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 $f(x) : A \to B$, 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:*

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

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

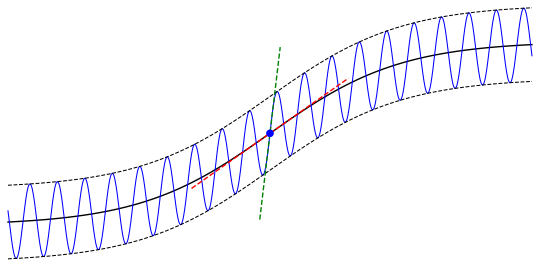


Figure 7: Illustrative example of a function approximation with accurate approximation, E.q.14 but large error in gradient, E.q.15.

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:

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

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

The error in gradient at x_0 is:

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

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 $\nabla f(x_0)$. □

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

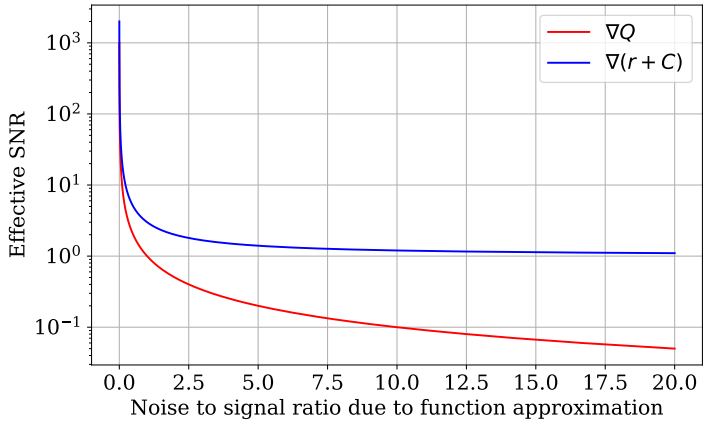


Figure 8: Signal to Noise Ratio (SNR) in Q gradient approximation as noise to signal ratio due to function approximation increases. Higher SNR is better. Using our proposed decomposition, $Q = r + C$, the effective SNR is higher than that without decomposition, when there is large noise due to function 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 $\nabla_a \hat{C}(s,a) = \texttt{snr}_c$ (say).
5. Noise strength = $\mathbb{E}[\epsilon_c^2] = S_n$ (say)
6. By definition of SNR, $\texttt{snr}_c = \frac{S_c}{S_n} \implies S_n = \frac{S_c}{\texttt{snr}_c}$
7. $$\begin{aligned} \text{Final signal} &= \nabla_a r(s,a) + \nabla_a \hat{C}(s,a) && (16)\\ &= \nabla_a r(s,a) + \nabla_a C(s,a) + \epsilon_c && (17)\\ &= (\nabla_a r(s,a) + \nabla_a C(s,a)) + \epsilon_c && (18)\\ &= \text{net true signal + net noise} && (19) \end{aligned}$$

8.

$$\begin{aligned}\text{Net signal strength} &= \mathbb{E}[(\nabla_a r(s,a) + \nabla_a C(s,a))^2] && (20)\\ &= \mathbb{E}[(\nabla_a r(s,a))^2] + \mathbb{E}[(\nabla_a C(s,a))^2] + 2\mathbb{E}[\nabla_a r(s,a)\nabla_a C(s,a)] && (21)\\ &= S_r + S_c + 2S_{r,C} && (22)\end{aligned}$$

9.

$$\begin{aligned}\text{Net SNR} &= \frac{\text{Net signal strength}}{\text{Net noise strength}} && (23)\\ &= \frac{S_r + S_c + 2S_{r,C}}{S_n} && (24)\\ &= \frac{S_r + S_c + 2S_{r,C}}{\frac{S_c}{\texttt{snr}_c}} && (25)\\ &= \texttt{snr}_c \left(\frac{S_r}{S_c} + 1 + \frac{2S_{r,c}}{S_c}\right) && (26)\\ & && (27)\end{aligned}$$

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

$$\begin{aligned}&\text{Net SNR in } \nabla_a \hat{C}(s,a) \geq \texttt{snr}_Q && (28)\\ \implies &\texttt{snr}_c \left(\frac{S_r}{S_c} + 1 + \frac{2S_{r,c}}{S_c}\right) \geq \texttt{snr}_Q && (29)\\ \implies &\texttt{snr}_c \geq \frac{1}{\left(\frac{S_r}{S_c} + 1 + \frac{2S_{r,c}}{S_c}\right)} \texttt{snr}_Q && (30)\end{aligned}$$

Thus, Net SNR in $\nabla_a \hat{C}(s,a)$ is higher than $\texttt{snr}_Q$ 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 $\frac{1}{\frac{S_r}{S_c}+1+\frac{2S_{r,c}}{S_c}} = \texttt{fraction}$. Thus, Eq. 30 reduces to $\texttt{snr}_c \geq \texttt{fraction} \times \texttt{snr}_Q$.

  In other words, even if $\texttt{snr}_C$ is a certain fraction of $\texttt{snr}_Q$, 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,

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

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

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

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 $\texttt{snr}_C$ is a certain fraction of $\texttt{snr}_Q$, 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 $\texttt{snr}_c > \texttt{snr}_Q$ by the same factor.

What determines relative values of $\texttt{snr}_c$ and $\texttt{snr}_Q$ in AIL?

$\texttt{snr}_c$ and $\texttt{snr}_Q$ 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 $\texttt{snr}_c$ and $\texttt{snr}_Q$ to have similar values. In AIL, the adversary reward is dense/shaped which is why $\texttt{snr}_c$ is likely to be greater or at least similar to $\texttt{snr}_Q$.

When is the decomposition likely to help in AIL?

As long as $\texttt{snr}_c$ is similar to higher than $\texttt{snr}_Q$ 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 $\texttt{snr}_c = \texttt{snr}_c$ 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 $\texttt{snr}_c$ is significantly lower than $\texttt{snr}_Q$
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 ofthe Q function [16]. The optimum Q function, Q^* is defined as:

$$\begin{aligned} Q^*(s,a) &= \max_\pi Q^\pi(s,a) \\ &= \max_\pi [r(s,a) + C^\pi(s,a)] \\ &= r(s,a) + \max_\pi C^\pi(s,a) \\ &= r(s,a) + C^*(s,a) \qquad (34) \\ \therefore C^*(s,a) &= Q^*(s,a) - r(s,a) \qquad (35) \end{aligned}$$

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

B.2 Bellman backup for C function

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

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

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.

$$\begin{aligned}
C^\pi(s_t,a_t) &= \mathbb{E} \sum_{k\geq 1} \gamma^k r_{t+k} && (36)\\
&= \mathbb{E}\left(\gamma r_{t+1} + \sum_{k\geq 2} \gamma^k r_{t+k}\right) && (37)\\
&= \gamma \left(\mathbb{E}[r_{t+1}] + \mathbb{E} \sum_{k\geq 1} \gamma^k r_{t+1}\right) && (38)\\
&= \gamma \left(\mathbb{E}[r_{t+1}] + \mathbb{E} \sum_{k\geq 1} \gamma^k r_{t+1+k}\right) && (39)\\
&= \gamma \mathbb{E}\left(r_{t+1} + C(s_{t+1}, a_{t+1})\right) && (40)\\
& && (41)
\end{aligned}$$

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

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

□

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*

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

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

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

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

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.

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

$$= \gamma \max_{s,a} \Bigg| \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s')[r(s',a') + C_1(s',a')]$$

$$- \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s')[r(s',a') + C_2(s',a')] \Bigg| \tag{45}$$

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

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

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

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

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

$$= \gamma ||C_1 - C_2||_\infty \tag{51}$$

$$\therefore ||F_{C_1} - F_{C_2}||_\infty \leq \gamma ||C_1 - C_2||_\infty \tag{52}$$

Eq. 52 implies that iterative operation of $F_{(.)}$ 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. □

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^*. □

C Popular Algorithms

C.1 Policy Iteration using Q function

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

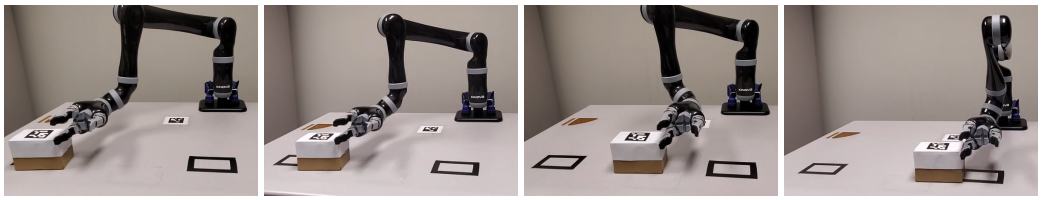
Algorithm 3: Policy Iteration using Q function

Initialize $Q^0(s,a) \forall s,a$;
while π *not converged* **do**
  // Policy evaluation
  for *n=1,2,... 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
  $\pi(s,a) \leftarrow \begin{cases} 1, \text{ if } a = \text{argmax}_{a'} Q(s,a') \\ 0, \text{ otherwise} \end{cases} \quad \forall s,a$

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.



(a) ARC-f-Max-RKL (Our) 1.94 $\pm$ 0.27 cm | (b) ARC-GAIL (Our) 5.95 $\pm$ 0.99 cm | (c) f-Max-RKL 11.48 $\pm$ 0.84 cm | (d) GAIL 17.48 $\pm$ 1.03 cm

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

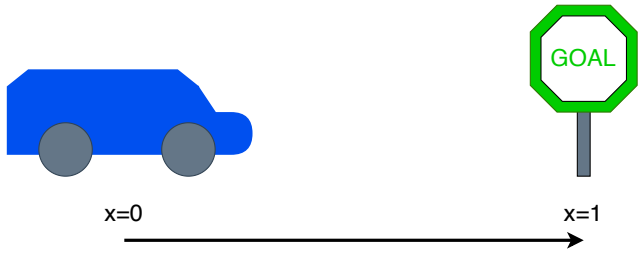


Figure 10: A 1D driving environment where a policy needs to imitate an expert driver which initially drives fast but slows down as the car approaches the goal.

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 that 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:

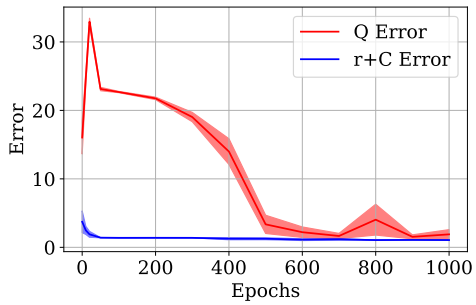
```
def expert_policy(obs):
    gain = 0.1
    goal = 1
    return gain*(goal-obs)+0.1

def agent_policy(obs):
    return 0.15

def reward_fn(obs, a):
    expert_a = expert_policy(obs)
    return -100*(a-expert_a)**2
```

Listing 1: Python code defining the expert policy

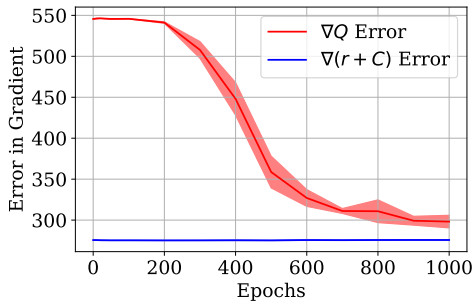
We uniformly sampled states and actions in this environment. We then fit a neural network to the Q function and to the C function by running updates for different numbers of epochs and repeating the experiment 5 times.

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).



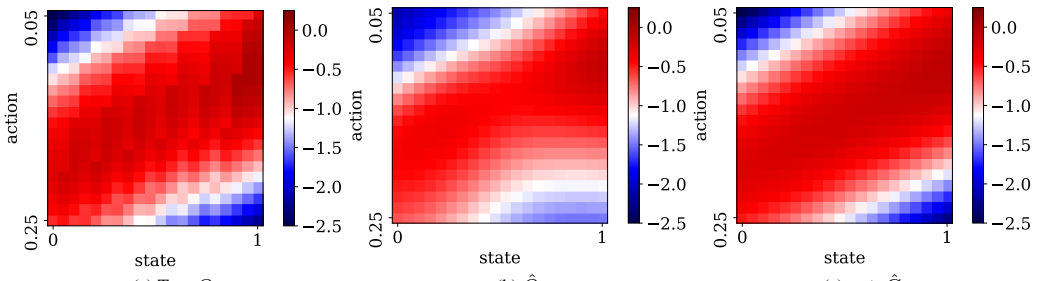
(a) Error in estimating Q



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

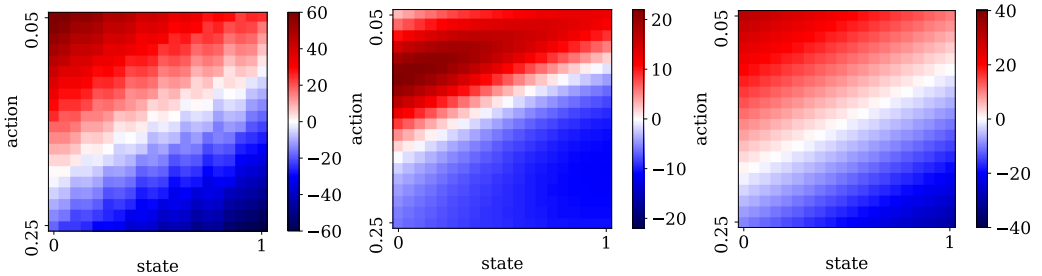
Clearly, the decomposition leads to lower error and variance in estimating both the true Q function and its gradient. Even with slight error with r+C initially, the corresponding error in gradient is much lower for r+C than for Q. Moreover, towards the tail of the plots (after 600 epochs), both Q and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



(a) True Q (b) $\hat{Q}$ (c) $r + \hat{C}$

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.



(a) True $\nabla_a Q(s, a)$ (b) $\nabla_a \hat{Q}(s, a)$ (c) $\nabla_a (r(s, a) + \hat{C}(s, a))$

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_1} 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 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 $\pm\Delta_{\max} = 3.3$cm. In the FetchReach task, the goal is initially located at (15cm,-15cm) + ϵ 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 (0cm,-10cm)+ϵ_{block} and the goal is located at (0cm,-30cm)+ϵ_{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 dispplacements 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:

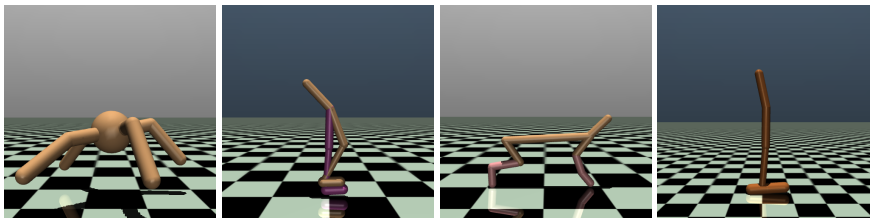
$$\mathbb{E}_{\mathcal{T}\sim\mathcal{D}}\left[\nabla_\theta r(s_1,a_1)+\gamma\nabla_\theta r(s_2,a_2)+\gamma^2\nabla_\theta r(s_3,a_3)+\dots\right] \tag{53}$$

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



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

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

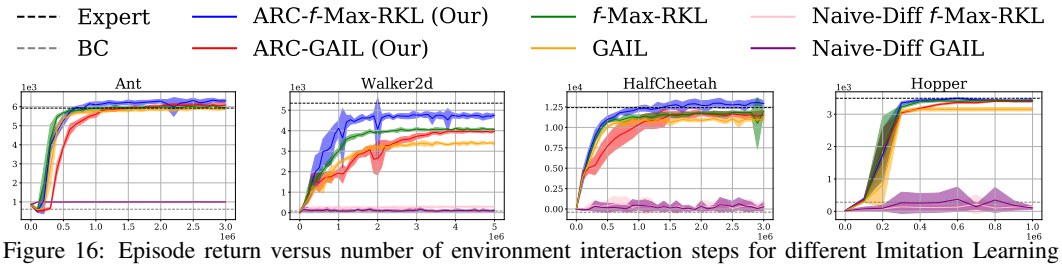


Figure 16: Episode return versus number of environment interaction steps for different Imitation Learning algorithms on Mujoco continuous-control environments.

Method	Ant	Walker2d	HalfCheetah	Hopper
Expert return	5926.18 ± 124.56	5344.21 ± 84.45	12427.49 ± 486.38	3592.63 ± 19.21
ARC-f-Max-RKL (Our)	**6306.25** ± 95.91	**4753.63** ± 88.89	**12930.51** ± 340.02	**3433.45** ± 49.48
f-Max-RKL	5949.81 ± 98.75	4069.14 ± 52.14	11970.47 ± 145.65	3417.29 ± 19.8
Naive-Diff f-Max-RKL	998.27 ± 3.63	294.36 ± 31.38	357.05 ± 732.39	154.57 ± 34.7
ARC-GAIL (Our)	**6090.19** ± 99.72	**3971.25** ± 70.11	**11527.76** ± 537.13	**3392.45** ± 10.32
GAIL	5907.98 ± 44.12	3373.26 ± 98.18	11075.31 ± 255.69	3153.84 ± 53.61
Naive-Diff GAIL	998.17 ± 2.22	99.26 ± 76.11	277.12 ± 523.77	105.3 ± 48.01
BC	615.71 ± 109.9	81.04 ± 119.68	-392.78 ± 74.12	282.44 ± 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 algoritms. 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.

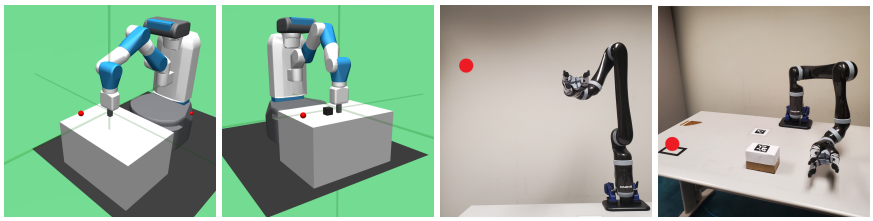
Ranking the algorithms Table 5 ranks the various algorithms in each of these environments. Amongst the 4 AIL algorithms, ARC-f-Max-RKL consistently ranked 1 and GAIL consistently ranked 4. The relative performance of f-Max-RKL and ARC-GAIL varied across the environments, i.e. sometimes the former performed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

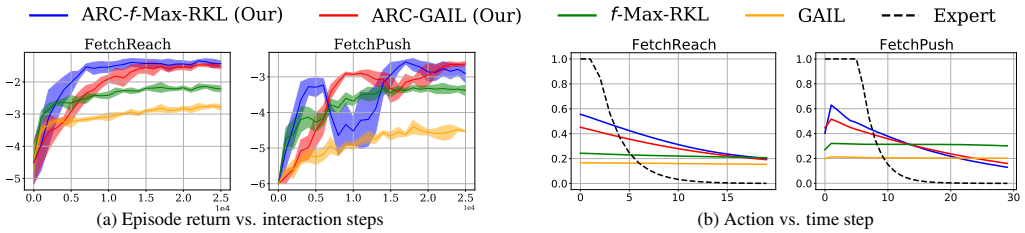
Table 5: Ranking different Imitation Learning algorithms based on policy return in Mujoco environments. Each algorithms is run with 5 random seeds and each seed is evaluated for 20 episodes.

F.2 Imitation Learning in robotic manipulation tasks



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

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].



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

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 $\pm$ 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 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-RKK 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.