbb{E}(r_{t+1} + C(s_{t+1}, a_{t+1})) \quad (40)$$

$$(41)$$

Using Eq. 40, we can write the recursive Bellman equation of C .

$$C(s, a) = \gamma \sum_{s'} P(s'|s, a) \sum_{a'} \pi(a'|s') (r(s', a') + C(s', a')) \quad (42)$$

□

B.3 Convergence of policy evaluation using C function

Theorem 3. *The following Bellman backup operation for policy evaluation using C function converges to the true C function, C^π*

$$C^{n+1}(s, a) \leftarrow \gamma \sum_{s'} P(s'|s, a) \sum_{a'} \pi(a'|s') (r(s', a') + C^n(s', a'))$$

Here, C^n is the estimated value of C at iteration n.

Proof. Let us define $F_{(\cdot)}$ as the Bellman backup operation over the current estimates of C values:

$$F_C(s, a) = \gamma \sum_{s'} P(s'|s, a) \sum_{a'} \pi(a'|s') (r(s', a') + C(s', a')) \quad (43)$$

We prove that F is a contraction mapping w.r.t ∞ norm and hence is a fixed point iteration. Let, C_1 and C_2 be any 2 sets of estimated C values.

$$\|F_{C_1} - F_{C_2}\|_\infty = \max_{s,a} |F_{C_1} - F_{C_2}| \quad (44)$$

$$= \gamma \max_{s,a} \left| \sum_{s'} P(s'|s, a) \sum_{a'} \pi(a'|s') [r(s', a') + C_1(s', a')] - \sum_{s'} P(s'|s, a) \sum_{a'} \pi(a'|s') [r(s', a') + C_2(s', a')] \right| \quad (45)$$

$$= \gamma \left| \max_{s,a} \sum_{s'} P(s'|s, a) \sum_{a'} \pi(a'|s') (C_1(s', a') - C_2(s', a')) \right| \quad (46)$$

$$\leq \gamma \max_{s,a} \sum_{s'} P(s'|s, a) \sum_{a'} \pi(a'|s') |C_1(s', a') - C_2(s', a')| \quad (47)$$

$$\leq \gamma \max_{s,a} \sum_{s'} P(s'|s, a) \max_{a'} |C_1(s', a') - C_2(s', a')| \quad (48)$$

$$\leq \gamma \max_{s'} \max_{a'} |C_1(s', a') - C_2(s', a')| \quad (49)$$

$$= \gamma \max_{s,a} |C_1(s, a) - C_2(s, a)| \quad (50)$$

$$= \gamma \|C_1 - C_2\|_\infty \quad (51)$$

$$\therefore \|F_{C_1} - F_{C_2}\|_\infty \leq \gamma \|C_1 - C_2\|_\infty \quad (52)$$

Eq. 52 implies that iterative operation of $F_{(\cdot)}$ converges to a fixed point. The true C^π function satisfies the Bellman equation Eq. 42. These two properties imply the policy evaluation converges to the true C^π function. \square

B.4 Convergence of policy iteration using C function

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

Proof. From Theorem 3, the policy evaluation step converges to true C^π function. The policy improvement step is exactly the same as in the case with Q function since $Q^\pi(s, a) = r(s, a) + C^\pi(s, a)$, which is known to converge to an optimum policy [16]. These directly imply that Policy Iteration with C function converges to the optimal C^* function and an optimal policy π^* . \square

C Popular Algorithms

C.1 Policy Iteration using Q function

We restate the popular Policy Iteration using Q function algorithm in Algorithm 3.

Algorithm 3: Policy Iteration using Q function

```

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

```

D Additional Results

D.1 Visualization of Real Robot Policy Execution

Fig. 9 shows example snapshots of the final block position in the JacoPush task using different AIL algorithms. ARC aided AIL algorithms were able to push the block closer the goal thereby achieving a lower final block to goal distance, as compared to the standard AIL algorithms.

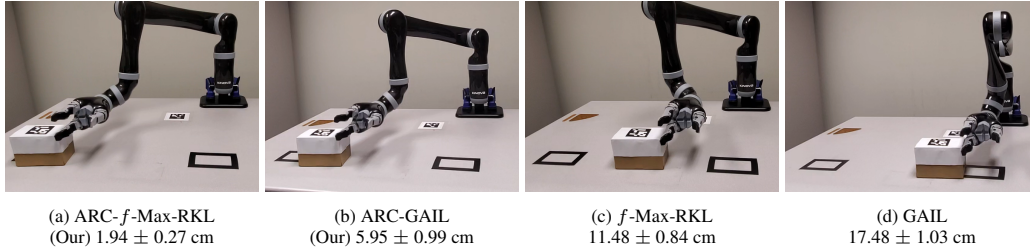


Figure 9: Example snapshots of final position of block in the JacoPush task using different Adversarial Imitation Learning algorithms and the average final block to goal distance (lower is better) in each case.

D.2 Accuracy of gradient of the proposed approach

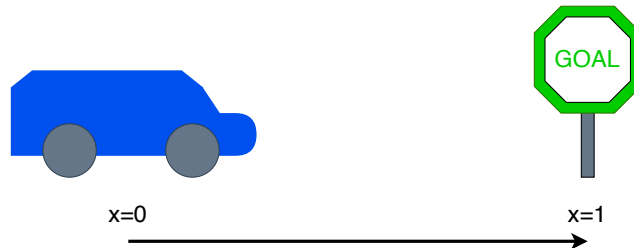


Figure 10: A 1D driving environment where a policy needs to imitate an expert driver which initially drives fast but slows down as the car approaches the goal.

We use a simple toy environment to empirically show that our proposed approach of fitting C function results in a better estimate of the policy gradient than the standard approach of fitting Q function. Fig. 10 shows the environment. An agent needs to imitate an expert policy that drives the car from

the start location $x = 0$ to the goal $x = 1$. The expert policy initially drives the car fast but slows down as the car approaches the goal.

The expert policy, agent policy and reward functions are described by the following python functions:

```

1 def expert_policy(obs):
2     gain = 0.1
3     goal = 1
4     return gain*(goal-obs)+0.1
5
6 def agent_policy(obs):
7     return 0.15
8
9 def reward_fn(obs, a):
10    expert_a = expert_policy(obs)
11    return -100*(a-expert_a)**2

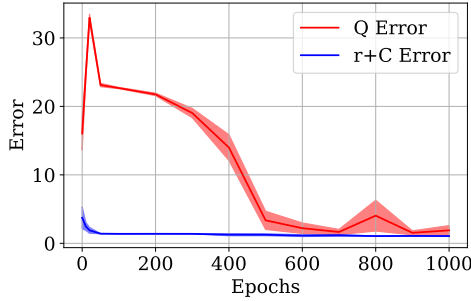
```

Listing 1: Python code defining the expert policy

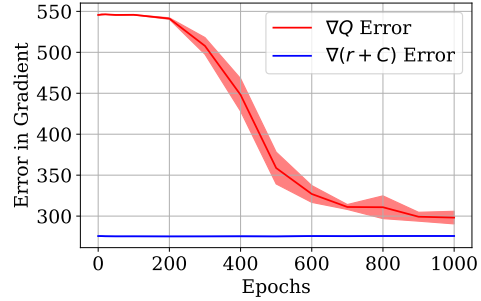
We uniformly sampled states and actions in this environment. We then fit a neural network to the Q function and to the C function by running updates for different numbers of epochs and repeating the experiment 5 times.

After that, we compare the learnt Q function and (r + learnt C) function to the true Q function. (The true Q function is obtained by rolling out trajectories in the environment).

The following 2 figures show the results. On the left, we show the error in estimating the true Q function and on the right we show the error in estimating the true gradient of Q. (The True gradient of Q is calculated by a finite difference method).



(a) Error in estimating Q



(b) Error in estimating $\nabla_a Q(s, a)$

Clearly, the decomposition leads to lower error and variance in estimating both the true Q function and its gradient. Even with slight error with r+C initially, the corresponding error in gradient is much lower for r+C than for Q. Moreover, towards the tail of the plots (after 600 epochs), both Q and r+C estimate the true Q function quite accurately but the error in the gradient of r+C is lower than that for directly estimating the Q function.

We visualize the estimated values of Q in Fig. 12 and the estimated gradients of Q in Fig. 13 by the two methods after 500 epochs of training. Using r+C estimates the gradients much better than Q.

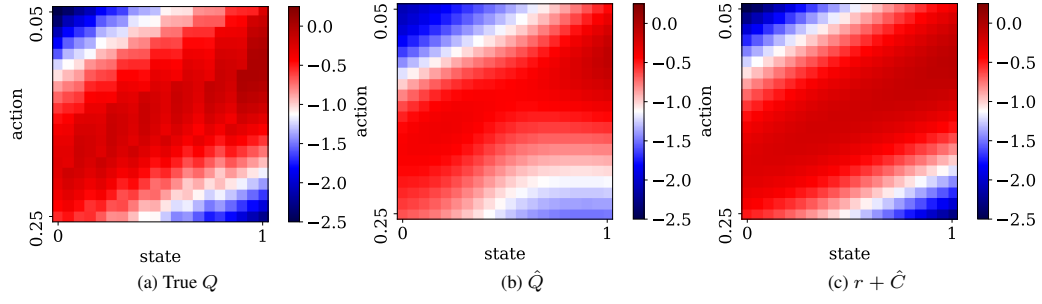


Figure 12: True value of Q , Fig. 13a along with estimated values of Q by directly fitting a Q network, Fig. 13b and by fitting a C network, Fig. 13c.

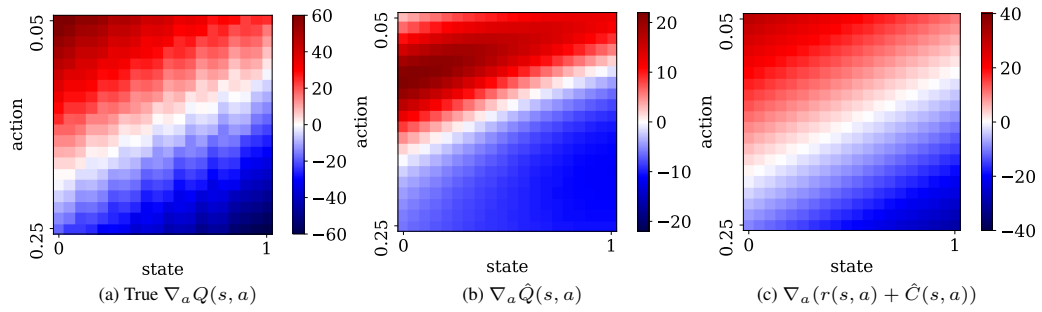


Figure 13: True value of $\nabla_a Q(s, a)$, Fig. 13a along with estimated values of $\nabla_a Q(s, a)$ function by directly fitting a Q network, Fig. 13b and by fitting a C network, Fig. 13c.

E Experimental Details

E.1 Policy iteration on a Grid World

Our objective is to experimentally validate if Policy Iteration (PI) with C function converges to an optimal policy (Theorem 1). We choose a simple Grid World environment as shown in Fig. 2 to illustrate this. At every time step, the agent can move in one of 4 directions - left, right, up or down. The reward is 1 for reaching the goal (G) and 0 otherwise. The discount factor $\gamma = 0.9$.

On this environment, we run two PI algorithms - PI with C function (Algorithm 1) and the standard PI with Q function (Appendix C.1 Algorithm 3). Fig. 2 shows the results of this experiment. Both the algorithms converge to the same optimal policy π^* shown in Fig. 2a. This optimal policy receives the immediate reward shown in Fig. 2b. Note that the immediate reward is 1 for states adjacent to the goal G as the agent receives 1 reward for taking an action that takes it to the goal. Fig. 2c and Fig. 2d show the values of C^* , Q^* that PI with C function and PI with Q function respectively converge to. In Fig. 2d, $Q^* = r^* + C^*$, which is consistent with the relation between Q function and C function (11). In Fig. 2d, the Q^* values in the states adjacent to the goal are 1 since Q function includes the immediate reward (9). C function doesn't include the immediate reward (10) and hence the C^* values in these states are 0 (Fig. 2c). This experiment validates that PI with C function converges to an optimal policy as already proved in Theorem 1.

E.2 Imitation Learning in Mujoco continuous-control tasks

Environment We use Ant-v2, Walker-v2, HalfCheetah-v2 and Hopper-v2 Mujoco continuous-control environments from OpenAI Gym [35]. All 4 environments use Mujoco, a realistic physics-engine, to model the environment dynamics. The maximum time steps, T is set to 1000 in each environment.

Code We implemented our algorithm on top of the AIL code of [28]. The pre-implemented standard AIL algorithms (f -MAX-RKL, GAIL) used SAC [3] as the RL algorithm and the ARC aided AIL algorithms are SARC-AIL (Algorithm 2) algorithms.

Expert trajectories We used the expert trajectories provided by [28]. They used SAC [3] to train an expert in each environments. The policy network, π_θ was a tanh squashed Gaussian which parameterized the mean and standard deviation with two output heads. Each of the policy network, π_θ and the 2 critic networks, Q_{ϕ_1}, Q_{ϕ_2} was a (64, 64) ReLU MLP. Each of them was optimized by Adam optimizer with a learning rate of 0.003. The entropy regularization coefficient, α was set to 1, the batch size was set to 256, the discount factor γ was set to 0.99 and the polyak averaging coefficient ζ for target networks was set to 0.995. The expert was trained for 1 million time steps on Hopper and 3 million time steps on the other environments. For each environment, we used 1 trajectory from the expert stochastic policies to train the imitation learning algorithms.

Standard AIL For the standard AIL algorithms (f -MAX-RKL [12] and GAIL [10]) we used the code provide by [28]. The standard AIL algorithms used SAC [3] as the RL algorithm. SAC used the same network and hyper-parameters that were used for training the expert policy except the learning rate and the entropy regularization coefficient, α . The learning rate was set to 0.001. α was set to 0.05 for HalfCheetah and to 0.2 for the other environments. The reward scale and gradient penalty coefficient were set to 0.2 and 4.0 respectively. In each environment, the observations were normalized in each dimension of the state using the mean and standard deviation of the expert trajectory.

Baseline GAIL in Hopper was slightly unstable and we had to tune GAIL separately for the Hopper environment. We adjusted the policy optimizer's learning rate schedule to decay by a factor of 0.98 at every SAC update step after 5 epochs of GAIL training.

For the discriminator, we used the same network architecture and hyper-parameters suggested by [28]. The discriminator was a (128,128) tanh MLP network with the output clipped within [-10,10]. The discriminator was optimized with Adam optimizer with a learning rate of 0.0003 and a batch size of 128. Once every 1000 environment steps, the discriminator and the policy were alternately trained for 100 iterations each.

Each AIL algorithm was trained for 1 million environment steps on Hopper, 3 million environment steps on Ant, HalfCheetah and 5 million environment steps on Walker2d.

ARC aided AIL For ARC aided AIL algorithms, we modified the SAC implementation of [28] to SARC - Soft Actor Residual Critic. This was relatively straight forward, we used the same networks to parameterize C_{ϕ_1}, C_{ϕ_2} instead of Q_{ϕ_1}, Q_{ϕ_2} based on the steps of SARC-AIL (Algorithm 2). For SARC, we used the same network and hyper-parameters as SAC except the following changes. Learning rate was set to 0.0001. Entropy regularization coefficient, α was set to 0.05 for HalfCheetah and 1 for the other environments. No reward scaling was used (reward scale was set to 1). The C networks were updated 10 times for every update of the policy network. We did so because we noticed that otherwise the C networks (C_{ϕ_1}, C_{ϕ_2}) were slower to update as compared to the policy network.

The discriminator was the same as with standard AIL algorithms except it had 2 Resnet blocks of 128 dimension each, with batch normalization and leaky ReLU activation. These changes were motivated by common tricks to train stable GANs [22]. In GANs, the generator is differentiated through the discriminator and the use of leaky ReLU and Resnet helps in gradient flow through the discriminator. In ARC aided AIL we have a similar scenario, the policy is differentiated through the reward function. We briefly tried to make the same changes with standard AIL algorithms as well but didn't see an improvement in performance.

Naive-Diff For the Naive-Diff aided AIL algorithms (Naive-Diff- f -MAX-RKL and Naive-Diff-GAIL), we used the same network architectures and hyper-parameters as with ARC aided AIL.

Behavior Cloning For Behavior Cloning, we trained the agent to regress on expert actions by minimizing the mean squared error for 10000 epochs using Adam optimizer with learning rate of 0.001 and batch size of 256.

Evaluation We evaluated all the imitation learning algorithms based on the true environment return achieved by the deterministic version of their policies. Each algorithm was run on 5 different seeds and each run was evaluated for 20 episodes. The final mean reward was used for comparing the algorithms. The results are presented in Table 2.

E.3 Imitation Learning in robotic manipulation tasks

Environment We simplified 2D versions of the FetchReach-v1 and FetchPush-v1 environments from OpenAI gym, [35]. In the FetchReach task, the observation is a 2D vector containing the 2D position of the block with respect to the end-effector and needs to take it's end-effector to the goal as quickly as possible. In the FetchPush task, the robot's ob can observe a block and the goal location and needs to push the block to the goal as quickly as possible. Actions are 2D vectors controlling the relative displacement of the end-effector from the current position by a maximum amount of $\pm \Delta_{\max} = 3.3\text{cm}$. In the FetchReach task, the goal is initially located at $(15\text{cm}, -15\text{cm}) + \epsilon$ w.r.t the end-effector. Where ϵ is sampled from a 2D diagonal Normal distribution with 0 mean and 0.01cm standard deviation in each direction. In the FetchPush task, initially, the block is located at $(0\text{cm}, -10\text{cm}) + \epsilon_{\text{block}}$ and the goal is located at $(0\text{cm}, -30\text{cm}) + \epsilon_{\text{goal}}$ w.r.t the end-effector. $\epsilon_{\text{block}}, \epsilon_{\text{goal}}$ are sampled from 2D diagonal Normal distributions with 0 mean and 0.01cm standard deviation in each direction. The reward at each time step is $-d$, where d is the distance between end-effector and goal (in case of FetchReach) or the distance between the block and the goal (in case of FetchPush). d is expressed in meters. FetchReach task has 20 time steps and FetchPush task has 30 time steps.

Expert trajectories We used hand-coded proportional controller to generate expert trajectories for these tasks. For each task, we used 64 expert trajectories.

Hyper-parameters For each AIL algorithm, once every 20 environment steps, the discriminator and the policy were alternately trained for 10 iterations each. Each AIL algorithm was trained for 25,000 environment steps. All the other hyper-parameters were the same as those used with the Ant, Walker and Hopper Mujoco continuous-control environments (Section E.2). We didn't perform any hyper-parameter tuning (for both our methods and the baselines) in these experiments and the results might improve with some hyper-parameter tuning.

Evaluation For the simulated tasks, each algorithms is run with 5 random seeds and each seed is evaluated for 20 episodes.

E.4 Sim-to-real transfer of robotic manipulation policies



Figure 14: Experimental setup for the real robot experiments with a Kinova Jaco Gen 2 arm, [37]. An overhead Intel RealSense camera tracks an Aruco marker on the table to calibrate its 3D position w.r.t the world. It also tracks an Aruco marker on the block to extract its position.

Environment We setup a Kinova Jaco gen 2 arm as shown in Fig. 14. Aruco marker are used to get the position of the block and forward kinematics of the robot is used to get the position of the robot end-effector.

The JacoReach and JacoPush tasks with the real robot have the same objective as the FetchReach and FetchPush tasks as described in the previous section. The observations and actions in the real robot were transformed (translated, rotated and scaled) to map to those in the simulated tasks. 20cm in the FetchReach task corresponds to 48cm in the JacoReach task. Thus, observations in the real robot were scaled by $(20/48)$ before using as input to the trained policies. Similarly 20cm in the FetchPush task corresponds to 48cm in the JacoPush task and thus observations were scaled by $(20/48)$. The policy commands sent to the robot were in the form of cartesian displacements which were finally executed by Moveit path planner. Due to inaccuracies in the real world, small actions couldn't be executed and this hurt the performance the algorithms (particularly the baseline algorithms which produced very small actions). To address this, the actions were scaled up by a factor of 7. Correspondingly, the timesteps were scaled down by a factor to 7 to adjust for action scaling. Thus the JacoReach task had $(20/7 \sim 3)$ timesteps and the JacoPush task had $(30/7 \sim 5)$ timesteps.

Due to the different scale in distance and length of episodes (timesteps), the rewards in the simulator and the real robot are in different scales.

Evaluation For the real robot tasks, the best seed from each algorithm is chosen and is evaluated over 5 episodes.

E.5 Comparison to fully differentiable model based policy learning

If we have access to a differentiable model, we can directly obtain the gradient expected return (policy objective) w.r.t. the policy parameters θ :

$$\mathbb{E}_{\mathcal{T} \sim \mathcal{D}} [\nabla_{\theta} r(s_1, a_1) + \gamma \nabla_{\theta} r(s_2, a_2) + \gamma^2 \nabla_{\theta} r(s_3, a_3) + \dots] \quad (53)$$

Since we can directly obtain the objective's gradient, we do not necessarily need to use either a critic (Q) as in standard Actor Critic (AC) algorithms or a residual critic (C) as in our proposed Actor Residual Critic (ARC) algorithms.

In ARC, we do not assume access to a differentiable dynamics model.

F Discussion on Results

F.1 Imitation Learning in Mujoco continuous-control tasks

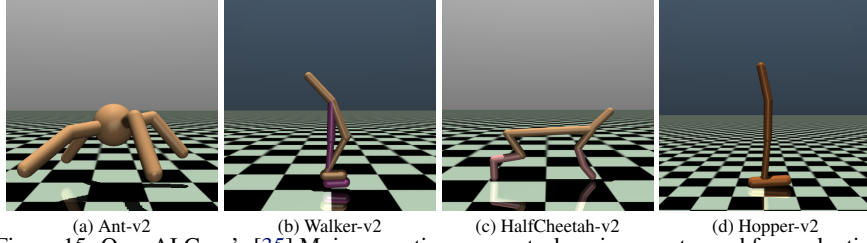


Figure 15: OpenAI Gym’s [35] Mujoco continuous-control environments used for evaluation.

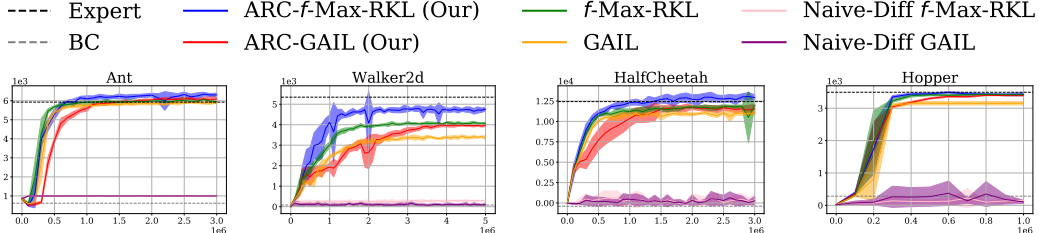


Figure 16: Episode return versus number of environment interaction steps for different Imitation Learning algorithms on Mujoco continuous-control environments.

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- <i>f</i> -Max-RKL (Our)	6306.25 \pm 95.91	4753.63 \pm 88.89	12930.51 \pm 340.02	3433.45 \pm 49.48
<i>f</i> -Max-RKL	5949.81 \pm 98.75	4069.14 \pm 52.14	11970.47 \pm 145.65	3417.29 \pm 19.8
Naive-Diff <i>f</i> -Max-RKL	998.27 \pm 3.63	294.36 \pm 31.38	357.05 \pm 732.39	154.57 \pm 34.7
ARC-GAIL (Our)	6090.19 \pm 99.72	3971.25 \pm 70.11	11527.76 \pm 537.13	3392.45 \pm 10.32
GAIL	5907.98 \pm 44.12	3373.26 \pm 98.18	11075.31 \pm 255.69	3153.84 \pm 53.61
Naive-Diff GAIL	998.17 \pm 2.22	99.26 \pm 76.11	277.12 \pm 523.77	105.3 \pm 48.01
BC	615.71 \pm 109.9	81.04 \pm 119.68	-392.78 \pm 74.12	282.44 \pm 110.7

Table 4: Policy return on Mujoco environments using different Adversarial Imitation Learning algorithms. Each algorithm is run with 10 random seeds and each seed is evaluated for 20 episodes.

Fig. 16 shows the training plots and Table 4 shows the final performance of the various algorithms. Across all environments and across both the AIL algorithms, incorporating ARC shows consistent improvement over standard AIL algorithms. That is, ARC-*f*-Max-RKL outperformed *f*-Max-RKL and ARC-GAIL outperformed GAIL. Across all algorithms, ARC-*f*-Max-RKL showed the highest performance. 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.

Walker2d ARC algorithms show the highest performance gain in the Walker2d environment. ARC-*f*-Max-RKL shows the highest performance followed by *f*-Max-RKL, ARC-GAIL and GAIL respectively. Naive-Diff and BC algorithms perform poorly and the order is Naive-Diff *f*-Max-RKL, Naive-Diff GAIL and BC.

Ant, HalfCheetah and Hopper ARC algorithms show consistent improvement over the standard AIL algorithms. However, there is only a modest improvement. This can be attributed to the fact that the baseline standard AIL algorithms already perform very well (almost matching expert performance). This leaves limited scope of improvement for ARC.

Ranking the algorithms Table 5 ranks the various algorithms in each of these environments. Amongst the 4 AIL algorithms, ARC-*f*-Max-RKL consistently ranked 1 and GAIL consistently ranked 4. The relative performance of *f*-Max-RKL and ARC-GAIL varied across the environments, i.e. sometimes the former performed better and at other times the later performed better. The relative

performance of Naive-Diff f -Max-RKL, Naive-Diff GAIL and BC also varied across the environments and they got ranks in the range 5 to 7.

Method	Ant	Walker2d	HalfCheetah	Hopper
ARC- f -Max-RKL (Our)	1	1	1	1
f -Max-RKL	3	2	2	2
Naive-Diff f -Max-RKL	5	5	5	6
ARC-GAIL (Our)	2	3	3	3
GAIL	4	4	4	4
Naive-Diff GAIL	6	6	6	7
BC	7	7	7	5

Table 5: Ranking different Imitation Learning algorithms based on policy return in Mujoco environments. Each algorithms is run with 5 random seeds and each seed is evaluated for 20 episodes.

F.2 Imitation Learning in robotic manipulation tasks

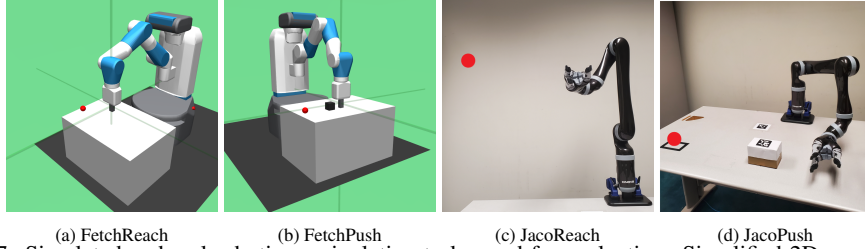


Figure 17: Simulated and real robotic manipulation tasks used for evaluation. Simplified 2D versions of the FetchReach **a** and FetchPush **b** tasks from OpenAI Gym, [35] with a Fetch robot, [36]. Corresponding JacoReach **c** and JacoPush **d** tasks with a real Kinova Jaco Gen 2 arm, [37].

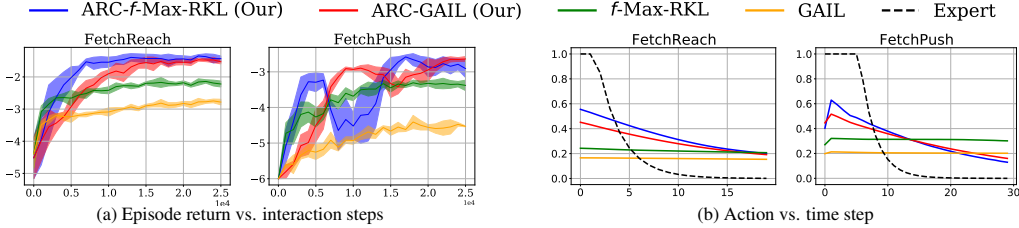


Figure 18: **a** Episode return vs. number of environment interaction steps for different Adversarial Imitation Learning algorithms on FetchPush and FetchReach tasks. **b** Magnitude of the 2^{nd} action dimension versus time step in a single episode for different algorithms.

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

Table 6: 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.

Fig. 18a shows the training plots and Table 6 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. Amongst all the evaluated algorithms, ARC- f -Max-RKL performed the best in the FetchReach task and ARC-GAIL performed the best in the FetchPush task.

Parameter robustness In the robotic manipulation tasks, we didn’t extensively tune the hyper-parameters for tuning (both ARC as well as the baselines). ARC algorithms performed significantly better than the standard AIL algorithms. This shows that ARC algorithms are parameter robust, which is a desirable property for real world robotics.

Ranking the algorithms Table 7 ranks the different algorithms based on the policy return. ARC- f -Max-RKL and ARC-GAIL rank either 1 or 2 in all the environments. f -Max-RKL and GAIL consistently rank 3 and 4 respectively.

Method	Simulation		Real Robot	
	FetchReach	FetchPush	JacoReach	JacoPush
ARC- f -Max-RKL (Our)	1	2	1	1
f -Max-RKL	3	3	3	3
ARC-GAIL (Our)	2	1	2	2
GAIL	4	4	4	4

Table 7: Ranking different Imitation Learning algorithms based on policy return in simulated and real robotic manipulation tasks. Each algorithms is run with 5 random seeds and each seed is evaluated for 20 episodes.

Fig. 18b shows the magnitude of the 2^{nd} 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.

F.3 Sim-to-real transfer of robotic manipulation policies

Table 6 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.

Ranking the algorithms Table 7 ranks the different algorithms based on the policy return. ARC- f -Max-RKL, ARC-GAIL, f -Max-RKL and GAIL consistently ranked 1, 2, 3 and 4 respectively.