Improving Robustness in Inverse Reinforcement Learning

Samin Yeasar Arnob

Master of Engineering
Electrical and Computer Engineering
McGill University
Montreal, Canada

May 2020

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Engineering.

2020/05/06

Acknowledgments

Many thanks to my supervisor, Aditya Mahajan, for his eminence patience, constant support, and guidance. His dedication to all of his students and research has been a great source of inspiration. I am thankful for his time and the opportunity to allow me to pursue the research that interests me most. I would like to thank my friends from the lab: Mohammad Afshari, Jayakumar Subramanian, Nima Akbarzadeh, Fatih Gurturk, Borna Syedana, Amit Sinha for being kind and helpful over the last two wonderful years; Raihan Seraj and Riashat Islam who have mentored and guided me throughout my masters. I can not thank enough to my family and my fiancee. It would be impossible to survive without their constant moral support. I hope someday I will make them proud.

Abstract

In Reinforcement learning (RL) framework, an agent is trained using a designed reward function to solve a particular task. Often designing reward function requires specific reward engineering. Imitation learning provides an alternative to reward engineering ambiguity, where an agent is trained to imitate expert without considering reward from the environment. As these algorithms only learn a policy, they fail miserably when there is a change in environment dynamics. Thus in inverse reinforcement learning (IRL) agent rather learns the underlying objective of expert through approximating reward function and trains a policy accordingly. But it comes with a cost of bad imitation performance. We propose an IRL algorithm based on adversarial learning, which gives comparable performance to state-of-the-art imitation learning algorithm in model-free setting and thus solves an important limitation of prior IRL algorithms over continuous control tasks while still being sample efficient. We show utility of learning reward function in IRL over imitation learning through experiments in transfer learning setup, where expert demonstrations are considered non-existent. We also discuss performance variance in off-policy actor critic algorithms. There has been a lot of work on reducing performance variance using a control variate in policy gradient update. Instead, we propose using doubly robust estimator as control variate in critic estimation. Our hypothesis is that a lower variance in critic estimate will lead to a lower variance in actor performance. We conduct all our experiments in continuous control environments.

Résumé

Dans le cadre de l'apprentissage par renforcement (RL), un agent est form l'aide d'une fonction de rcompense conue pour rsoudre une tche particulire. Souvent, la conception d'une fonction de rcompense ncessite une ingnierie de rcompense spcifique. L'apprentissage par imitation fournit une alternative pour rcompenser l'ambigut technique, o un agent est form pour imiter un expert sans tenir compte de la rcompense de l'environnement. Comme ces algorithmes n'apprennent qu'une politique, ils chouent lamentablement lorsqu'il y a un changement dans la dynamique de l'environnement. Ainsi, dans l'apprentissage par renforcement inverse (IRL), l'agent apprend plutt l'objectif sous-jacent de l'expert en approximant la fonction de rcompense et forme une politique en consquence. Mais cela vient avec un cot de performance d'imitation plus faible. Nous proposons un algorithme IRL bas sur l'apprentissage contradictoire, qui donne des performances comparables l'algorithme d'apprentissage d'imitation de pointe dans un cadre sans modle et rsout ainsi une limitation importante des algorithmes IRL antrieurs sur les tches de contrle continu tout en restant efficace sur l'chantillon. Nous montrons l'utilit de la fonction de rcompense d'apprentissage en IRL sur l'apprentissage par imitation travers des expriences dans la configuration de l'apprentissage par transfert, o les dmonstrations d'experts sont considres comme inexistantes. Nous discutons galement de la variance des performances dans les algorithmes de critique d'acteurs hors politique. Il y a eu beaucoup de travail sur la rduction de la variance des performances l'aide d'une variable de contrle dans la mise jour du gradient de stratgie. Au lieu de cela, nous proposons d'utiliser un estimateur doublement robuste comme variable de contrle dans l'estimation critique. Notre hypothse est qu'une variance plus faible dans l'estimation critique conduira une variance plus faible dans la performance des acteurs. Nous menons toutes nos expriences dans des environnements contrle continu MuJoCo [1].

Contribution of Authors

- Implementation of *Discriminator Actor-Critic* algorithm discussed on chapter 3 was initially implemented as ICLR-2019 Reproducibility Challenge along with *Sheldon Benard* and *Vincent Luczkow* [2]. I would like to thank *Ilya Kostrikov* for looking into our code and helping further on the implementation. The code that has been used in this thesis is an extension of the project. Experiments and respective results mentioned in chapter 3 are done for the purpose of this thesis.

- Doubly robust estimation in actor-critic algorithm discussed in chapter 4 is an on-going research along with *Riashat Islam* and *Doina Precup*. We include part of the work that was submitted to RLDM 2019. Riashat came up with the idea of using control variate in critic. I helped in implementing the code and running the experiments.

Contents

List of Figures

List of Tables

List of Acronyms

RL	Reinforcement Learning
BC	Behavior Cloning
IL	Imitation Learning
AIL	Adversarial Imitation Learning
IRL	Inverse Reinforcement Learning
AIRL	Adversarial Inverse Reinforcement Learning
DAC	Discriminator Actor Critic
DR	Doubly Robust
MCP	Multiplicative Compositional Policies
DDPG	Deep Deterministic Policy Gradient
TD3	Twin-delayed Deep Deterministic Policy Gradient
SAC	Soft Actor Critic

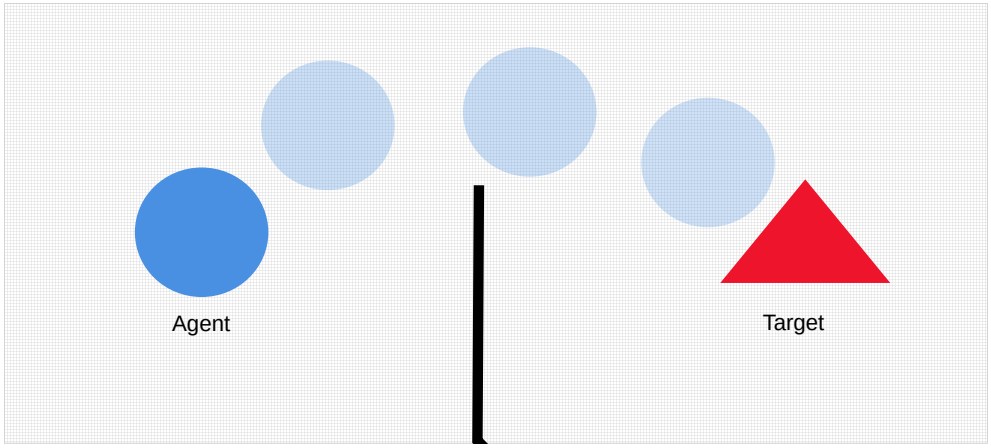
Chapter 1

Introduction

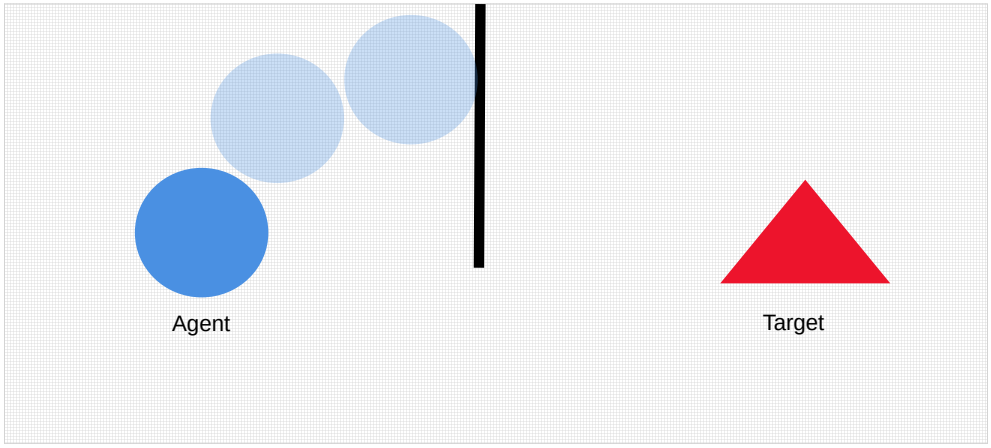
Reinforcement learning (RL) is a useful framework for learning complex behavior, where an agent interacts with an environment and tries to learn an expected behavior through maximizing the cumulative reward over specific time steps. In some applications, there is a natural reward function which can be used (e.g., the score in a video game), but in others, there is no natural reward function, and a reward function is often hand-engineered such that it provides higher rewards for desired behavior (e.g., teaching a humanoid robot to walk). Designing a reward function for a complex environment can be tricky [3]. Poorly designed reward function may lead to fatal mistakes when applied in real-world applications [5].

The behavior of an agent is specified by a *policy*, which is a rule that tells the agent how to act in each state of the environment. The agent leverages the reward from the environment and uses a policy-update-rule that optimizes the policy through trial and error over millions of iterations in the environment. There are different methods to update the policy, and a very common problem exists with these methods, a *high variance in performance*. Which means, the performance variation in learning the task over different runs can be very high. For some runs, the policy succeeds while fails in others depending on randomness in initialization and sampling. Multiple studies show this high variance in performance makes learning unreliable [6, 7, 8, 9].

Since training an RL algorithm requires a large number of iterations, training is often done in simulated environments. However, the simulation environments used for training the agent do not capture all the aspects of the real-world. Thus it is also desired for the



(a) Training environment, where light blue balls represent trajectory of a trained agent.



(b) Test environment, where dynamic of the environment is changed and light blue balls represents agent's trajectory to cross the barrier

Fig. 1.1 Example of policy under changed dynamics.

agent to adapt to dynamic changes where the goal/objective of the task is still the same. Let us consider a simple example from Figure 1.1, where an agent (*blue ball*) is required to reach to a target (*red triangle*). But there is a barrier on the way. Now a reward function is designed such that the agent gets positive reward for visiting the states those are left to the barrier and towards the target, and a higher reward once the agent reaches the target. Otherwise, the reward function gives a negative reward. In Figure 1.1(a), the agent is trained following an RL algorithm that takes the per-step reward from the designed reward function and provides the agent with an optimal policy to go around the barrier on the

left. But consider the scenario in Figure 1.1(b) when the environment dynamic is changed during the test, and the agent is required to take a different direction to reach the target. The agent trained for the first task is not able to redirect itself to the right path on the second task. While the goal is fixed, we refer this adaptability under changing dynamic as *transfer learning* in this thesis.

Thus, in this thesis we focus on three key aspects of the RL (i) *reward function approximation*, where instead of using the reward from environment we consider learning a reward approximator that captures the underlying objective of the task, (ii) *improving policy learning algorithm*, where we propose using a well-known statistical method to tackle high performance variance and (iii) *transfer learning*, where we show the utility of learning the reward approximator through retraining the agent under changed dynamics.

1.1 Alternate approach to human-engineered reward function

An alternative to the human-engineered reward function is to teach the agent the expected behavior using expert demonstrations and two approaches have been proposed in the literature: (i) *Imitation learning* (IL), where an expert demonstrates a desired behavior and without learning any reward function a policy is trained to generate similar behavior [10, 11] and (ii) *Inverse Reinforcement Learning* (IRL), where the agent learns to approximate a reward function and leverages the reward output to learn the policy [12, 13].

Behavior cloning (BC) [14, 15, 10] is a simple form of IL algorithm. BC allows an agent to sample expert data as a pair of observation and corresponding action to train the policy parameters in supervised learning. However, BC suffers from dataset bias and overfitting, and training BC is unstable [16]. Unlike supervised training in classification tasks, RL agent takes sequential decisions where for taking each action, the agent finds itself in a new state. Thus even marginal error in taking action propagates over time. As a result, the agent gets deviated from the expected trajectory it has observed during training. As BC overfits over the training datasets, it fails to generalize over a task [16]. For example, a self-driving car trained with BC is never able to redirect itself to its desired state in the real-world, if not trained with enough samples under such specific scenario [5].

Learning from direct expert policy might be ambiguous, and [3] explained this ambiguity and also emphasized on the importance of learning reward function through a simple example (see Figure 1.2). An expert demonstration is presented to an agent, where expert

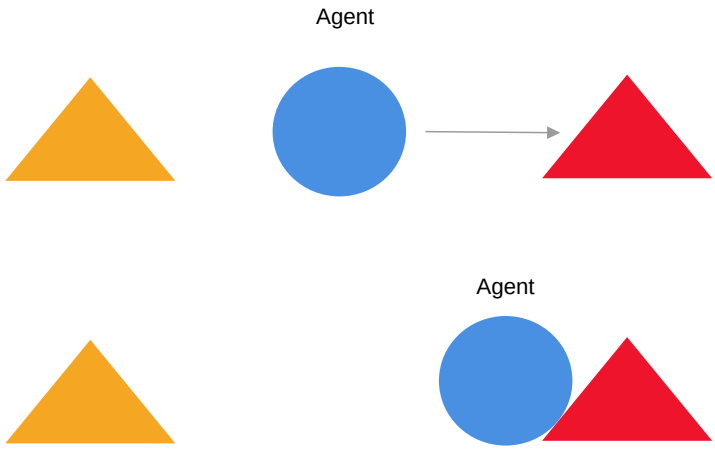


Fig. 1.2 Example of learning policy ambiguity from expert demonstration [3].

(blue ball) rolls on its right and reaches to the red triangle. Now from this demonstration, it is a bit ambiguous what is the objective of this demonstration. (1) The expert might be trying to move away from the orange triangle or (2) it might be trying to move close to the red triangle or (3) it might simply be trying to roll right despite the triangles position or their respective colors or even their existence. *Adversarial Imitation Learning* (AIL)[11] is a class of algorithms in IL that addresses this *ambiguity of learning from expert demonstrations*. AIL does not consider expert demonstration into training its policy; rather it learns a policy distribution that minimizes the difference from expert behavior in *adversarial setting* [11, 17, 18]. Thus unlike BC, AIL algorithms do not overfit over the expert demonstrations, and the learned policy better generalizes over the task. On the self-driving task, policy trained AIL will show more robust behavior. *Generative Adversarial Imitation Learning* (GAIL) [11] introduces AIL where a policy, referred as *generator*, is used to compute a sequence of actions and a binary classifier called *discriminator* classifies the generated behavior from the expert. Even though AIL allows the agent to learn a policy that is more robust, it fails when there is a change in environment dynamics. In figure 1.2, if the environment dynamics are changed, where the orange triangle is removed, and a barrier is imposed between the agent and the red triangle (see Figure 1.1), the agent will fail to reach to the target. Thus a *reward function approximation* is often desired that captures the expert's objective and allows the agent to retrain under dynamic changes.

1.1.1 Policy retraining under changing dynamics

Despite the advantage of AIL, it is still an imitation learning algorithm that only learns a policy. Thus an agent trained with AIL fails when there is a significant change in environment dynamics, and the expert demonstrations are not available. In Figure 1.1(a) let us now consider, the agent is trained with expert demonstrations. Without any expert demonstrations under changed dynamics in Figure 1.1(b), the agent trained for the first task is not able to redirect itself to the right path on the second task. Here, we consider the expert demonstrations non-existent because a real-world application often faces such random variation that is not always possible to predict.

Inverse reinforcement learning (IRL) [12, 13] addresses the importance of learning a reward function as it captures the underlying object of the expert demonstrations. IRL approximates a reward function and uses the approximation to train its policy. Thus as long as the objective of the task is fixed, IRL can use the approximated reward function to retrain its policy to complete the task in Figure 1.1(b). *Adversarial Inverse Reinforcement Learning* (AIRL) [17] leverages the idea of AIL, integrates a reward function approximation along with learning the policy in adversarial setting and shows a promising result in a *transfer learning*, where there is considerable variability in the environment from the demonstration setting.

1.2 Reducing high variance in policy performance

For any class of algorithms in RL, an agent follows a policy learning method to optimize its policy. As discussed earlier, these methods suffer high variance in performance making the learning unreliable [6, 7, 8, 9]. Policy update equation often considers a baseline that controls the policy update from diverging too much to reduce the variance over different runs, and this baseline is referred as *control variate* [19]. The control-variate can be either fixed [20] or a learnable parameter [20, 21, 22, 23] and there exists a perplexity of what is actually a good baseline [24].

Even though IRL[17] algorithms learn policy along with reward function, and recent works [17, 25] show promising results in transfer learning tasks, it comes at the cost of poor imitation performance [26, 17]. Current IRL algorithms fail to provide expert-like imitation performance in continuous control tasks. In this thesis, we propose an *Off-Policy Adversarial Inverse Reinforcement Learning* (Off-policy-AIRL) algorithm, where we focus on improving AIRL [17] and justify the improvement comparing imitation performance with the state-of-the-art AIL algorithm. We experiment with different reward function approximators and show the utility of learning our algorithm over AIL by using the learned reward function in the transfer learning task. We also propose an alternate approach to reduce high variance in the policy learning method to have a further improved imitation performance.

We summarise the contents of the different chapters in this thesis as follow:

- In chapter 2 we formulate the problem setup for this thesis. We show the derivation of different policy learning algorithms and discuss the drawbacks of these algorithm to justify our obvious choice for complex robotic locomotion task. We also discuss early literature on IRL algorithms and its evolution to more recent works and their drawbacks.

- In chapter 3, we discuss on implementation details of proposed off-policy-AIRL algorithm and compare imitation performance with state-of-the-art AIL algorithm.

- In chapter 4, we focus on improving performance of existing policy learning algorithms where we propose using a well-known statistical method to tackle high performance variance, one of the key issue in policy learning literature.

- In chapter 5, we show the utility of learning IRL over imitation learning algorithms. Along with the policy, an IRL algorithm learns a reward function that captures the underlying objective of the expert demonstration. We show learned reward function from off-policy-AIRL algorithm introduced in chapter 2 is cable of retraining its strategy under signification variation in test setting.

Chapter 2

Background on Adversarial Inverse Reinforcement Learning

Reinforcement learning (RL) models a probabilistic system where a decision-maker called an *agent* takes sequential decisions to complete a desired task. In this chapter we start with formulating a framework called Markov decision process (MDP) [27] which is widely used in RL literature to capture the most important aspects of the real-world problem [19]. MDP models the problem using set of *states* and *actions*. To complete the desired task the agent needs a *policy*. At any given state, the agent uses its policy to take an action. For each state-action pair, agent receives a reward using a *reward function* that provides an insight into the underlying objective of the problem. MDP also models a probabilistic *transition function* that allows the agent to transfer to a new state. RL objective is to train the agent so that it learns an optimal policy for the desired task. In this chapter, we discuss the evolution of state-of-the-art policy learning algorithms under the assumption that the true reward function of the MDP is known. In Inverse reinforcement learning (IRL) setting this assumption is excluded and the policy is trained using an *approximated reward function* instead.

We consider an episodic Markov decision process (MDP), defined as tuple $\langle S, A, P, R, T \rangle$, where S,A are the finite state and action space respectively, $P : S \times A \times S \to \mathbb{R}$ is the transition function with $P(s'|s, a)$ being the probability of being at next-state s' after taking action a at state s, $R : S \times A \to \mathbb{R}$ is the reward function $r(s, a)$ and an episode terminates after T deterministic timesteps. For each state $s_t \in S$, the agent takes an action $a_t \in A$

using policy $\pi : S \rightarrow \mathcal{P}(A)$, with probability $\pi(a_t|s_t)$ such that, the action maximizes *expected return* G_t, which is defined as the sum of the rewards until the end of the episode:

$$G_t = R_t + R_{t+1} + ... + R_T.$$

In an episodic MDP, the environment resets at a *terminal state* and starts from an initial distribution of starting states. But in many real-world examples (i.e., robotic application), agent-environment interaction does not stop until reaching a goal and are denoted in [19] as *continuing tasks* where $T = \infty$. This requires maximizing the returns over infinite time-steps. Thus a *discounting factor* γ is considered to represent both episodic and continuing tasks uniformly. An *infinite horizon* MDP can now be represented as episodic MDP using discounted reward and an additional notion of *absorbing state* [19] where the state transitions to itself and returns 0 reward. Now the objective can be represented as to maximize the *expected discounted return*:

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2}... = \sum_{k=0}^{\infty} \gamma^k R_{t+k} \approx \sum_{k=0}^{T} \gamma^k R_{t+k},$$

where $\gamma \in [0, 1)$. A *value-function* represents how good is a state for an agent to be in, and it is quantified by the future reward that can be expected form that state onward. The *value function* $V_\pi(s)$ for any state s, when the following policy π is defined as follows:

$$V_\pi(s) = E_\pi\Big[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)|S_0 = s\Big]. \tag{2.1}$$

Similarly, an *action-value function* $Q_\pi(s, a)$ for any state s and taking action a following policy π is defined as follows:

$$Q_\pi(s, a) = E_\pi\Big[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)|S_0 = s, A_0 = a\Big]. \tag{2.2}$$

An agent interacts with the environment using a policy and gets feedback in terms of reward. For small state space where all possible states can be represented in tabular form, and the *model is known* (referred as *model-based* where transition function P is known), it is possible to compute the value for each state. As the value function quantifies how

good a state is for the agent, an optimal policy is achieved by taking the actions that lead to the states with a higher value. But this method does not scale up to solve a complicated real-world problem where *state space is large* and *model in unknown*(referred as *model-free* where transition function P is not known). It becomes impractical to compute the value for each state. Rather the value function can be approximated and in Section 2.1 we discuss on different methods to *approximate* the *value function*. Value function approximation is essential in the large state-space as it allows the agent to compute value function without exploring all possible states. *Policy gradient algorithm* is another popular class of algorithm where the policy is parameterized, and that can select actions without consulting a value function. A value function approximation may still be required to train the policy parameters but is not required for action selection. Section 2.2 discusses the theory and derivation of the policy gradient method. This class of algorithms takes one or multiple recent episodic experiences into account to update the policy and throws away the data after policy-update, thus not very sample efficient. In this thesis, we use off-policy actor-critic algorithms, which use historical data to update its policy and hence are more sample efficient. Section 2.3 discusses the literature on actor-critic algorithms used in this thesis. Section 2.4 reviews the early literature on inverse reinforcement learning algorithms and their drawbacks. We also discuss how *Maximum Causal Entropy* [28] an IRL framework leads to the progression of current imitation learning algorithms in adversarial setting that allows the agent to perform imitation in complex robotic tasks, and further discuss the advantages and the limitations of recent IRL algorithms in Section 2.5. In Section 2.6, we describe the continuous control tasks which we use in our experiments.

2.1 Policy Evaluation

For an MDP with large state-action space, it is more efficient to evaluate a policy using *value function approximation* instead of learning value for each state. The value function parameters can be updated using *Monte-Carlo* (MC) or *Temporal Difference* (TD) method. As discussed in [19], MC takes rollout for the whole *trajectory* $\tau = \{s_0, a_0, s_1, a_1 ... s_T, a_T\}$ and gives an update of the policy once agent reaches to a terminal condition. On the other hand, TD exploits Markov property [29] and allows per step update. TD does not require any terminal condition to update the policy and has low variance compared to MC [19]. In this thesis, we focus on algorithms that use the TD method for value function

approximation.

2.1.1 Temporal difference method

TD is a *model-free* method to estimate the value function. TD method is used to train a policy from the raw experience without knowing the environment dynamics. It does not wait for the current episode to end rather can learn the current estimate of the value function using bootstrapping. It can bootstrap n-step to make an estimation. In this thesis, we focus on the algorithms that only consider its special case TD(0), which is a single step bootstrapping. Value function for any state s while following policy π can be written as:

$$\begin{aligned} V_\pi(s_t) =& E_{a\sim\pi(s)}[G_t], && (2.3)\\ =& E_{a\sim\pi(s)}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + ...], && (2.4)\\ =& E_{a\sim\pi(s)}[R_t + \gamma G_{t+1}], && (2.5)\\ =& E_{a\sim\pi(s)}[R_t + \gamma V_\pi(s_{t+1})]. && (2.6) \end{aligned}$$

A repeated update of the policy π and then the value function for the state is done using *TD update rule* where TD(0) estimates the value function of the current state and compare the value with sum over reward and discounted value function of the next state:

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha[R_t + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)]. \quad (2.7)$$

Here $R_{t+1} + \gamma V_\pi(s_{t+1})$ is considered as *TD target* and $\delta = TD\ target - V_\pi(s_t)$ is called *TD error*. Similar to value function, we can also learn action-value function $Q_\pi(s, a)$ using the TD method. Now the value function V_ϕ and the action-value function Q_ϕ can be approximated using a neural network where the network parameter ϕ can be learned using TD update [30].

2.2 Policy Gradient Method

In this section, we formalize the *objective function* of the *Policy gradient method* and discuss the *practical implementation* of this class of algorithm. We also show this class

of algorithm is *model-free*, and thus policy trained using the policy gradient method can learn good policy in larger state space. We also discuss about *control variate*, an important concept to reduce high variance in the policy gradient method.

Policy evaluation methods discussed in previous Section 2.1 proposes to learn about good actions by learning its corresponding values. *Policy gradient theorem* [31] tries to learn a parameterized policy π_θ directly. It assigns an objective function $J(\theta)$ and updates the current policy by adjusting the parameter θ in the direction of the gradient of the objective function,

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta). \tag{2.8}$$

For a parameterized policy π_θ if an agent takes action over horizon T, probability of taking a trajectory $\tau = (s_0, a_0, s_1, a_1 ...., s_T, a_T)$:

$$P_\theta(s_0, a_0, s_1, a_1 ...., s_T, a_T) = P(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t), \tag{2.9}$$

$$P_\theta(\tau) = P(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t). \tag{2.10}$$

The objective for the policy is to get optimal parameter θ^* that provides maximum expected reward under the trajectory distribution:

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta E_{\tau \sim \pi_\theta(\tau)} \Big[\sum_{t=0}^{T} r(s_t, a_t) \Big]. \tag{2.11}$$

Here $\sum_{t=1}^{T} r(s_t, a_t) = r(\tau)$ is the sum of reward for each trajectory τ and thus the objective function $J(\theta)$ and its gradient can be written as follows:

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}\Big[\sum_t r(s_t, a_t)\Big], \tag{2.12}$$
$$= E_{\tau\sim\pi_\theta(\tau)}\Big[r(\tau)\Big],$$
$$\stackrel{a}{=} \sum_\tau \pi_\theta(\tau) r(\tau),$$
$$thus\ \nabla_\theta J(\theta) = \sum_\tau \nabla_\theta \pi_\theta(\tau) r(\tau). \tag{2.13}$$

In (a) probability of having a trajectory τ following policy π_θ is to be denoted as $\pi_\theta(\tau)$.

For *practical implementation* of this algorithm the gradient of the objective function (2.13) is converted into an expectation form so that the gradient can be simply computed using the samples from the agent's experience (interactions with the environment). Thus,

$$\bigtriangledown_\theta J(\theta) = \sum_\tau \nabla_\theta \pi_\theta(\tau) r(\tau),$$
$$= \sum_\tau \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} r(\tau),$$
$$\stackrel{a}{=} \sum_\tau \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau),$$
$$= E_{\tau\sim\pi_\theta(\tau)}\Big[\nabla_\theta \log \pi_\theta(\tau) r(\tau)\Big]. \tag{2.14}$$

Here in (a) *likelihood ratio of the gradient* $\frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)}$ [32] is estimated using *log trick* [33]. A further simplification of the gradient objective estimation is done by using equation (2.9),

and an alternative of the gradient term $\nabla_\theta \log \pi_\theta(\tau)$ is derived as follows:

$$
\begin{aligned}
P_\theta(s_0, a_0, s_1, a_1 ...., s_T, a_T) &= P(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t), \\
\pi_\theta(\tau) &= p(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t), \\
\pi_\theta(\tau) &= P(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t), \\
\log\, \pi_\theta(\tau) &= \log\, P(s_0) + \sum_{t=0}^{T} \log\, \pi_\theta(a_t|s_t) + \log\, P(s_{t+1}|s_t, a_t), \\
thus\ \ \bigtriangledown_\theta \log\, \pi_\theta(\tau) &= \bigtriangledown_\theta \Big[\log\, P(s_0) + \sum_{t=0}^{T} \log\ \pi_\theta(a_t|s_t) + \log\, P(s_{t+1}|s_t, a_t) \Big], \\
\bigtriangledown_\theta \log\, \pi_\theta(\tau) &= \sum_{t=0}^{T} \bigtriangledown_\theta \log\, \pi_\theta(a_t|s_t). \qquad (2.15)
\end{aligned}
$$

So from derived equation (2.15) it is concluded that the gradient of the objective function in equation (2.14) does not require to know the dynamics of the model and is independent of the initial state distribution. Thus *policy gradient is model-free*. By computing an average of N number of samples, the gradient of the objective function (2.14) can be approximated and can be re-written as follows:

$$
\begin{aligned}
\bigtriangledown_\theta J(\theta) &= E_{\tau \sim \pi_\theta(\tau)} \Big[\big(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \big) \big(\sum_{t=1}^{T} r(s_t, a_t) \big) \Big], \qquad (2.16) \\
&\approx \frac{1}{N} \sum_{i=1}^{N} \Big[\big(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \big) \big(\sum_{t=1}^{T} r(s_t, a_t) \big) \Big], \qquad (2.17) \\
&\overset{a}{=} \frac{1}{N} \sum_{i=1}^{N} \Big[\big(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \big) \big(\sum_{t'=1}^{t-1} r(s_{t'}, a_{t'}) + \sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \big) \Big]. \qquad (2.18)
\end{aligned}
$$

For $t < t'$, policy at time t' can not affect reward at time t. Thus in (a) the term $\sum_{t'=1}^{t-1} r(s_{i,t'}, a_{i,t'})$ is ignored. For a given state-action pair, sum of the rewards from time

t and onward is equivalent to action-value function $Q(s_{i,t}, a_{i,t})$. Also, a discounted reward (using discount factor γ) allows to compute action-value functions in an infinite horizon $T \to \infty$ and reduces the biases from earlier visited states [19]. Thus,

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \Big[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \sum_{t'=t}^{T} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'}) \Big], \tag{2.19}$$

$$= \frac{1}{N} \sum_{i=1}^{N} \Big[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) Q(s_{i,t}, a_{i,t}) \Big], \tag{2.20}$$

$$= E_{s\sim\rho^\pi, a\sim\pi} \Big[\nabla_\theta \log \pi_\theta(a|s) Q(s,a) \Big]. \tag{2.21}$$

Here ρ^π is trajectory distribution using policy π. Based on the policy gradient theorem [31] the policy parameters θ are updated using the policy gradient equation (2.21). For example, REINFORCE [20] algorithm uses Monte-Carlo estimation and takes following steps to update policy:

1. sample τ^i from policy $\pi_\theta(a_t|s_t)$,
2. compute gradient of the objective: $\nabla_\theta J(\theta) \approx \sum_i \Big[\big(\sum_t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \big) \big(\sum_t r(s_t^i, a_t^i) \big) \Big]$,
3. update parameters: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$.

2.2.1 Reduce variance using baseline/control variate

Policy gradient theorem lays the theoretical foundation for various policy gradient algorithms. Even though unbiased, these algorithms suffers from high variance. Various methods have been proposed to reduce variance [20, 21, 22, 23]. The most common approach is to use *control variate* in policy gradient estimator to reduce the variance.

For example, an objective function $b = E_D[g(s,a)]$ can be estimated with samples $(s_t, a_t)_{t=1}^n$ drawn from a distribution D, which is assumed to have a large variance $var(g)$. Now a control variate $f(s,a)$ can be used to reduce the variance. The modified objective function $E_D[g(s,a) - f(s,a)]$ is an unbiased estimator of b if without losing of the generality, the expectation of the gradient $E_D[\nabla f(s,a)] = 0$. Using control variate in policy gradient

theorem, the gradient of the objective is as follows:

$$\nabla_\theta J(\theta) = E_\pi[\nabla_\theta \log \pi_\theta(a|s)(Q(s,a) - f(s,a))]. \tag{2.22}$$

It is considered as an unbiased estimate of the gradient expectation if the additional term is zero, $E_\pi[\nabla_\theta \log \pi_\theta(a|s) f(s,a)] = 0$. Considering the control-variate/baseline is a design choice and it often has a large impact on reducing variance. As discussed in [19], for some state all actions have high values, and thus its often required to have a baseline to differentiate the higher valued actions from the less highly valued ones. And in many cases, all actions corresponding to the states will have low values, and using a low baseline is expected. Often value function is considered as a control-variate/baseline. Thus:

$$\nabla_\theta J(\theta) = E_\pi[\nabla_\theta \log \pi_\theta(a|s)(Q(s,a) - V(s))], \tag{2.23}$$

$$\stackrel{a}{=} E_\pi[\nabla_\theta \log \pi_\theta(a|s)(A(s,a))], \tag{2.24}$$

where in (a), $A(s,a) = Q(s,a) - V(s)$ is called *advantage function*. As $V(s) = \sum_a Q(s,.)$, advantage function provides an intuition of how good a particular action a is when agent is at state s compared to all possible actions.

2.3 Actor-Critic Algorithm

Policy evaluation methods (discussed in Section 2.1) are indirect in the sense that they focus on constructing a good approximation of the value function and lack reliable guarantees in terms of near-optimality of the resulting policy [34]. On the other hand, in policy gradient methods discussed so far in Section 2.2, the policy parameters gets updated according to the current gradient estimate, which is independent of the past estimates. It throws away older information, and the change in the policy is dominated by the recent experience, thus suffers from high variance.

Actor-Critic [34] aims to use the best feature of both methods and enables a two-time-scale algorithms in which a value function referred as *critic* uses TD learning (discussed in Section 2.1.1), and a policy referred as *actor* is updated in the approximate gradient direction based on the information provided by the critic. Actor-Critic model consists of two networks. In this Section we discuss the formulation of classic actor-critic [34] algorithm

and further dives into formulating more sample efficient *Off-Policy Actor-Critic* algorithms (see Sections 2.3.1, 2.3.2 and 2.3.3). Our focus is to understand the learning procedure of the algorithms used in this thesis.

- *Critic* is a value function approximator, which estimates action-value $Q_\phi(s, a)$. In some algorithms, critic estimates both value and action-value function. Critic uses the reward function to compute TD error and update its network, where

$$\begin{aligned} &\delta_{i,t} = r(s_{i,t}, a_{i,t}) + Q_\phi(s_{i,t+1}, a_{i,t+1}), \\ &\textit{and } L(\phi) = \sum_i ||Q_\phi(s_{i,t}, a_{i,t}) - \delta||. \end{aligned} \tag{2.25}$$

- *Actor* is the policy network $\pi_\theta(a|s)$ that takes action for givens state and is trained through optimizing the objective function (2.21) using the value estimated by the critic:

$$\nabla_\theta J(\theta) \approx \sum_i \Big[\nabla_\theta \log \pi_\theta(a|s) Q_\phi(s, a)\Big]. \tag{2.26}$$

As demonstrated in Figure 2.1, an agent takes action using the actor network. The experience collected using the current actor is used to update the critic network and the updated critic estimate is used to update the actor network.

So far, we have discussed on-policy policy gradient and actor-critic methods. Once the policy network is updated, on-policy algorithms throw away past experiences, and this is not sample efficient [30]. Off-policy actor-critic algorithm increases sample efficiency by using a replay buffer, that stores agents experience as s, a, s', r tuples. A replay buffer can be considered as a mixture of multiple policies [30]. Therefore the sampled data helps to learn an exploratory behavior. During actor and critic update, historical data from the buffer is randomly sampled. Randomizing the samples breaks correlations between sequential samples and therefore reduces the variance of the updates [30]. It greatly stabilizes the policy update and improves its performance. In the off-policy actor-critic algorithm the sampled data is from *behavior policy* μ different than *target policy* π. Critic estimates the action-value function of target policy Q^π using data collected from behavior policy μ. Thus an important sampling (IS) [35] correction is required in the objective function:

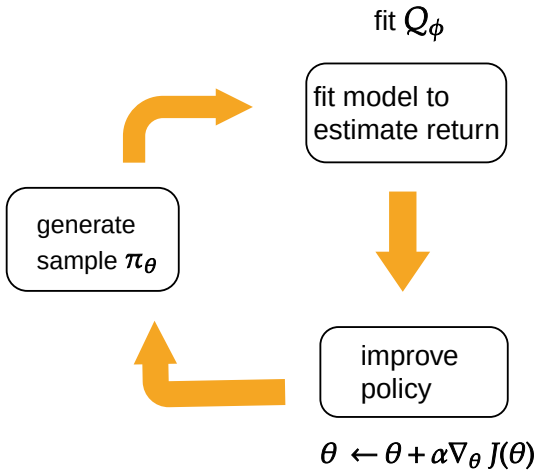


Fig. 2.1 On-policy actor-critic update rule [4].

$$J(\theta) \overset{a}{=} \sum_s \rho^\mu(s) \sum_a \pi(a|s) Q^\pi(s,a), \tag{2.27}$$

$$= E_{s\sim\rho^\mu, a\sim\mu}\left[\frac{\pi(a|s)}{\mu(a|s)} Q^\pi(a,s)\right]. \tag{2.28}$$

Here in (a) consider the objective function of the gradient derived in equation (2.21).

2.3.1 Deep deterministic policy gradient

Deep deterministic policy gradient (DDPG) [36] is an off-policy actor-critic algorithm that uses a deterministic target policy $\pi_\theta : S \to A$. Consider a behavior policy $\mu_\theta \neq \pi_\theta$. The objective function and its gradient to update are following:

$$J(\theta) = \sum_s \rho^\mu(s) \sum_a \pi_\theta(a|s) Q^\pi(s,a), \tag{2.29}$$

$$\stackrel{a}{=} \sum_s \rho^\mu(s) Q^\pi(s, \pi_\theta(s)) = E_{s\sim\rho^\mu(s)}[Q^\pi(s, \pi(a))], \tag{2.30}$$

$$thus\ \nabla_\theta J(\theta) = \nabla_\theta E_{s\sim\rho^\mu(s)}[Q^\pi(s, \pi_\theta(a))],$$

$$\stackrel{b}{=} E_{s\sim\rho^\mu(s)}[\nabla_\theta Q^\pi(s, \pi_\theta(s))]. \tag{2.31}$$

Here, in equation (a) π_θ is deterministic thus inner expectation can be ignored. Unlike objective function for stochastic policy from equation (2.28), it does not require IS correction. To compute policy gradient [33] in equation (b), expectation and differentiation terms are interchanged. Using *chain rule* a gradient expectation can re-written as:

$$\nabla_\theta E_{z\sim d}[F(x(\theta), z)] = E_{z\sim d}[\nabla_\theta F(x(\theta), z)] = E_{z\sim d}[\frac{d}{dx} F(x(\theta), z) \frac{dx}{d\theta}]. \tag{2.32}$$

Thus, the gradient of the critic estimate in equation (2.31) can be re-written as:

$$\nabla_\theta Q^\pi(s, \pi_\theta(a)) = \nabla_\theta [r(s, \pi_\theta(a)) + \gamma \sum_{s'} p(s'|s, \pi_\theta(s)) v^\pi(s')],$$

$$= \nabla_a r(s,a) \nabla_\theta \pi_\theta(s) + \gamma \sum_{s'} \nabla_a p(s'|s,a) \nabla_\theta \pi(s) v^\pi(s')],$$

$$= \nabla_\theta \pi_\theta(s) \nabla_a [r(s,a) + \gamma \sum_{s'} p(s'|s,a) v^\pi(s')], \tag{2.33}$$

$$= \nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s,a). \tag{2.34}$$

And thus using equation (2.31) and (2.34), the policy gradient for DDPG can be derived as the following:

$$\nabla_\theta J(\theta) = E_{s\sim\rho^\mu(s)}[\nabla_a Q^\pi(s,a) \nabla_\theta \pi_\theta(s)]. \tag{2.35}$$

During off-policy update, the policy target network is freezed as $\pi_{\theta'}$ to gain a smoother policy update [30], and the parameter θ' gets periodically updated. Along with a critic network Q_ϕ, it also assigns a critic target network $Q_{\phi'}$. From the buffer (contains sam-

ples from behavior policy), random minibatch of N transitions of $\{s_i, a_i, r_i, s_{i+1}\}$ tuple are sampled. The network update rules are as follows:

- Critic update rule:
  1. Compute critic target: $y_i = r_i + \gamma Q_{\phi'}(s_{i+1}, \pi_{\theta'}(s_{i+1}))$.
  2. Update critic $\phi \leftarrow \frac{1}{N}\sum_{i=1}^{N}(y_i - Q_\phi(s,a))^2|_{s=s_i,a=a_i}$.
- Actor update rule:
  1. Compute policy gradient:$\nabla_\theta J(\theta) = E_{s\sim\rho^\mu(s)}[\nabla_a Q_\phi(s,a)\nabla_\theta \pi_\theta(s)]$
     $\approx \frac{1}{N}\sum_{i=1}^{N}\nabla_a Q_\phi(s,a)|_{s=s_i,a=\pi(s_i)}\nabla_\theta \pi_\theta(s)|_{s=s_i}$.
  2. Update policy parameter: $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$.
- Target networks update: τ controls the slow update of the target network, which allows smaller TD error and smoother policy update, which uses following updates:

$$\theta' \leftarrow \tau\theta + (1 - \tau\theta'), \tag{2.36}$$

$$\phi' \leftarrow \tau\phi' + (1 - \tau\phi'). \tag{2.37}$$

2.3.2 Twin Delayed Deep Deterministic policy gradient

In value-based reinforcement learning methods such as deep Q-learning [30], the function approximation error leads to an overestimated value estimate and sub-optimal policies. This problem persists in the actor-critic setting as well. Actor-Critic algorithm updates its policy with respect to the value estimate of an approximate critic. When the policy is updated using the policy gradient, the update induces overestimation in the value estimate. *Twin Delayed Deep Deterministic policy gradient algorithm* (TD3) [37] proposes a novel mechanism to minimize its effects on both actor and critic. TD3 is built on top of DDPG [36] and uses Double Q-learning [38] to limit the overestimation by taking the minimum value from a pair of critics Q_{ϕ_1}, Q_{ϕ_2}. TD3 also draws the connection between the target networks and the overestimation bias. It suggests delaying policy updates to reduce per-update error which further improves the performance. TD3 uses target network for both actor ($\pi_{\theta'}$) and critic ($Q_{\phi'_1}, Q_{\phi'_2}$) estimate. The network update rules are as follows:

- Critic update rule:

1. Compute critic target: $y_i = r_i + \gamma \min_{k=1,2} Q_{\phi'_k}(s_{i+1}, \pi_{\theta'}(s_{i+1}))$.
2. Update critic $\phi_k \leftarrow \frac{1}{N} \sum_i (y_i - Q_{\phi_k}(s_i, a_i))^2$.

- Actor update rule:

1. Compute policy gradient:$\nabla_\theta J(\theta) = \frac{1}{N} \sum_i \nabla_a Q_{\phi_1}(s, a)|_{s=s_i, a=\pi(s_i)} \nabla_\theta \pi_\theta(s)|_{s=s_i}$.
2. Update policy parameter: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$.

- Target networks update: τ controls the slow update of the target network, which allows smaller TD error and smoother policy update, which uses following updates:

$$\theta' \leftarrow \tau\theta + (1 - \tau\theta'), \tag{2.38}$$

$$\phi'_k \leftarrow \tau\phi'_k + (1 - \tau\phi'_k) \; for \; k \in \{1, 2\}. \tag{2.39}$$

2.3.3 Soft Actor-Critic Algorithm

Soft-actor-critic (SAC) [39] incorporates an entropy-regularizer along with the reward function to drive an agent to a more exploratory behavior while optimizing its policy. *Entropy* gives a measure of how unpredictable a random variable is. If a random variable always takes a single value, then it has zero entropy because it is not unpredictable at all. If a random variable can be any real number with equal probability, then it has very high entropy as it is very unpredictable. Entropy is used to encourage policy to do more exploratory behavior and also to ensure that it does not collapse into repeatedly selecting a particular action. The entropy regularized objective function is as follows:

$$J(\theta) = \sum_{t=0}^{T} \mathbf{E}_{(s_t, a_t) \sim \rho_\pi} [r(s_t, a_t) + \alpha H(\pi(.|s))], \tag{2.40}$$

$$= \sum_{t=0}^{T} \mathbf{E}_{(s_t, a_t) \sim \rho_\pi} [r_\pi(s_t, a_t)]. \tag{2.41}$$

Here $r_\pi(s_t, a_t)$ is entropy augmented reward and α is *temperature parameter*, which controls the stochasticity of the optimal policy. It determines the relative importance of the entropy term against the reward. The conventional objective for policy gradient is recovered when $\alpha \to 0$.

Learning a Q-function using entropy regularized objective is referred as soft Q function. The soft Q-function parameters are trained to minimize the following objective:

$$J(\phi) = E_{(s,a)\sim D}\Big[\frac{1}{2}\times\big(Q_\phi(s_t,a_t) - r(s_t,a_t) + \gamma E_{s\sim\rho^\pi}[V_{\phi'}(s_{t+1})]\big)^2\Big], \tag{2.42}$$

$$\underset{=}{(a)} E_{(s,a)\sim D}\Big[\frac{1}{2}\times\big(Q_\phi(s_t,a_t) - r(s_t,a_t) + \gamma E_{s\sim\rho^\pi, a\sim\pi}[Q_{\phi'}(s_{t+1},a_{t+1}) - \alpha\log\pi_\theta(a_{t+1}|s_{t+1})]\big)^2\Big]. \tag{2.43}$$

Here in (a) the value function can be implicitly parameterized using soft Q-function $V_{\phi'} = Q_{\phi'}(s_{t+1},a_{t+1}) - \alpha\log\pi_\theta(a_{t+1}|s_{t+1})$ and can be optimized with stochastic gradient:

$$\nabla_\phi J(\phi) = \nabla_\phi Q_\phi(s_t,a_t)\Big[Q_\phi(s_t,a_t) - r(s_t,a_t) + \gamma Q_{\phi'}(s_{t+1},a_{t+1}) - \alpha\log\pi_\theta(a_{t+1}|s_{t+1})\Big]. \tag{2.44}$$

In the policy improvement step, the policy is updated towards the exponential of the new soft-Q function. Policy parameters are learned by minimizing the expected Kullback-Leible (KL) divergence with the exponential of the new soft-Q function:

$$J(\theta) = D_{kl}(\pi_\theta(.|s_t)||\frac{exp(Q_\phi(s_t,.))}{Z(s_t)})], \tag{2.45}$$

$$= D_{kl}(\pi_\theta(.|s_t)||exp(Q_\phi(s_t,.) - \log Z(s_t)], \tag{2.46}$$

$$\overset{(a)}{=} E_{s_t\sim\rho^\pi, a_t\sim\pi}[\log\frac{\pi_\theta(a_t|s_t)}{exp(Q_\phi(s_t,a_t) - \log Z(s_t)}], \tag{2.47}$$

$$= E_{s_t\sim\rho^\pi, a_t\sim\pi}[\log\pi_\theta(a|s) - Q_\phi(s_t,a_t) + \log Z(s_t)]], \tag{2.48}$$

$$\overset{(b)}{=} E_{s_t\sim\rho^\pi, a_t\sim\pi}[\alpha\log\pi_\theta(a|s) - Q_\phi(s_t,a_t)]]. \tag{2.49}$$

Where (a) uses KL rules which is $D_{kl}(P||Q) = \sum_x P(x)\log\frac{P(x)}{Q(x)}$. In (b) the likelihood of policy is multiplied with the temperature parameter α and the partition function $Z = \int_a Q_\phi(s,.)\,da$ does not contribute to the gradient with respect to the new policy, thus it can be ignored. For standard critic estimation $Q_\phi(s_t,a_t)$ does not depend on θ but SAC applies a reparameterization trick $a_t = \mu_\theta + \epsilon\sigma_\theta(s_t)$, where $\epsilon\sim N(0,1)$. Thus critic estimate can be re-written as $Q_\phi(s_t,a_t) = Q_\phi(s_t,\mu_\theta(s_t) + \epsilon_t\sigma_\theta(s_t))$ and gradient over ϕ leads to smaller

variance. Thus objective function can be written as:

$$J(\theta) = E_{s\sim\rho^{\pi},\epsilon\sim N(1,0)}[\alpha \log \pi_\theta(f_\theta(\epsilon_t; s_t)|s_t) - Q_\phi(s_t, f_\theta(\epsilon_t; s_t))]. \tag{2.50}$$

The gradient can be approximated using the same trick used in equation (2.32):

$$\nabla_\theta J(\theta) = \nabla_\theta \alpha \log \pi_\theta(a|s) + \nabla_\theta f_\theta(\epsilon; s)\nabla_a[\alpha \log \pi_\theta(a_t|s_t) - Q_\phi(s_t, a_t)]. \tag{2.51}$$

Similar to TD3 algorithm, SAC also uses double-Q learning to reduce over-estimation bias of the critic, and uses target networks with similar update rules for a more controlled update of the policy.

2.4 Inverse Reinforcement Learning

In RL, the fundamental idea is to provide a positive or negative reinforcement to an agent for its action so that the agent can learn the expected behavior. Thus it is required to design a proper reward function. But such hand-engineered reward function can be flawed and deviate from the desired outcome. Also, the way a human and the agent perceive the model of the world can be vastly different. Thus inverse reinforcement learning (IRL) tries to learn the underlying reward function using an approximation R_ψ from the perspective of the agent while it learns a policy π. Initial state distribution $\rho_0(s)$ and reward function $r(s, a)$ are considered to be unknown in IRL setup and can be learned through interaction with the MDP.

In this Section we discuss early literature on IRL algorithm along with their drawbacks.

2.4.1 Linear reward function approximation

It is proposed in [40] to approximates a *linear reward function* using expert trajectory that linearly maps each *state feature* f_s to a reward function

$$r(s) = \psi f_s, \tag{2.52}$$

where ψ is the weight required to be learned by an agent. It considers a finite-horizon MDP with small state-action space that can be represented in a tabular setting. For given

expert trajectory τ reward function, $R(\tau) = \sum_{t=1}^{t=T} r(s_t, a_t) = \Psi^T f_\tau$, here $f_\tau = f_{s_1}, f_{s_2}, ...f_{s_T}$ is called *feature count*.

2.4.2 Matching Feature Expectation

A policy is expected to demonstrate expert-like behavior and thus instead of approximating a reward function, it is proposed in [13] to learn a policy π directly with the objective of matching *feature expectation* $\mu(\pi)$ under a bounded constraint. If π_E is considered to be an expert policy then under a certain constrain objective it is proposed in [13] to approximate the feature expectation of the expert $\mu(\pi_E)$,

$$
\begin{aligned}
&find\ \pi, \\
&s.t.\ ||\mu(\pi) - \mu(\pi_E)||_2 < \eta.
\end{aligned}
$$

Feature expectation $\mu(\pi)$ can also be interpreted as the frequency of visited state feature. A policy is a mapping from states to probability distributions over actions, and the value of the policy for an infinite horizon problem can be expressed in terms of feature expectation as follows:

$$
\begin{aligned}
E[V(\pi|s_0)] &= E\Big[\sum_t \gamma^t r(s,a)|\pi\Big], \\
&= E\Big[\sum_t \gamma^t \psi f_s|\pi\Big], \\
&= \psi E\Big[\sum_t \gamma^t f_s|\pi\Big], \\
&= \psi\mu(\pi). \qquad (2.53)
\end{aligned}
$$

Equation (2.53) shows, comparing the feature expectation is proportional to comparing the value function. For the same expectation of features, policy π gets the same value, and thus it gets the same rewards for the trajectory.

Limitations

There are few drawback of learning a policy through feature matching. For the same set of expert demonstrations, any reward function can correspond to the same policy such that for two different π_1 and π_2

$$\mu(\pi_1) \approx \mu(\pi_E),$$
$$\mu(\pi_2) \approx \mu(\pi_E).$$

Also, many stochastic mixtures of policies correspond to the same feature expectation

$$\mu(\alpha\pi_1 + (1-\alpha)\pi_2) \approx \mu(\pi_E),$$

thus it is hard to choose which policy is better. This algorithm does not scale up to solve complex state-action space.

2.4.3 Maximum Causal Entropy Inverse Reinforcement Learning

Using *Principle of Maximum Entropy* the ambiguity in choosing a distribution over decisions is resolved in [28]. Under the maximum entropy objective, it is proposed in [28] to maximize the log likelihood of the trajectory that satisfies the constraint of feature matching. The objective function is following:

$$\max_{\psi} \sum_{\tau} -p(\tau;\psi)\log p(\tau;\psi), \tag{2.54}$$
$$s.t. \sum_{\tau} p(\tau;\psi) f_\tau = \mu(\pi_E) = \bar{f}, \tag{2.55}$$
$$and \sum_{\tau} p(\tau;\psi) = 1. \tag{2.56}$$

For given M expert demonstrations $D = \{\tau_1, \tau_2, ...\tau_M\}$, expert feature count can be expressed as:

$$\bar{f} = \frac{1}{M} \sum_{\tau_d \in D} f_{\tau_d}. \tag{2.57}$$

Using Lagrange multipler this objective maximization function can be rewritten as follows:

$$\begin{aligned}
\frac{d}{dp(\tau;\psi)}\big[-\sum_{\tau} p(\tau;\psi)\log p(\tau;\psi) - \lambda \times (\sum_{\tau} p(\tau;\psi) - f_\tau - \bar{f}) - \mu \times (\sum_{\tau} p(\tau;\psi) - 1)\big] &= 0, \\
[-\sum_{\tau} \log p(\tau;\psi) - \sum_{\tau}(1)] - \lambda \sum_{\tau} f_\tau - \mu \sum_{\tau}(-1) &= 0, \\
-\sum_{\tau} \log p(\tau;\psi) + \lambda \sum_{\tau} f_\tau - \sum_{\tau}(1+\mu) &= 0, \\
-\sum_{\tau}[\log p(\tau;\psi) + \lambda f_\tau] - \lambda_0] &\overset{(a)}{=} 0, \\
p(\tau;\psi) &= e^{-\lambda_0 - \lambda f_\tau}, \\
p(\tau;\psi) \overset{(b)}{=} \frac{e^{\lambda f_\tau}}{z} = \frac{e^{\psi^T f_\tau}}{z} \quad p(\tau;\psi) &\propto e^{R_\psi(\tau)}.
\end{aligned} \tag{2.58}$$

Here λ, μ are Lagrange constants, $Z = \int_\tau e^{R(\tau)} d\tau$ is the partition function and $e^{-\lambda_0}$ is constant over $\tau \in D$. Also (a) considers $\lambda_0 = 1 + \mu$ and (b) uses following derivation:

$$\begin{aligned}
\sum_{\tau} p(\tau;\psi) &= 1, \\
\sum_{\tau} e^{-\lambda_0} e^{-\lambda f_\tau} &= 1, \\
e^{-\lambda_0} &= \frac{1}{e^{-\lambda f_\tau}} = \frac{1}{Z}.
\end{aligned} \tag{2.59}$$

A new objective (2.58) derived from the feature matching can be thus expressed as the probability of trajectories with equal reward/cost are equally likely, and the probability of trajectories with higher rewards are exponentially more likely. The objective function (2.58) can also be interpreted as solving a maximum likelihood problem. It is expected to learn a reward function that maximizes the maximum likelihood of demonstration of the expert,

$$\max_{\psi} J(\psi) = \max_{\psi} \mathbf{E}_{\tau \sim D}[\log p(\tau;\psi)]. \tag{2.60}$$

Limitations

This limitation of [28] is that the transition function $P(s'|s,a)$ is assumed to be known and applied on small state-action space.

2.5 Inverse Reinforcement Learning in Adversarial setting

Adversarial imitation learning and inverse reinforcement learning evolved from the maximum casual entropy IRL framework [28], which considers an entropy regularized MDP with the goal to find the optimal policy $\pi*$ that maximizes the expected entropy-regularized discounted reward under π,T and ρ_0:

$$\pi^* = \arg\max_{\pi} \mathbf{E}_{\tau\sim\pi}\Big[\sum_{t=0}^{T} \gamma^t r(s_t,a_t) + H(\pi(.|s_t))\Big]. \tag{2.61}$$

Here $\tau = (s_0, a_0 ... s_T, a_T)$ defines the sequence of states and actions induced by policy and $H(\pi(.|s_t))$ is the discounted causal entropy of policy π. In imitation task, expert demonstrations are given. It is expected from the agent to produce trajectory which is similar to the expert, and infer the reward function for given demonstration $D = \tau_1, \tau_2 ... \tau_n$.

2.5.1 Generative Adversarial Imitation Learning

Generative Adversarial Imitation Learning (GAIL) [11], a model-free algorithm, is the first framework to draw connection between *generative adversarial network* (GAN) and *imitation learning*. GAIL is also the first imitation learning algorithm to show expert-like imitation performance on the continuous state-action space. It tries to learn the occupancy measure of the expert using a maximum entropy regularized objective function [28]. Despite being inspired from an IRL framework [28], GAIL is an imitation learning algorithm as it only learns the policy and does not learn underlying reward function.

For a policy π, *occupancy measure* ρ_π is defined as the visitation frequency of a state-action pair for following the policy, which is analogous to the distribution of the policy π. The occupancy measure for state s and action a is as follows:

$$\rho_\pi(s,a) = \pi(s,a)\sum_t \gamma^t P(s_t = s|\pi). \tag{2.62}$$

GAIL uses a generative model or generator G that acts as the policy. The purpose of the generator is to learn the occupancy measure of the expert ρ_{π_E} without directly being provided with the expert demonstrations. A discriminator D is used to produce a reward signal to train the generator. A discriminator is a binary classifier which tries to distinguish the data distribution of the generator ρ_π from the expert's ρ_{π_E}. The objective of a generator is to minimize the difference with expert occupancy measure, while the discriminator wants to maximize the difference. Hence the entropy regularized objective is as follows:

$$\begin{aligned} distance_{\min_{\pi_\theta}\max_{D_\psi}}(\rho_\pi, \rho_{\pi_E}) = \min_{\pi_\theta}\max_{D_\psi} \Big[& \mathbf{E}_{\pi_\theta}[\log D_\psi(s,a)] \\ & + \mathbf{E}_{\pi_E}[\log(1 - D_\psi(s,a))] - \lambda H(\pi_\theta)\Big], \end{aligned} \tag{2.63}$$

where the entropy term H is a policy regularizer controlled by $\lambda \geq 0$ and π_θ, π_E are policies corresponding to the generator and the expert. GAIL uses a single neural network as a generator. For any given state, it tries to take expert like action. Discriminator takes state-action pairs as data and computes the likelihood of the input coming from an expert. Generator uses a reward function $R(s,a) = -\log D_\psi(s,a)$ in order to train its network parameters.

Limitations

GAIL derives a model-free (does not require to know the transition function P) algorithm that is able to solve high-dimensional complex imitation learning problems. The reward function used to train the policy provides a constant negative reward, which incentivizes an agent to complete the task sooner and detracts from following expert trajectory [26]. GAIL is sample efficient in terms of the number of expert demonstrations required but not quite sample efficient in terms of required interaction with the environment to learn the policy. For some experiments GAIL requires as many as 25 millions of iterations to get expert-like performance. For being an imitation learning algorithm, instead of learning

cost/reward function, it only recovers the expert policy and thus fails miserably when there is a dynamic change in the test environment.

2.5.2 Generative Adversarial Inverse Reinforcement Learning

A discriminator function is proposed in [18] that learns a policy as well as a cost/reward function using the following discriminator/reward optimization:

$$D_\psi = \frac{p_\psi(\tau)}{p_\psi(\tau) + q(\tau)} \overset{a}{=} \frac{1/z * e^{R_\psi}}{1/z * e^{R_\psi} + q(\tau)}, \tag{2.64}$$

$$thus\ \mathcal{L}_{dis} = \max_{D_\psi} \Big[E_{\tau\sim p}[\log D_\psi(\tau)] + E_{\tau\sim q}[\log(1 - D_\psi(\tau))] \Big], \tag{2.65}$$

$$= \min_{D_\psi} \Big[- E_{\tau\sim p}[\log D_\psi(\tau)] - E_{\tau\sim q}[\log(1 - D_\psi(\tau))] \Big], \tag{2.66}$$

where (a) considers equation (2.58), $p(\tau)$ is the data distribution and $q(\tau)$ is the policy distribution. But no experimental demonstration of policy performance is provided in [18]. *Adversarial inverse reinforcement learning* (AIRL) [17] shows, when a complete trajectory is considered to update a discriminator, it suffers from high variance issue and that leads to a poor performing generator. Instead, AIRL computes discriminator update using (s,a,s') tuple:

$$D_{\psi,\omega} = \frac{p_{\psi,\omega}(s, a, s')}{p_{\psi,\omega}(s, a, s') + q(s, a)}, \tag{2.67}$$

$$= \frac{e^{f_{\psi,\omega}(s,a,s')}}{e^{f_{\psi,\omega}(s,a,s')} + q(s, a)}, \tag{2.68}$$

$$= \frac{e^{f_{\psi,\omega}(s,a,s')}}{e^{f_{\psi,\omega}(s,a,s')} + \pi_\theta(a|s)}, \tag{2.69}$$

where reward function $f_{\psi,\omega}(s, a, s') = r_\psi(s, a) + \gamma\Phi_\omega(s') - \Phi_\omega(s)$ computes *disentangled reward* [17] [41], $r_\theta(s, a)$ is the reward network and the reward shaping term Φ_ω is controlled by the dynamics. A reward function that disentangle expert objective from dynamics of the environment is called a *disentangled reward function.* A class of reward transformations is described in [12] that preserves the optimal policy. Without prior knowledge of the dynamics, this reward transformation ensure constant optimal policy which remains constant

under the changing dynamics. For transition dynamic $P(s,a) = s'$ reward function can be written as:

$$\hat{r}(s,a) = r(s,a) + \gamma\phi(s') - \phi(s), \tag{2.70}$$

$$= r(s,a) + \gamma\phi(P(s,a)) - \phi(s). \tag{2.71}$$

For two different MDPs, M and M'; if the transition dynamics are P and P', for disentangled reward, reward approximator $r(s,a)$ does not have any impact due to the change in the environment dynamics.

Limitations

Unlike GAIL, AIRL learns a reward function. The learned reward function is useful retraining the agent when there is a significant change in the test environment. But AIRL gives bad imitation performance compared to GAIL. Both of these algorithms use on-policy update and throws away the data after policy update step, which is sample inefficient.

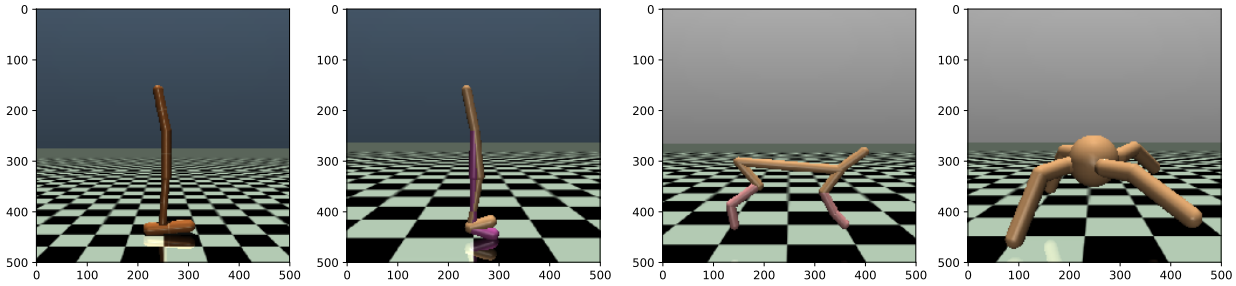
2.6 Environment Details

For conducting experiments in this thesis, we use continuous control tasks available in *gym* [42] from *OpenAI*. It uses *MuJoCo* [1] physics engine that allows a fast and accurate simulation environment for robotics experiments, widely used in RL experiments for comparison algorithm performance and setting benchmarks [33, 36, 11, 17, 26, 39, 37, 43].

We run experiments on locomotion tasks where a robot of different physical structure tries to move forward without falling. The observation for these environments includes joint angles, joint velocities, coordinates of the center of mass and constraint forces. Both observation and actions are continuous values and bounded within constant $[-1, 1]$. Environments that are used in our experiments are the following:

- *Hopper*: A two-dimensional one-legged robot hops forward as fast as possible. A stable hopping gait has to be learned without falling.
- *Walker*: A two-dimensional robot walks on two legs. A more challenging environment than hopper, since it has more degrees of freedom and is also prone to falling.

- *Half-Cheetah*: A two-dimensional cheetah robot runs forward.
- *Ant*: A four-legged robot walks as fast as possible. It has higher degree of freedom than others.



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

Fig. 2.2 MuJoCo environments.

Chapter 3

Off-Policy Adversarial Inverse Reinforcement Learning

In this chapter, we discuss the implementation of *Off-Policy Adversarial Inverse Reinforcement Learning* (off-policy-AIRL) algorithm. Our objective is to introduce an IRL algorithm which ensures improved imitation performance while being sample efficient. We conduct experiments using different reward approximators to get expert-like imitation performance, and in principle learning a good reward function further helps to retrain policy under changing dynamics [17, 3], which is discussed in Chapter 5.

Off-Policy-AIRL algorithm gets the inspiration from *Discriminator Actor-Critic* (DAC) [26], which is an adversarial imitation learning algorithm. DAC addresses two problems with AIL framework: reward function bias and sample inefficiency. Reward functions used in these algorithms induce an implicit bias, which may work for some environments but, in many cases, cause sub-optimal behavior. GAIL [11]) can learn a policy using demonstrations from a few experts, but they require a large number of interactions with the environment to learn expert like behavior. We reproduce the results of DAC and demonstrate the imitation performance on MuJoCo [1] continuous control tasks.

Like any other imitation learning algorithms, DAC only learns the policy. Similar to GAIL [11] (see Section 2.5.1), DAC uses a binary discriminator. We use the best of both worlds. In off-policy-AIRL, we replace the binary discriminator and use the discriminator function introduced in *Adversarial Inverse Reinforcement Learning* (AIRL) (see Section 2.5.2) [17] to learn reward function along with the policy. From [26, 17], it is evident that

AIRL provides a poor imitation performance compared to AIL. We hypothesize that using the features introduced in DAC will also help in IRL algorithm, and in this Chapter, we experimentally test our hypothesis. Off-policy-AIRL improves comparable performance to DAC, which is the state-of-the-art imitation algorithm, while still being sample efficient by using Soft-Actor-Critic (SAC) [39], an off-policy actor-critic algorithm as the generator.

Implementations of the discriminator from AIRL does not work off the shelf. We experiment on different ways to compute the reward function for the generator update to ensure expert-like imitation performance and compare imitation performance with DAC on continuous control tasks [1].

3.1 Motivation Behind DAC: Issues with Adversarial Imitation Learning Algorithms

DAC uses an off-policy reinforcement learning algorithm to improve upon the sample efficiency of existing methods; it extends the learning environment with absorbing states and uses a new reward function to achieve unbiased rewards. DAC shows these changes are essential to remove the survival bias and solves the sample inefficiency problem of GAIL [11].

3.1.1 Reward Bias

AIL uses the reward functions $r(s,a) = -\log(1 - D(s,a))$ or $r(s,a) = \log(D(s,a))$. The former assigns *only positive rewards*. Per-step positive reward encourages the agent to survive longer and to collect additional rewards. However, if the task is time-sensitive and must be done quickly, the agent can behave sub-optimally, as the positive reward may incentivize the agent to move in loops or take a longer path. The latter, in contrast, assigns *only negative rewards*. This reward function assigns per-step penalty, which encourages to take shorter path, but is not suitable for tasks with survival bonuses. As such, a new reward function is needed to generalize the imitation learning paradigm across tasks.

Furthermore, imitation learning improperly handles the terminal state and introduces an implicit reward prior that can affect the policy performance. In particular, many imitation learning implementations [44, 11, 17] and MDPs omit absorbing states s_a. Thus the environment assigns 0 reward to the terminal/absorbing state and implicitly adds bias

to the reward learning process. If the reward formulation for a given task is $r(s,a) = -\log(1 - D(s,a))$, an agent is never incentivized to explore the absorbing state. The strictly positive reward function rewards the policy for avoiding the absorbing state, regardless of how the discriminator classifies the state-action pair. Conversely, a strictly negative reward pushes the policy to end the trajectory as soon as possible, exploring the absorbing state that the expert may never visit.

Reward of the terminal state is defined in DAC [26] as $R_T = r(s_T, a_T) + \sum_{t=T+1}^{\infty} \gamma^{t-T} r(s_a, .) = r(s_T, a_T) + \gamma \frac{r(s_a,.)}{1-\gamma}$, where $r(s_a, 0)$ is learned reward, instead of just $R_T = r(s_T, a_T)$.

3.1.2 Sample Inefficiency

For some tasks, GAIL requires as few as 200 expert transitions from 4 expert trajectories to robustly imitate the expert and achieve expert-like performance. However, for other tasks, GAIL requires as many as 25 million policy transitions sampled from the environment to reach this point. GAIL is sample inefficient and is not suitable for many real-world applications.

DAC uses *Twin-delayed deep deterministic actor-critic* (TD3) [37], an off-policy actor-critic algorithm as the generator. In value-based reinforcement learning methods such as *DDPG*, function approximation error is known to lead to overestimated value estimates and sub-optimal policies. TD3 [37] uses Double-Q learning (discussed in 2.3.2) to resolve this issue and improves the imitation performance as well.

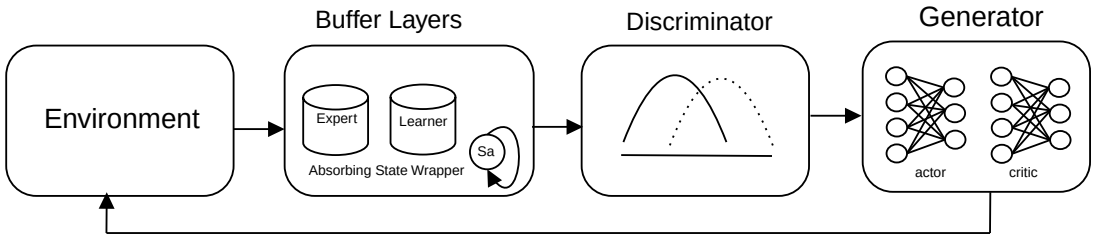


Fig. 3.1 Discriminator-Actor-Critic imitation learning framework.

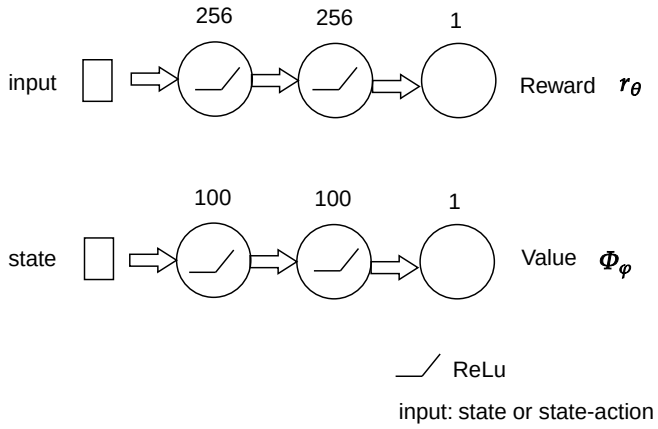


Fig. 3.2 Discriminator networks of off-policy-AIRL framework.

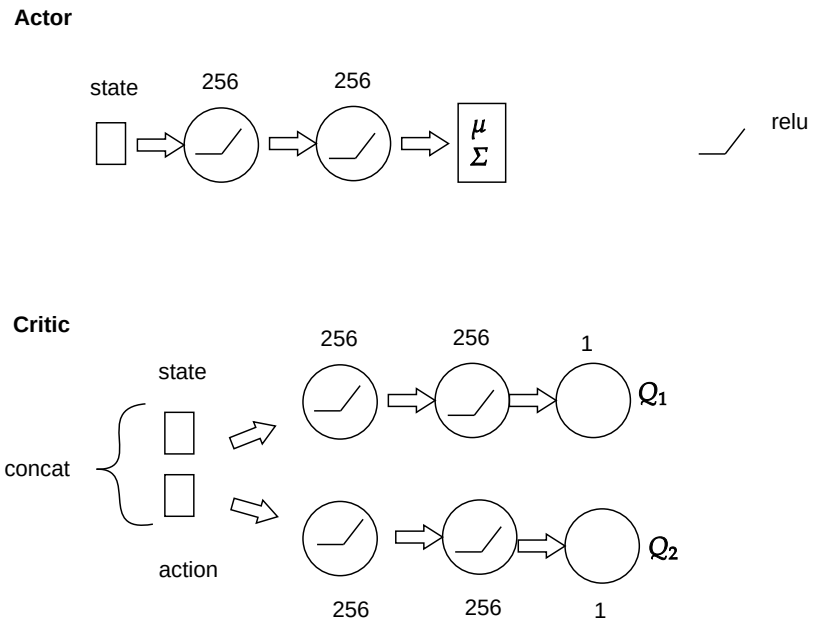


Fig. 3.3 Generator networks of off-policy-AIRL framework.

Algorithm 1 off-policy-AIRL Algorithm

Input: Expert Replay Buffer R_E
while $n \neq 10^6$ **do**
    episodic timestep = 0
    while *not done* **do**
        $a \leftarrow \pi(a|s)$
        $\mathcal{R} \leftarrow \mathcal{R} \cup s, a, ., s'$
        episodic timestep += 1
        n += 1
        if done WarpAbsorbingState
    for `episodic timesteps` **do**
        sample random mini-batch $\{(s_b, a_b, ., s_b^{'})\}_{b=1}^{B} \in \mathcal{R}$
        sample random mini-batch $\{(s_{b_E}, a_{b_E}, ., s_{b_E}^{'})\}_{b=1}^{B} \in \mathcal{R}_{\mathcal{E}}$
        Compute loss: $\mathcal{L}_{\psi,\omega} = \sum_{b=1}^{B}[-\log D_{\psi,\omega}(s_b, a_b, s_b^{'}) - \log(1 - D_{\psi,\omega}(s_{b_E}, a_{b_E}, s_{b_E}^{'}))]$
        update $D_{\psi,\omega}$ + gradient-penalty
    for `episodic timesteps` **do**
        sample random mini-batch $\{(s_b, a_b, ., s_b^{'})\}_{b=1}^{B} \in \mathcal{R}$
        use current reward function $\{(s_b, a_b, r_b, s_b^{'})\}_{b=1}^{B} \leftarrow r_\psi(s_b)$
        update SAC

3.2 Update Rule for off-policy-AIRL

We store the agent-environment interactions in a replay buffer $\mathcal{R}$ to utilize them during the off-policy update of generator and discriminator. At the end of each episode, we update the discriminator and the generator for the same episodic timesteps. We use an expert buffer $\mathcal{R}_{\mathcal{E}}$ to store the expert trajectories and sample the $(s_E, a_E, {s_E}')$ pair in batch while updating the discriminator. In this Section, we discuss the formulation of generator and discriminator update rules.

3.2.1 Discriminator Update Rule

During discriminator update, we randomly sample mini-batch of state-action-next-state pairs $(s, a, ., s')_b$ from both expert buffer $\mathcal{R}_{\mathcal{E}}$ and replay buffer $\mathcal{R}$. We use equation (2.69) derived in [17] to compute the discriminator output,

$$D_{\psi,\omega} = \frac{e^{f_{\psi,\omega}(s,a,s')}}{e^{f_{\psi,\omega}(s,a,s')} + \pi_\theta(a|s)}. \tag{3.1}$$

Here we use potential-based reward function $f_{\psi,\omega}(s, a, s')$, which can be referred as disentangled reward function (discussed in Section 2.5.2) for being indifferent to the changes in transition dynamics and can be expressed as:

$$f_{\psi,\omega} = r_\psi(s) + \gamma\Phi_\omega(s') - \Phi_\omega(s), \tag{3.2}$$

$$= r_\psi(s) + \gamma V_\omega(s') - V_\omega(s), \tag{3.3}$$

where reward approximator r_ψ can be a function of state, state-action or state-action-next-state and Φ can be any function that gives a measure of being at any state s. Similar to [17] we use value function V_ω as the reward shaping terms. We find the best imitation performance using state dependent reward function discussed in the following Section 3.2.3. As discussed in Section 2.5, output of the discriminator predicts likelihood of being an expert and thus the objective of the discriminator (from the objective function (2.66)) for our algorithm is to minimize the following binary cross-entropy loss:

$$min\ \mathcal{L}_{\psi,\omega} = min\ \sum_{b=1}^{B}[-\log D_{\psi,\omega}(s_b, a_b, s_b^{'}) - \log(1 - D_{\psi,\omega}(s_{b_E}, a_{b_E}, s_{b_E}^{'}))]. \tag{3.4}$$

3.2.2 Generator Update Rule

In IRL setting, no reward is received from the environment. A reward function is approximated and then used to train a policy. We use the reward function approximator proposed in [17]

$$\hat{r}_{\psi,\omega}(s, a, s') = \log D_{\psi,\omega}(s, a, s') - \log(1 - D_{\psi,\omega}(s, a, s')). \tag{3.5}$$

Using equation (2.69) the reward approximator can be written as:

$$\begin{aligned}\hat{r}_{\psi,\omega}(s,a,s') &= \log \frac{e^{f_{\psi,\omega}(s,a,s')}}{e^{f_{\psi,\omega}(s,a,s')} + \pi(a|s)} - \log \frac{\pi_\theta(a|s)}{e^{f_{\psi,\omega}(s,a,s')} + \pi_\theta(a|s)}, \\ &= f_{\psi,\omega}(s,a,s') - \log \pi_\theta(a|s), \qquad (3.6) \\ &\overset{(3.3)}{=} r_\psi(s) + \gamma V_\omega(s') - V_\omega(s) - \log \pi_\theta(a|s). \qquad (3.7)\end{aligned}$$

This reward function is also referred as *entropy regularized reward function*. For improving imitation performance and sample efficiency we use SAC as the generator. SAC is an off-policy algorithm, which samples $(s, a, s', done)$ tuple from replay buffer $\mathcal{R}$ when trains the actor and the critic networks. For our algorithm we experiment with different reward approximator which is discussed in the following Section 3.2.3 and use the trained reward function $\hat{r}_{\psi,\omega}(s,a,s')$ in the gradient objective (2.44) to update critic network Q_ϕ through the following gradient step

$$\nabla_\phi J_Q(\phi) = \nabla_\theta Q_\phi(s_t,a_t)\Big[Q_\phi(s_t,a_t) - \hat{r}_{\psi,\omega}(s,a,s') + \gamma Q_{\phi'}(s_{t+1},a_{t+1}) - \alpha \log \pi_\theta(a_{t+1}|s_{t+1})\Big]. \tag{3.8}$$

Similar to [37, 39] we update the actor network π_θ for every second update of the critic Q_ϕ to have a smoother policy update. Using the updated critic estimation, actor takes gradient steps to update its network parameters following the gradient objective derived in equation (2.51):

$$\nabla_\theta J(\theta) = \nabla_\theta \alpha \log \pi_\theta(a|s) + \nabla_\theta f_\theta(\epsilon; s)\nabla_a[\alpha \log \pi_\theta(a_t|s_t) - Q_\phi(s_t,a_t)]. \tag{3.9}$$

3.2.3 Reward Function Selection

An off the shelf implementation of AIRL discriminator [17] does not work. Unlike AIRL, we train an off-policy actor-critic algorithm to improve sample-efficiency. We experiment with different way to compute the reward function for the generator update.

In AIRL, the value function $V_\omega(s)$ is used to measure additional shaping terms. We experiment with different variations of reward approximator $\hat{r}_{\psi,\omega}$ to update policy and disentangled reward function $f_{\psi,\omega}$ to compute discriminator output so that we find the combination of these two approximators that gives the best performance in off-policy-AIRL.

Implementation 1:

- Policy update using : $\hat{r}_{\psi,\omega}(s,a,s') = \log D_{\psi,\omega}(s,a,s') - \log(1 - D_{\psi,\omega}(s,a,s'))$
- Disentangled reward function : $f_{\psi,\omega} = r_\psi(s,a) + \gamma V_\omega(s') - V_\omega(s)$

Implementation 2

- Policy update using : $\hat{r}_{\psi,\omega}(s,a,s') = \log D_{\psi,\omega}(s,a,s') - \log(1 - D_{\psi,\omega}(s,a,s'))$
- Disentangled reward function : $f_{\psi,\omega} = r_\psi(s) + \gamma V_\omega(s') - V_\omega(s)$

Implementation 3

- Policy update using : $\hat{r}_{\psi,\omega}(s,a,s') = r_\psi(s,a)$
- Disentangled reward function : $f_{\psi,\omega} = r_\psi(s,a) + \gamma V_\omega(s') - V_\omega(s)$

Implementation 4

- Policy update using : $\hat{r}_{\psi,\omega}(s,a,s') = r_\psi(s)$
- Disentangled reward function : $f_{\psi,\omega} = r_\psi(s) + \gamma V_\omega(s') - V_\omega(s)$

For implementation (1) and (2), we use the exact policy update rule used in AIRL [17]. In implementation (1)a state-dependent reward function $r_\psi(s)$, and in implementation (2) a state-action dependent reward function $r_\psi(s,a)$ is considered. As discussed in the previous section, the reward computed using equation (3.5) is equivalent to an entropy regularized reward function. We evaluate the disentangled reward $f_{\psi,\omega}$ as both state-dependent $r_\psi(s)$ and state-action dependent $r_\psi(s,a)$ functions. For implementation (3) and (4), we followed a similar comparative study. We use the direct output from reward approximator r_ψ to train the generator, and discard the entropy regularization and the reward shaping terms. As discussed in DAC [26], strict negative or positive reward approximator induces bias. Similar to DAC, our reward approximator r_ψ gives both positive and negative rewards thus does not suffer reward biasing.

3.3 Experimental Analysis

In this section, we present the results of DAC implementation. We also conduct an ablation study by comparing the performance of DAC with a different off-policy generator, which is not done in [26].

To perform imitation, a learner needs expert demonstrations. We compare the performance of TD3, SAC in RL setting (considering the reward from the environment) for expert selection. We train the algorithms on MuJoCo tasks to collect expert data. We also collect expert trajectories for the environments used in AIRL [17] to compare transfer learning performance in Chapter 5.

We demonstrate experiments using different reward functions to update the policy for off-policy-AIRL and compare the imitation performance with DAC.

3.3.1 Expert Data Collection

We require expert demonstrations to train an agent in an imitation task. We first compare the performance of TD3, SAC on MuJoCo continuous control task, and also on environments used in [17] in RL setting for $1e6$ iterations. As discussed in Section 2.3.3, *temperature parameter* α controls the stochasticity in SAC. Thus we compare SAC performance for both fixed and learnable temperature parameters. We select 0.2 as the fixed temperature value for SAC. We get the performance curve by seeding 10 experiments from 0-9.

As can be seen from Figure 3.4, except for Ant-v2 in MuJoCo tasks, keeping the temperature value fixed for SAC gives better performance than TD3. For the environments from [17] (*PointMazeLeft-v0, PointMazeRight-v0, CustomAnt-v0, DisabledAnt-v0*), SAC with learned temperature seems to perform marginally better. Thus we select SAC with learned temperature as our expert for all the following experiments.

We collect 50 trajectories from 5 best performing expert policies. The policy update is done by random sampling from the buffer, to break the correlation between samples and help the learning policy to better generalize on the task [30]. We use the same hyperparameter (i.e., temperature) that gives the best results for the expert for all the experiments discussed in this thesis.

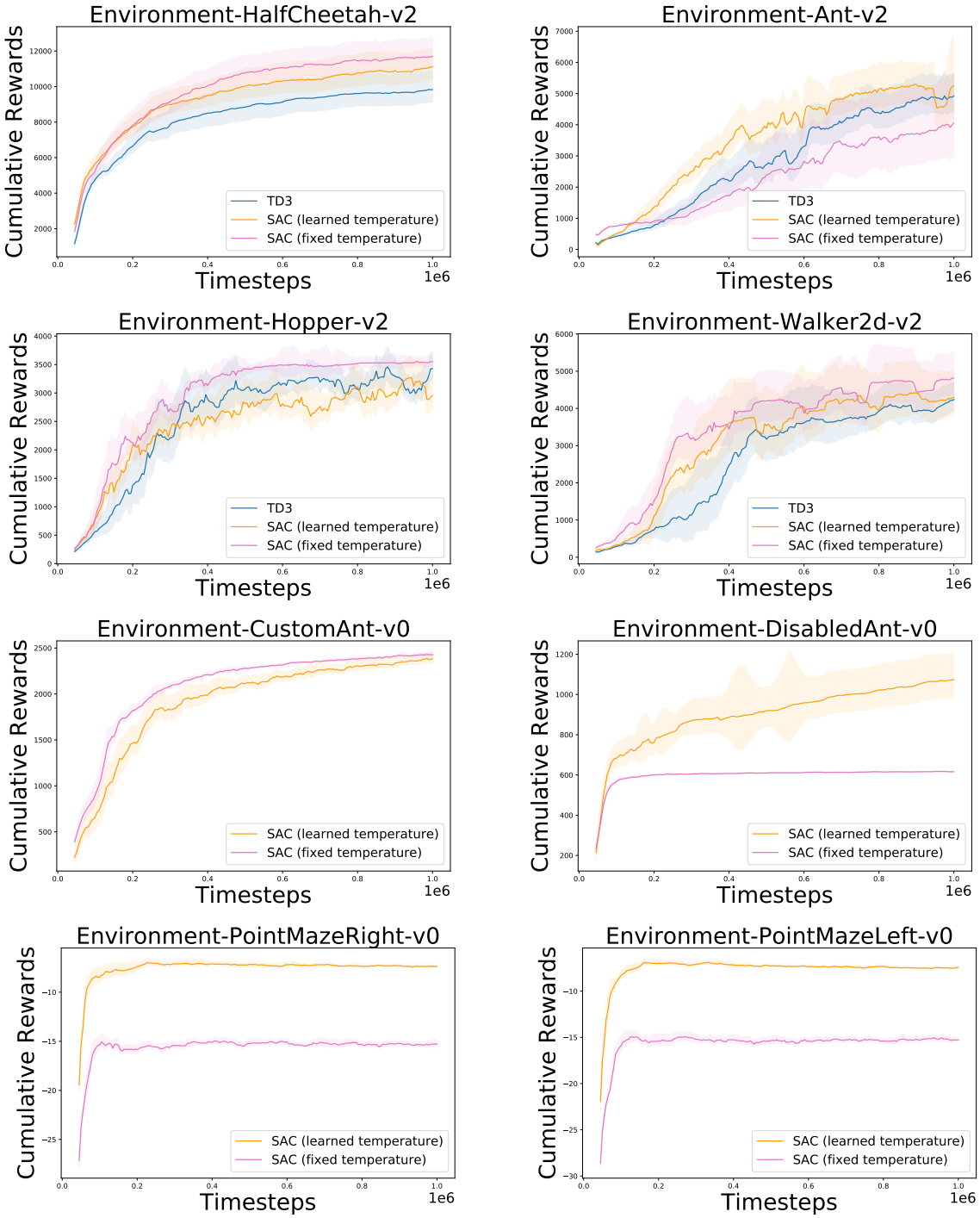


Fig. 3.4 Performance of policy gradient algorithms (TD3, SAC) over continuous control tasks in RL setting.

Environment	Avg returns over seed 0-9	Temperature parameter
HalfCheetah-v2	11083	fixed 0.2
Ant-v2	4294	learned
Hopper-v2	3544	fixed 0.2
Walker2d-v2	4490	fixed 0.2
CustomAnt-v0	2430	fixed 0.2
DisabledAnt-v0	1089	learned
PointMazeLeft-v0	-7.37	learned
PointMazeRight-v0	-7.33	learned

Table 3.1 Expert Average Performance over 0-9 seed after 10^6 iterations.

3.3.2 Reproduced Results of DAC

Experimental Setup

In DAC, the discriminator is 2 layer neural network with 100 hidden units and tanh activation. As discussed in Section 2.5.1, discriminator in AIL algorithms acts as a binary classifier where it predicts the likelihood of state-action pairs coming from an expert. At the beginning of the training, the generator takes random exploratory actions. Within a few iterations of the discriminator update, discriminator easily classifies the expert from the generator data and overfits on the training data [45, 46]. Thus to make the learning more stable, DAC uses *gradient penalty* [46]. In [26] TD3 is used as the generator, which is an off-policy actor-critic algorithm. In our implementation, we use the same generator architecture as [26, 37], which is 2 layer MLP with ReLU activations and 400 and 300 hidden units, respectively.

We train all the neural networks with the Adam optimizer [47] from *PyTorch*, which uses the default learning rate of $1e^{-3}$. We train on *MuJoCo* [1] continuous control task environments, and the performance curves (Figure 3.5) are obtained using the mean over 10 experiments for 0-9 seeds and evaluated (after each 5000 interaction with the environment) over 10^6 steps in the environment. During each evaluation, we store the average performance of 10 runs.

Performance of DAC with Different Policies

No ablation study is conducted in [26]. Thus, we also investigate the performance of DAC with *SAC* as the generator. SAC has an entropy regularized objective function (2.51) that

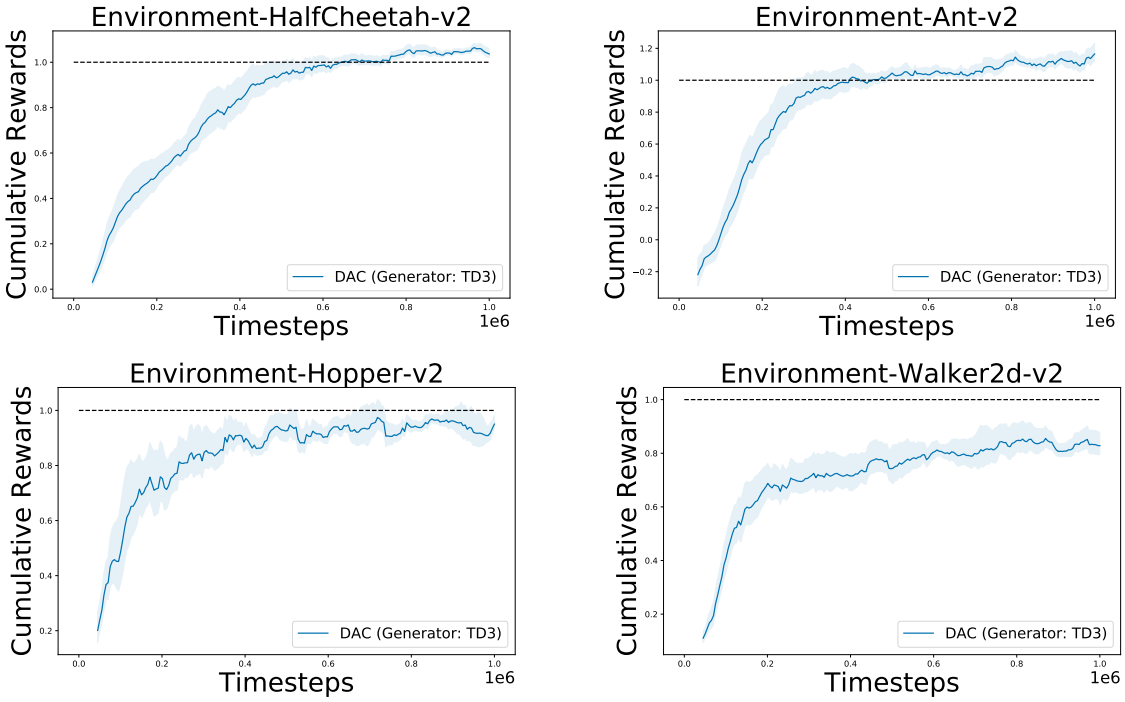


Fig. 3.5 Performance comparison of DAC with expert performance over MuJoCo environments.

allows exploration while taking the best possible action. Our experiment using SAC in Figure 3.6 also show that DAC is sample efficiency.

3.3.3 Performance of off-policy-AIRL

In AIL [11, 26] there is no separate reward function approximator to provide the reward signal during the policy update. Thus a reward function $\hat{r}_\psi(s,a) = \log D_\psi(s,a)$ is introduced in GAIL [11] to update its policy, which provides a penalty/negative-reward for each state-action pair. As discussed in Section 3.1.1, it is argued in [26] that this reward function introduces bias and leads to sub-optimal policies. Thus DAC introduces an alternate reward function $\hat{r}_\psi(s,a) = \log(D_\psi(s,a)) - \log(1 - D_\psi(s,a))$ which gives both positive and negative reward output. But the reward function is still computed using direct discriminator output, and it has a drawback. In AIL algorithms, the discriminator is a binary classifier. Once the generator is trained, it provides expert like behavior, and the discriminator can no longer differentiate between the expert and generator samples. As AIL constantly updates the discriminator using sampled data coming from similar distribution, the discriminator

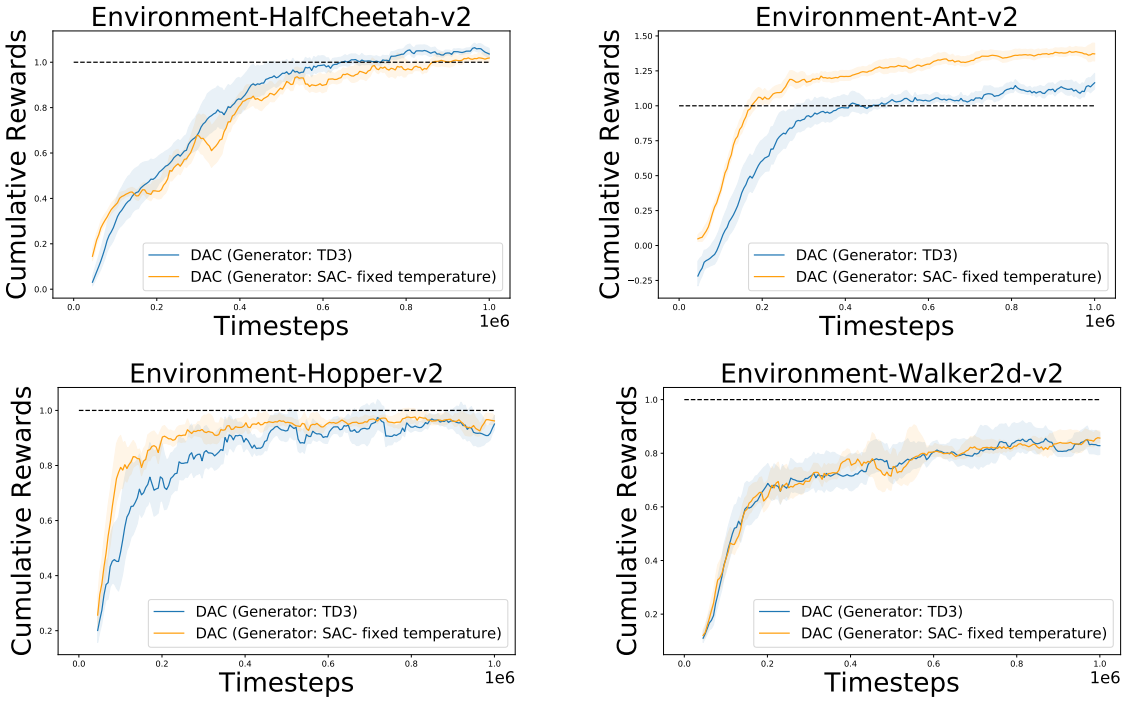


Fig. 3.6 Performance of DAC for different generator (TD3, SAC).

becomes incompetent, and the discriminator can no longer differentiate the generated data from the experts. On the other hand, [18, 17] incorporates a reward function approximator within the discriminator function which can be observed from equations (3.5) and (3.3). Thus the reward function $\hat{r}_{\psi,\omega}(s,a,s') = \log D_{\psi,\omega}(s,a,s') - \log(1 - D_{\psi,\omega}(s,a,s'))$ can be used without being biased and can be reused for future training as well.

We conduct an experiment on *CustomAnt-v0* [17] to select the appropriate reward function from different choices of reward approximator discussed in Section 3.2.3. We train off-policy-AIRL with different configuration of reward approximators. We conduct this experiment for keeping the seed fixed 0. Figure 3.7(a) plots the performance curve using *actual reward from the environment* (hand-engineering reward function), and it is evident that using the reward signal directly from the reward function approximator (*Implementation 4*) gives a drastic improvement in policy performance in imitation tasks. We also see in Figure 3.7(b) that, the cumulative reward achieved from reward approximator r_ψ replicates the actual performance curve displayed in 3.7(a).

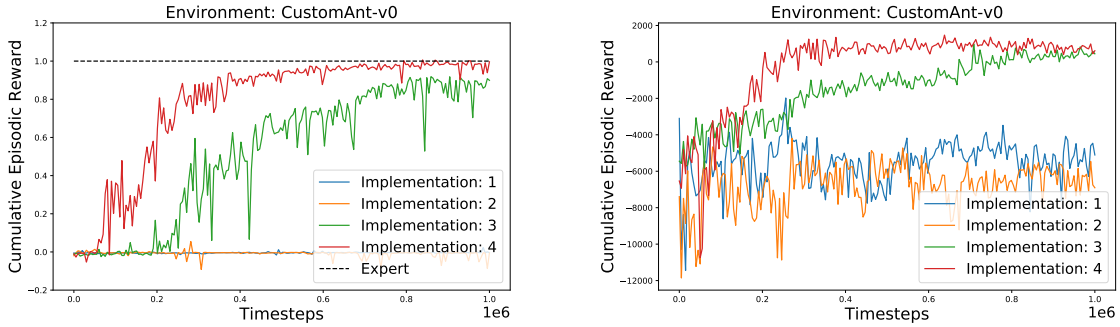


Fig. 3.7 Performance comparison of off-policy-AIRL for different update rules. Here, Figure (a) give the performance curve using actual reward from the environment (b) shows the cumulative reward achieved from reward aprximator $\hat{r}_{\psi,\omega}$.

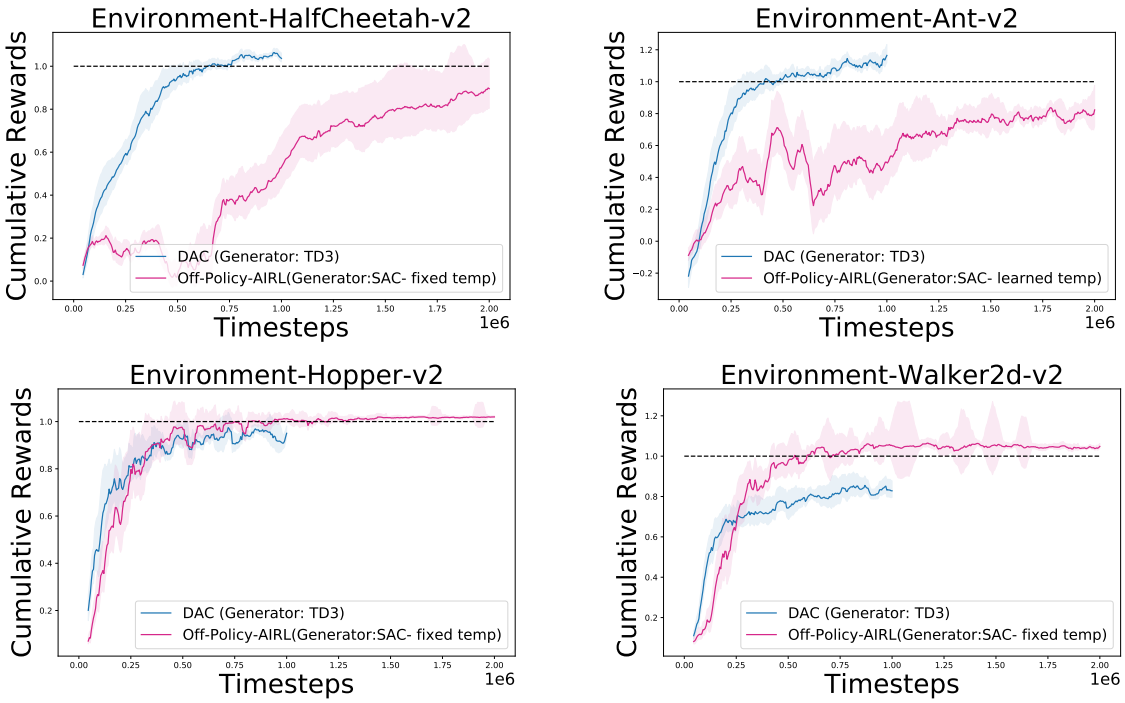


Fig. 3.8 Performance Comparison of DAC and off-policy-AIRL over continuous control task.

It is evident from Figure 3.8 that using implementation-4, off-policy-AIRL gives a better imitation performance than DAC in Hopper-v2 and Walker2d-v2 environments within 10^6 iterations, while provides a comparable performance for HalfCheetah-v2 and Ant-v2

environments after 20^6 iterations.

3.4 Discussion

Adversarial Imitation Learning (AIL) is a class of algorithms in *Reinforcement learning* (RL), proposed in [11], tries to imitate an expert without taking any reward from the environment and does not provide expert behavior directly to the policy training. Rather, an agent learns a policy distribution that minimizes the difference from expert behavior in adversarial setting. *Adversarial Inverse Reinforcement Learning* (AIRL) leverages the idea of AIL, integrates a reward function approximation along with learning the policy, and shows the utility of IRL in the transfer learning setting. But compared to AIL, AIRL algorithm does not perform well in continuous control tasks. *Discrimintor Actor Critic* (DAC) [26], a recently proposed AIL algorithm, identifies the reward bias and the inappropriate use of the absorbing layer in imitation learning. DAC also proposes to use a sample efficient off-policy actor-critic algorithm to further improves the imitation performance. Our hypothesis is that the features introduced in DAC to improve imitation performance should also work for an IRL algorithm.

We start this Chapter discussing our implementation of the DAC algorithm over continuous control tasks. While DAC [2] only focuses on using a deterministic policy (TD3) as a generator, we also show the scope of using a stochastic policy (SAC) in the algorithm, and it gives similar imitation performance.

We propose an off-policy IRL algorithm in adversarial learning which learns policy using its approximated reward function. We use SAC as the generator to improve imitation performance while still being sample efficient. We compare different way to compute reward functions for off-policy generator update and experimentally find simple state-dependent reward network works best for our algorithm and similar to DAC it provides both positive and negative rewards thus the policy does not suffer from reward bias (discussed in Section 3.1.1).

We compare the imitation performance of off-policy-AIRL with DAC, which is the state-of-the-art imitation learning algorithm, to show its persistent expert like performance over continuous control environments. Even though off-policy-AIRL gives a comparable imitation performance to DAC, it has a certain advantage. It approximates a reward function that, in principle, should capture the objective of the expert demonstrations [3].

And the utility of the learned reward function is tested the Chapter 5.

Chapter 4

Doubly Robust Actor-Critic in Off-Policy Algorithm

Off-policy actor-critic algorithms are sample efficient because they use historical data to learn the policy [33, 36, 37, 39]. In actor-critic algorithms, the policy, known as the actor, can be updated through the *policy gradient algorithm* [33]. Similar to policy gradient algorithms [48, 49], they suffer from *high variance* in actor performance where the performance variation in learning the task over different runs can be very high and makes the learning unreliable [6, 7, 8, 9]. This is also evident in the experiments from Chapter 3 (see Figures 3.4 and 3.8). In actor-critic algorithms, the actor and the critic have a separate objective function (discussed in Section 2.3). The gradient objective of the actor uses the critic estimation to update its network parameters (see the policy gradient objective (2.31)) and thus the performance of actor largely depends on the action-value estimation of the critic. A better estimate of the critic is a key step towards the stability of the gradient update of the actor [37]. Our hypothesis is that a better critic estimate leads to lower variance in actor performance. In this chapter, we test our hypothesis by using *doubly robust estimator* to compute critic estimation of the off-policy actor-critic algorithms.

As discussed in Section 2.2.1, several state-action dependent baseline/control-variates in the policy gradient estimator have been introduced [50] to reduce performance variance. Still, choosing the right baseline is not clear [24]. Instead of adding a control variate at the policy gradient estimator directly, we propose using a control variate in estimating the critic.

Doubly robust (DR) estimator has a sound theoretical understandings in bandit literature [51] and sequential decision-making [52]. The idea of doubly robust off-policy evaluation as a control variate has been proposed in [53] to reduce variance in value estimation for both finite and infinite horizon settings, and it is proven to be unbiased. We extend the idea of doubly robust (DR) off-policy evaluation [52] to actor-critic algorithms. We implement DR estimator on *deterministic* actor-critic algorithms because they do not require importance sampling correction in off-policy evaluation (see equation (2.30)).

In Section 4.1, we define the MDP and discuss the theory of DR estimation and its implementation in recent works. In Section 4.2 we discuss the high variance issue in policy gradient methods and propose the DR estimator method in off-policy value evaluation to reduce performance variance. In Section 4.3 we formulate the DR estimator in a model-free method for off-policy value evaluation in the actor-critic algorithm and show that using deterministic policy reduces the computation complexity compared to stochastic policy. In Section 4.4 we use *Deep deterministic policy gradient* (DDPG) [36] and *Twin-delayed deep deterministic policy gradient* (TD3) [37] (discussed in Sections 2.3.1 and 2.3.2) algorithms in our experiments to evaluate whether policy improvement persists over actor-critic algorithms in general. The contributions of this Chapter are as follows:

- Implement the DR estimator in actor-critic algorithm.
- Unlike control variate implemented on gradient of policy, we implement DR estimator as control variate in critic estimation.
- We estimate the DR estimator in a model-free setting.

4.1 DR estimator in Off-policy value evaluation

For this chapter, we consider episodic RL problems with a MDP defined as $M = \langle S, A, P, R, \gamma \rangle$, where S is the finite state space, A is the finite action space, $P : S \times A \times S \to \mathbb{R}$ is the transition function with $P(s'|s, a)$ being the probability of being at next-state s' after taking action a at state s, $R : S \times A \to \mathbb{R}$ is the reward function $r(s, a)$, and $\gamma \in [0, 1)$ is the discount factor. At each t discrete time step, for given state $s_t \in S$, an agent takes action $a_t \in A$ with probability $\pi(a_t|s_t)$ following policy $\pi : S \to \mathcal{P}(A)$. Value function for any

state s, when following policy π can be defined as follows:

$$V_\pi(s) = E_\pi\Big[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) | S_0 = s\Big]. \tag{4.1}$$

In off-policy value evaluation, data is considered to be sampled from *behavior policy* μ and our *goal* is to estimate value for *target policy* π. There are two popular ways in which we can compute off-policy value evaluation:

1. Regression estimator
2. Importance sampling estimator

4.1.1 Regression estimator

In off-policy value evaluation value of target policy V_π can be computed using the recursive process (derived in equation (2.6)) as follows:

$$V_\pi(s) = E_{a\sim\mu(.|s),s'\sim P(.|s,a)}[r(s,a) + \gamma V_\pi(s')]. \tag{4.2}$$

This process is completed following a two-step regression-based procedure: the first step is to fit an MDP model from data and approximate reward function $\hat{R}$ and transition model $\hat{P}$ using sampled data from behavior policy; second, compute the value function $\hat{V}_\pi$ using the fitted model. Regression based approach has a relatively low variance and works well when the model can be learned to satisfactory accuracy [52].

4.1.2 Importance sampling estimator

In statistics, important sampling (IS) is used to estimate properties of a distribution when the samples are generated from a different distribution. In order to find the expectation of function f, where true distribution is p but data sampled from q distribution, importance sampling estimator can be used as follows:

$$
\begin{aligned}
E_{x\sim p(x)}\big[f(x)\big] &= \sum_x p(x)f(x) \\
&= \sum_x \frac{q(x)}{q(x)} p(x)f(x) \\
&= \sum_x q(x)\frac{p(x)}{q(x)} f(x) \\
&= E_{x\sim q(x)}[\rho \times f(x)] \qquad (4.3) \\
&\approx \frac{1}{n}\sum_{i=1}^{n} \rho_i f(x_i). \qquad (4.4)
\end{aligned}
$$

Here, $\rho_i = \frac{p(x_i)}{q(x_i)}$ is the per step importance sampling ratio. Using *step-wise* IS estimator [52] value function from equation (4.1) can be written as:

$$V_\mu(s) = \sum_{t=0}^{T} \gamma^t \rho_t r(s_t, a_t) \qquad (4.5)$$

where, per-step importance ration $\rho_t = \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}$. And the recursive form of value function from equation (4.2) can be written as:

$$V_\pi(s) = \rho_t[r(s,a) + \gamma V_\pi(s')]. \qquad (4.6)$$

4.1.3 Doubly Robust Estimation

DR estimation is a statistical approach for estimation from incomplete data. DR estimator is initially proposed in RL for the contextual bandits [51], where a variance of *importance sampling* and a *regression model* is used to compute value function. The term "*doubly robust*" refers to the fact that if either one of these functions is properly estimated then, it ensures a good estimation of the value function.

Contextual bandit can be defined as MDP with horizon 1 and sample trajectory is the

form of (s, a, r). DR estimator for contextual bandits is defined as following:

$$V_{DR}(s) = \hat{V}(s) + \rho\big[r - \hat{R}(s,a)\big] \tag{4.7}$$

where, $\hat{R}(s,a)$ is the estimated reward through model-fitting, $\rho = \frac{\pi}{\mu}$ is the IS corrections for the mismatch in the action distributions and $\hat{V}(s) = E_{a\sim\mu}[\hat{R}(s,a)] = \sum_a \pi(a|s)\hat{R}(s,a) = \sum_a \mu(a|s)\frac{\pi(a|s)}{\mu(a|s)}\hat{R}(s,a) = E_{a\sim\mu}[\rho\hat{R}(s,a)]$. For a good estimation of the reward function $\hat{R}(s,a) \approx r$ and variance of $\rho(r - \hat{R}(s,a))$ becomes smaller than ρr. Thus DR estimator has lower variance in IS. Here, both ρ (as π is unknown) and $\hat{R}$ are unknown. If either one of these functions are properly estimated, then it ensures a good estimation of value function.

DR estimators is extended to the off-policy evaluation in RL setting [52] to reduce the variance of off-policy evaluation, where finite horizon MDP is considered with sample trajectory $\tau = (s_1, a_1, r_1, ...s_{T+1})$. The IS estimator provides an unbiased estimate value function of target policy π but suffers from the variance issue even when the value function is computed using a step-wise IS estimator, equation (4.6). The key step DR estimator is to use the following estimator:

$$V_{DR}(s) = \hat{V}(s) + \rho[r(s,a) + \gamma V_{DR}(s') - \hat{Q}(s,a)]. \tag{4.8}$$

Here, DR estimation of the off-policy evaluation V_π [52] is referred as V_{DR}. A key requirement in DR estimators is to use an approximation to the MDP model since the $\hat{V}$ requires the rewards from an approximation of the MDP. In other words, $\hat{R}$ is used to compute $\hat{V}$ is the model's prediction of the reward. Given the samples from past data, and an approximate model of the MDP, the goal of DR estimators is to produce a low variance regression mean equated error estimate $MSE(V_{DR}, V_\pi)$.

4.2 Doubly Robust Estimators in Off-Policy Actor-Critic

Policy gradient methods, discussed in detail in Section 2.2, learn a parameterized behavior policy $\pi_\theta(a|s)$ to maximize the discounted sum of cumulative returns along the sampled trajectories, given by $J(\theta) = E_{s\sim\rho^\pi, a\sim\pi}[\sum_{t=0}^{T}\gamma^t r(s_t, a_t)]$. Based on the policy gradient theorem [31], the policy parameters θ are updated using the gradient step (derived in equation (2.21)) $\nabla_\theta J(\theta) = E_{s\sim\rho^p i, a\sim\pi}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)]$, where, the E uses samples

under the current policy π. Often the policy gradient estimator can suffer from high variance, thus often a *control variate* is used to reduce this variance, discussed in Section 2.2.1. But deciding the control-variate is a design choice, and choosing the right control-variate is not often clear [24].

On-policy gradient estimators can be sample-inefficient in practice as they can not re-use data from past experiences. Thus an off-policy gradient estimate, based on the *deterministic policy gradient* theorem [33], is preferred. For continuous control tasks, DDPG algorithm [36] is often used due to their ease ability to learn from experience replay buffer. TD3 [37] algorithm further improves policy performance by removing overestimation bias in critic estimate. The off-policy policy gradient estimator derived in equation (2.31) is as follows:

$$\nabla_\theta J(\theta) = E_{s\sim\rho^\mu}[\nabla_\theta Q_\phi(s, \pi_\theta(s)], \tag{4.9}$$

where $E_{s\sim\rho^\mu}$ is under samples from the experience replay buffer, Q_ϕ is the critic estimate and π_θ is the deterministic policy which outputs continuous actions. The critic is be evaluated with a regression based objective (from equation (2.25)) $L(Q) = E_{a\sim\mu}[(r(s,a)+ \gamma Q(s', \pi_\theta(s')) - Q(s, \pi_\theta(s)))^2]$, where the off-policy critic evaluation is a regular one-step temporal difference (TD) based update without requiring off-policy corrections. However, training actor-critic algorithms can often be unstable to use in practice as they suffer from high variance [54] due to the policy gradient estimate is directly related to the estimate of the critic (see equation (4.9)). Thus careful fine-tuning is required for these algorithms [54].

In this work, the idea of the DR estimator is implemented in the *deterministic actor-critic* algorithms, where we propose to use a doubly robust estimator as a control variate to reduce the variance of the critic estimates. The key idea of our work is that, instead of using a control variate in the policy gradient estimate, we use a low variance and unbiased DR estimator to estimate the critic. Our key intuition is as follows: since the policy gradient estimate in actor-critic relies directly on finding the gradient of the critic in equation (4.9), the important quantity in actor-critic is to have a reliably good low variance estimate of the critic. We, therefore, aim to reduce the variance of the critic estimate. We use DDPG [36] and TD3 [37] algorithms (discussed in detail in Section 2.3.1 and Section 2.3.2) to verify policy improvement using DR estimator.

4.3 Algorithm

The off-policy critic evaluation is a regular one-step TD based update without requiring off-policy corrections (see Section 2.3.1). A critic target network $Q_{\phi'}$ is used to compute the critic estimate:

$$Q_{\phi'}(s,a) = r(s,a) + \gamma Q_{\phi'}(s', \pi_\theta(s')), \tag{4.10}$$

and the mean square error between the critic target $Q_{\phi'}$ and the current critic estimate Q_ϕ over N batch of samples are computed to optimize the critic parameter ϕ as follows:

$$\begin{aligned} L(\phi) =& E_{s\sim\rho^\mu}\big[(Q_{\phi'}(s,a) - Q_\phi(s,\pi_\theta(s)))^2\big], && (4.11)\\ =& E_{s\sim\rho^\mu}\big[(r(s,a) + \gamma Q_{\phi'}(s',\pi_\theta(s')) - Q_\phi(s,\pi_\theta(s)))^2\big], && (4.12)\\ \approx& \frac{1}{N}\sum_{i=1}^{N}\big[(r(s,a) + \gamma Q_{\phi'}(s',\pi_\theta(s')) - Q_\phi(s,\pi_\theta(s)))^2\big]. && (4.13) \end{aligned}$$

When we incorporate the DR estimator, the off-policy critic estimate in equation (4.10) can be re-written as follows:

$$Q_{DR}(s,a) = \hat{Q}(s,a) + \Big[r(s,a) + \gamma Q_{DR}(s',\pi_\theta(s')) - \hat{V}(s)\Big], \tag{4.14}$$

where $Q_{DR}(s,a)$ is the doubly robust critic estimation and $\hat{Q}$, $\hat{V}$ are computed based on an approximation of the rewards $\hat{R}$ using a *reward function approximator*. Due to deterministic policy we can ignore the IS correction for off-policy sampling (see 2.30). Thus the critic objective (4.13) using DR estimation can be written as:

$$\begin{aligned} L(\phi) =& E_{s\sim\rho^\mu}\big[(Q_{DR}(s,a) - Q_\phi(s,\pi_\theta(s)))^2\big], && (4.15)\\ =& \frac{1}{N}\sum_{i=1}^{N}\Big[\big[\hat{Q}(s,a) + r(s,a) + \gamma Q_{DR}(s',\pi_\theta(s')) - \hat{V}(s)\big] - Q_\phi(s,\pi_\theta(s)))^2\Big]. && (4.16) \end{aligned}$$

4.3.1 Reward Function Approximation

As discussed in Section 4.1.3, DR estimation requires an approximate MDP model to have an estimate of the rewards $\hat{R}$. In this work, instead of using an approximate model, we use a reward function approximator $\hat{R} = f_\psi(s, a)$ to predict the rewards. In other words, we use an approximation $\hat{R}$ of the true reward function $r(s, a)$ and learn the reward function approximator. This is a supervised learning (SL) step, where for s, a, r, s' tuples in the replay buffer, and we train an approximate reward model $\hat{R}$ to predict the true rewards r based on the following regression-based update:

$$L(\psi) = E_{s,a,r\sim Buffer}[(\hat{R}(s, a) - r(s, a))^2]. \tag{4.17}$$

The reward function approximator $f_\psi(s, a)$, therefore, predicts the reward for the batch of experiences available in the replay buffer.

4.3.2 Update rule for DR estimator

The predicted reward $\hat{R}$ is used to compute $\hat{Q}$ using one-step TD update as follows:

$$L(\hat{Q}) = E_{s,a,r,s'\sim Buffer}[(\hat{R}(s, a) + \gamma\hat{Q}(s', \pi_\theta(s')) - \hat{Q}(s, a))^2]. \tag{4.18}$$

Following equation (4.18) we further estimate $\hat{V}(s)$ and follow similar update rule. Therefore, without explicitly using an approximation of the model of the MDP, we compute Q_{DR} as in doubly robust regression estimation using our reward function approximator.

Therefore, for learning the off-policy critic estimate in actor-critic, we use the following update rule, where the Q_{DR} is estimated with a parameterized function approximator ϕ,

$$L(\phi) = E_{s,a,r,s'\sim Buffer}\Big[\hat{Q}(s, a) + \big(r(s, a) + \gamma Q_{DR}(s', \pi_\theta(s')\big) - \hat{V}(s) - Q_{DR}(s, a)\Big]. \tag{4.19}$$

Equation (4.19) can be interpreted as introducing a DR control variate [53] in the critic update, where we subtract a state-dependent baseline $\hat{V}(s)$ (as typically done to reduce variance of policy gradient estimates discussed in Section 2.2.1) and add a state-action dependent control variate $\hat{Q}(s, a)$. $\hat{Q}$ and $\hat{V}$ are estimated based on a reward function approximator $\hat{R}$. These added control variates takes in the form of adding an advantage function estimation, $\hat{A} = \hat{Q} - \hat{V}$. In our experiments, we show that adding only state-action

dependent control variate in critic estimation improves policy performance in the DDPG algorithm.

4.4 Experimental Analysis

For baseline implementations (DDPG and TD3 algorithms without DR estimator), we use the implementation from [37] and the same hyper-parameters. We run all our experiments for 10^6 time steps. We allow our policy to interact at most for 1000 time steps. Once the environment reaches to a terminal state, we evaluate the updated policy for the equal number of environment interactions for any particular episode. To get the variance in performance, we run the experiments over 5 random seeds from 0-9.

To estimate DR terms, we require additional neural networks to compute $\hat{Q}, \hat{V}, \hat{R}$. We use the same configuration for all the networks, which is a 3 layer neural network (400 unit x 400 unit x 300 units) with rectified linear unit (ReLU) activation after the first two units and no activation at the output unit.

4.4.1 Results

We evaluate the performance of off-policy actor-critic algorithm with a doubly robust critic estimator on several continuous control MuJoCo tasks [1]. We use two variants of the DR estimator in the critic: (a) we use the advantage term $\hat{Q}(s,a) - \hat{V}(s)$ and (b) add only the state-action dependent $\hat{Q}(s,a)$ in the critic estimate.

As shown in Figure 4.1, the DR estimator improves the DDPG-baseline performance over all the environments. Even though we have exciting results on the DDPG algorithm, we do not get similar improvements for the TD3 algorithm. In Figure 4.2 we see that the performance drops over all four MuJoCo tasks.

TD3 uses double-Q (discussed in Section 2.3.2) to reduce the overestimation bias of the critic. The DR estimator is proven to be unbiased in off-policy value estimation [53, 52], but it seems important to see if DR estimator along with double Q-learning remains unbiased. We also approximate a reward function which predicts the reward $\hat{R}$ to estimate the DR terms $(\hat{Q}, \hat{V})$. It is shown in [55] that noisy reward function can yield a high variance in learning. Thus, further experiments and theoretical analysis on bias-variance trade-off in critic evaluation and reward approximation may give better insight in the role of the variance of critic on the performance of the actor.

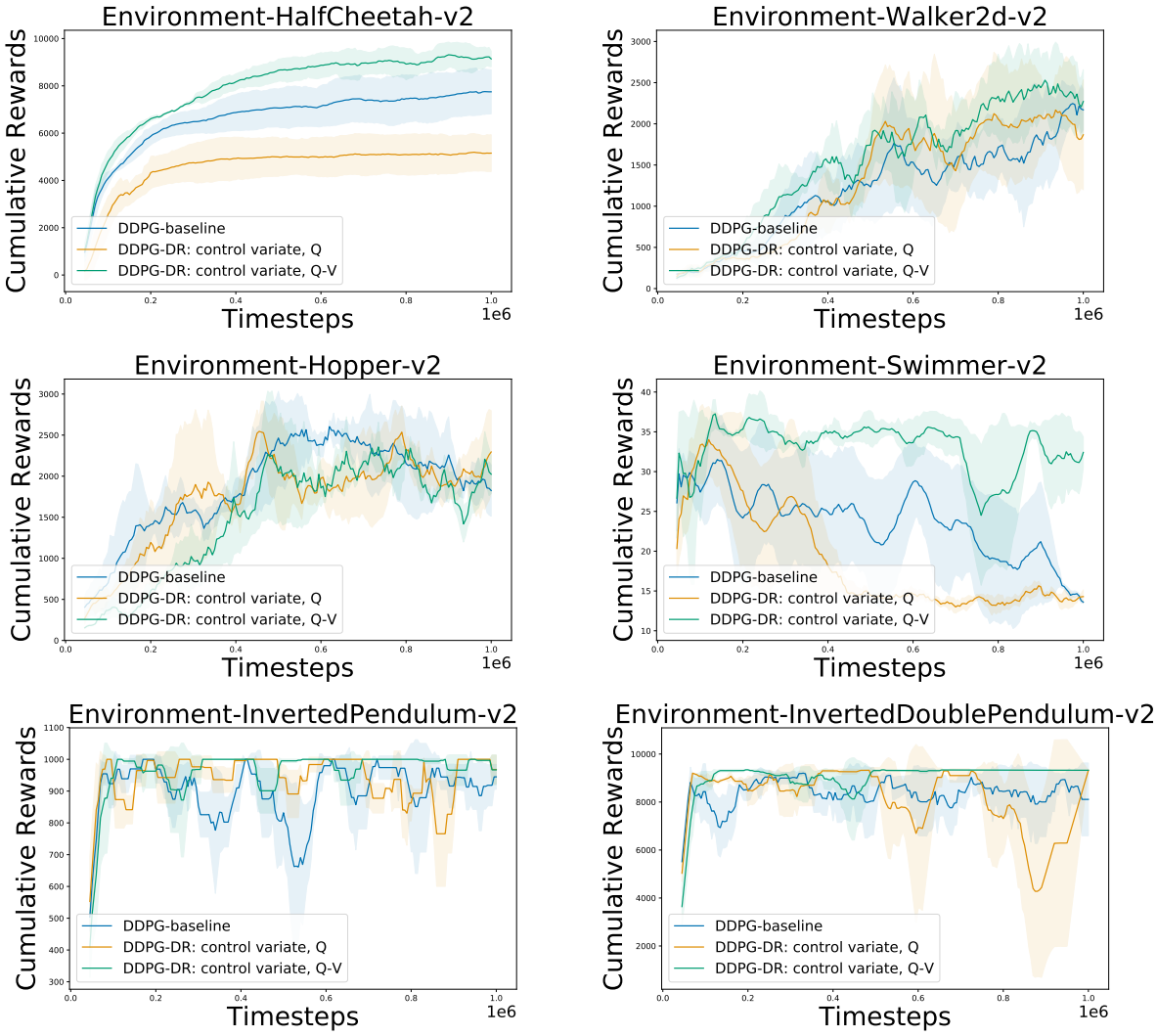


Fig. 4.1 Performance Comparison of DDPG and Variants of Doubly Robust DDPG.

4.5 Discussion

Off-policy actor-critic algorithms suffer from high variance in actor performance. Previously in policy gradient algorithms [31], it is suggested to use a baseline/control-variate in the policy gradient update to have a smoother policy performance. But deciding the baseline is a design choice, and there is no apparent justification for choosing one baseline over another [24]. On the other hand, the gradient update of the actor-network uses the critic estimation to optimize its objective function. Thus a better critic estimation is essential for

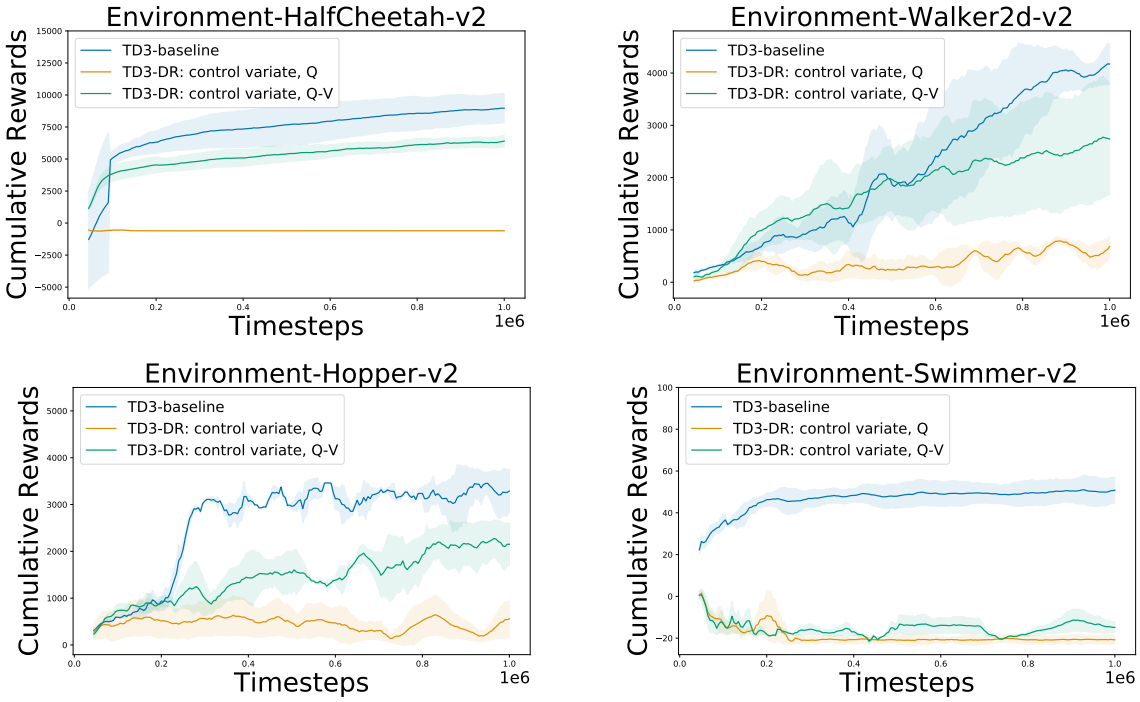


Fig. 4.2 Performance Comparison of TD3 and Variants of Doubly Robust TD3.

a good actor update. We hypothesize that a better critic estimate leads to lower variance in actor performance.

In this chapter, we propose using doubly robust (DR) estimator, a popular statistical estimation method, in the *deterministic* off-policy actor-critic algorithm, where the critic is estimated with a DR estimator to reduce variance in actor performance. Even though the DR estimator depends on an approximate model of the MDP, we propose a model-free approach by approximating a reward function $\hat{R}_\psi$ and using a deterministic policy also reduces the complexity of estimating the DR terms. The reward function approximator is learned using the batch of data in the experience replay buffer. We evaluate the performance of our algorithm on standard baselines, building on top of the deterministic actor-critic algorithms, and show that the introduction of a control variate in the critic estimate leads to marginally better performance in DDPG [36] algorithm. But for TD3 [37] algorithm the performance degrades. For both of these algorithm same control-variate and update rules are applied.

The purpose of this work is to improve actor performance in off-policy actor-critic algo-

rithm, so that we further can implement this as the generator in off-policy AIRL algorithm proposed in Chapter 3. But the current implementation of DR estimator is not proven to improve the performance of actor-critic algorithms in general. Thus we do not consider double robust critic estimation in off-policy-AIRL. Further experiments and theoretical analysis on bias-variance trade-off in critic evaluation [37] and reward approximation [55] will give a more generalized implementation of doubly robust in the actor-critic algorithm.

Chapter 5

Off-policy-AIRL in Transfer Learning

We propose *off-policy-AIRL*, an IRL algorithm in adversarial setting, in Chapter 3 which improves imitation performance of *AIRL* [17] and gives a comparable performance with state-of-the-art AIL algorithm (see Figure 3.8). The advantage of our proposed algorithm over AIL is that it can leverage the learned reward function to retrain the policy when there is a significant change in the test environment. In principle, the reward function captures the underlying objective of the desired task from expert demonstrations; thus, it is possible to use the reward function to retrain policy under changed dynamics. We test our hypothesis through *transfer learning* experiments, where we train an agent in one task through imitation (where expert demonstration are available) and transfer the learned skill to another task, where we consider certain dynamic changes. We start this Chapter by defining the *dynamic changes*. In the transfer learning experiments, we also consider the objective/goal of the agent is fixed, and the expert demonstrations are unavailable. As AIL algorithms only learn the policy, there is no way to retrain its policy for the new task if expert demonstrations are not available.

To the best of our knowledge, *AIRL* [17] is the only IRL algorithm that shows the utility of reward function in transfer learning for continuous control tasks. Thus we demonstrate our experiments using the same environments from [17]. It is important to note that the best performing policy in AIRL [17] for imitation task is trained using state-action dependent reward approximator $r_\psi(s, a)$, which fails to relearn policy in transfer learning. For state-dependent reward approximator $r_\psi(s)$, AIRL successfully completes the transfer learning task, but it comes with a cost of poor imitation performance. In this Chapter, we

perform transfer learning using the same reward approximator that works best in imitation performance.

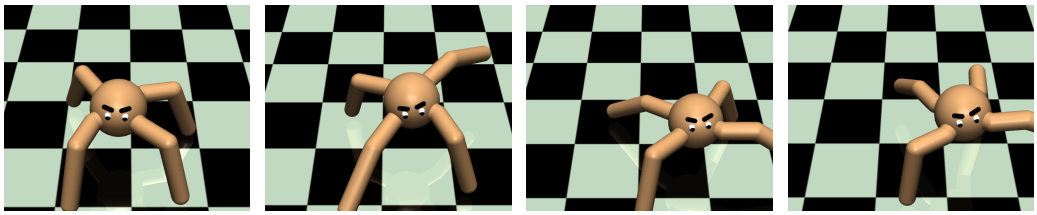
For transfer learning experiments, we consider dynamic changes in the following two criteria, where either (1) dynamics of the agent or (2) dynamics of the environment change while the goal/objective of the agent and action dimension remain the same. Through these experiments, we show the learned reward approximator r_ψ indeed can help the policy to act under dynamic changes even when expert behavior is absent. For transfer learning experiments, we use the hyper-parameter and update rules described in Chapter 3.

Criterion:1

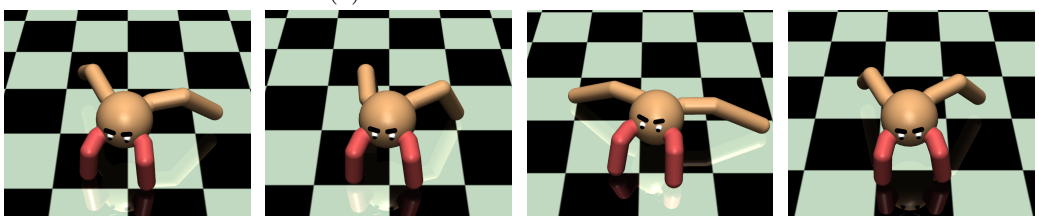
We consider a structural change in an agent by changing the dynamics of the agent. To evaluate transfer learning performance under this criterion we use *Customized Ant* [17] environment (see Figure 5.1). In both imitation and transfer task, the quadrupedal Ant requires to run forward, while the dynamics of the agent during the test session is changed by disabling and shrinking two legs. In MuJoCo setting, [17] sets the gear ratios to 1, whereas the other joints were set to 150. The gear ratio is a parameter which translates actions to torques on the joints, so the disabled joints are $\sim 150x$ more difficult to move. This significantly changes its gait. In a MDP this puts a restriction in action space A while the action space is still the same, which results in a new transition dynamics $T'(s|s,a)$. Thus optimal policy π^* is changed even though the objective and action-dimension remain the same.

Criterion:2

We consider notable variation in the trained environment such that it directly results in a different transition dynamics $T'(s|s,a)$ without putting any restriction in action-space A. The objective of *Shifting Maze* [17] (see Figure 5.2) task is to reach to a specific goal position, where during imitation learning, agent is trained to go around the wall on the left side, but during transfer task environment setup is changed and must go around the wall on the right. Thus optimal policy π^* is changed even though the objective and the dynamics of the agent remain the same.

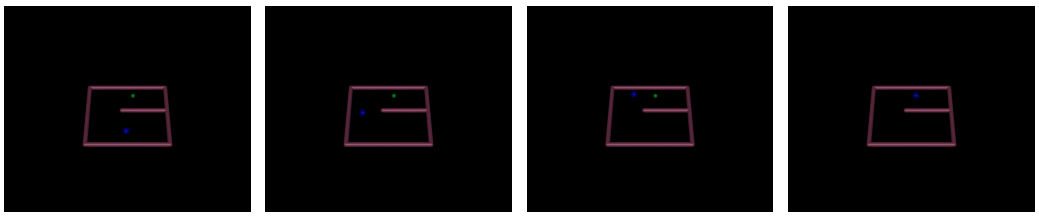


(a) Environment CustomAnt-v0

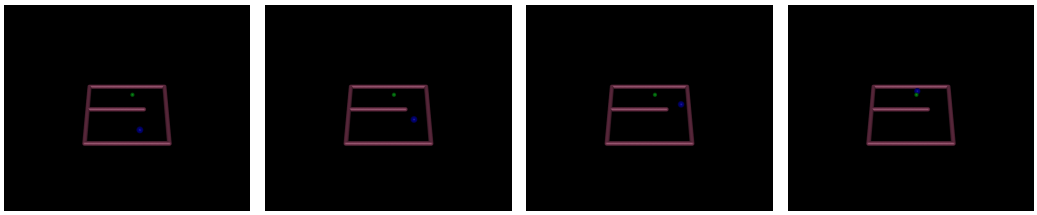


(b) Environment DisabledAnt-v0

Fig. 5.1 *Top row:* An Ant running forward using off-policy-AIRL (Generator: SAC) *Bottom row:* For DisabledAnt-v0 our algorithm fails to understand new action co-ordination required between legs to properly move forward.



(a) Environment PointMazeLeft-v0



(b) Environment PointMazeRight-v0

Fig. 5.2 For shifting maze task, the agent (blue) is required to reach at the goal (green). (a) PointMazeLeft-v0 environment, used during training, requires an agent to go around the wall on the left side. But during transfer learning, illustrated in (b), agent must go around on the right.

We train an agent using off-policy-AIRL algorithm through imitation and use the learned reward function to retrain the policy during transfer learning task. For transfer learning experiment under criterion-1 agent tries to use active legs to crawl random directions but fails to co-ordinate the legs to move forward. We conjecture that the *specific*

increment of active legs and decrement of disabled legs of quadrupedal works in favor of [17]. As discussed in [56], changing the specific structure of an agent for a specific task can be better suited for learning policy. Further experiments, on environments with more structural flexibility of the agent is required to make a concrete conclusion, which is not done in this thesis. For transfer tasks under criterion-2, our algorithm successfully completes the objective but suffers from high variance in performance.

RL algorithms often overfits the policy to one task and thus, it becomes harder to re-train during the transfer learning task. Different techniques, such as early stopping or regularization, can be helpful in this case. *AIRL* requires early stopping over imitation tasks so that the policy can be reusable in a transfer setting. An alternative solution is provided by [57] through using *variational bottleneck* technique to control the information that is fed to the discriminator using information theory, which allows more controlled update in the generator. But this technique is highly dependent on the hyper-parameter of the bottleneck itself. Thus performance can be varied vastly if not tuned properly for each task. We want to learn a policy that gives a good performance during transfer learning without being dependent on exact hyper-parameter tuning. We propose using *Multiplicative Compositional Policy* (MCP) [58] architecture to improve policy performance in the transfer learning task. MCP was introduced to combine multiple primitive skills in the RL setting, and these primitive skills are controlled using a gating function. In this work, we propose using multiple primitive networks to learn a single skill in the IRL setting. Our experiments show that using a weighted multiplicative composition of Gaussian primitives improves imitation performance and allows more flexibility in composing into new policy in transfer learning.

Furthermore, we observe more stable performance by initializing the agent at random state as it enables better exploration at the new environment. We use expert policy to initiate a random starting state so that the agent can learn about the expected states to be visited under changed dynamics. But we do not take any consideration of the expert action or expert trajectory into account as we consider expert demonstration is non-existent in transfer learning.

Under the condition that we do not have expert demonstration in transfer learning task we summarize our contributions as following:

- We show the utility of learned reward function of *off-policy-AIRL* in transfer learning

without compromising imitation performance.

- We propose using the MCP model as an alternative to using a regularizer or early stopping, which allows more flexibility in retraining policy in the transfer task and, at the same time, improves imitation performance.

- We observe that allowing the agent to explore from the expected states that are to be visited in transfer learning, reduces the variance in policy performance.

5.1 Multiplicative Compositional Policy

Multiplicative Compositional Policy [58] tries to learn different skills with different primitive policies and then reuse those learned skills by combining them to do a more sophisticated task. In hierarchical learning [59, 60, 61, 62] it is common to learn the primitives and then find a composite policy $\pi(a|s)$ by weighted sum of distribution from primitives $\pi_i(a|s)$. A gating function computes the weight w_i, which determines the probability of activating each primitive for a given state, s. Composite policy can be written as:

$$\pi(a|s) = \sum_{i=1}^{k} w_i(s)\pi_i(a|s), \quad \sum_{i=1}^{k} w_i(s) = 1, w_i(s) > 0. \tag{5.1}$$

Here k denotes the number of primitives, and this can be referred as *additive model*. Standard hierarchical learning models can sequentially select particular skills over time. In other words, it can activate a single primitive at each timestep. *MCP* [58] proposes an *multiplicative model*, where multiple primitives are activated at a given time-step. This allows an agent to learn a complex task requiring to perform more than one sub-task at a time.

MCP [58] decomposes agent's policy into primitives, trains those primitive policies on different sub-tasks, and then obtain a composite-policy by the multiplicative composition of these primitives. Here each primitive is considered as a distribution over actions, and the composite policy is a multiplicative composition of these distributions. Thus,

$$\pi(a|s, g) = \frac{1}{Z(s, g)} \prod_{i=1}^{k} \pi_i(a|s, g)^{w_i(s,g)}, \quad w_i(s, g) \geq 0. \tag{5.2}$$

In the original implementation of MCP[58], author considers policy is state s and goal g dependent. This model enables an agent to activate multiple primitives simultaneously with each primitive specializing in different behaviors that can be composed to produce a continuous spectrum of skills, whereas the standard hierarchical algorithm only activates single primitive at a time.

Each primitive $\pi_i(a|s,g) = \mathcal{N}\big(\mu_i(s,g), \Sigma_i(s,g)\big)$ is modeled as a Gaussian with mean $\mu_i(s,g)$ and diagonal covariance matrix $\Sigma_i(s,g) = diag\big(\sigma_i^1(s,g), \sigma_i^2(s,g)...\sigma_i^{|\mathcal{A}|}(s,g)\big)$, where $\sigma_i^j(s,g)$ denotes variance of j^{th} action parameter from primitive i and $|\mathcal{A}|$ represents the dimensionality of the action space. A multiplicative comparison of Gaussian primitives yields yet another Gaussian policy $\pi(a|s,g) = \mathcal{N}\big(\mu_i(s,g), \Sigma_i(s,g)\big)$. Since the primitives construct each action parameter with an independent Gaussian, the action parameters of the composite policy π will also assume the form of independent Gaussians with component-wise mean $\mu^j(s,g)$ and variance $\sigma^j(s,g)$. Component-wise mean and variance can be written as:

$$\mu^j(s,g) = \frac{1}{\sum_{l=1}^{k} \frac{w_l(s,g)}{\sigma_l^j(s,g)}} \sum_{i=1}^{k} \frac{w_i(s,g)}{\sigma_i^j(s,g)} \mu_i^j(s,g), \tag{5.3}$$

$$\sigma^j(s,g) = (\sum \frac{w_i(s,g)}{\sigma_i^j(s,g)})^{-1}. \tag{5.4}$$

5.2 MCP in off-policy-AIRL

The MCP architecture learns primitive policies π_i by training them into different sub-tasks and then try to learn more complex task leveraging the learned skills. But in transfer learning experiments, we use the MCP model to learn a single task. The underlying idea is to observe whether it can relearn the composition of different action parameters when we put them in dynamic scenarios. For example, during transfer learning experiments [17] disables two legs, and thus it is important for the agent to re-optimise the policy to walk in this transfer learning setup. Our initial objective is to see if the MCP model can decompose motor skills for a single task and re-optimize the learned policy by re-training the gating function.

In [58], the MCP model is implemented on-policy algorithms in the RL setting, and

the policy parameters are considered to be state and goal dependent. We explore this concept to decompose motor skills over a single task in an IRL setting using an off-policy algorithm. As illustrated in Figure 5.3, we modify the *actor* network of SAC by adding multiple primitive networks with the same configuration described in [58]. We refer to this new policy as *SAC-MCP*.

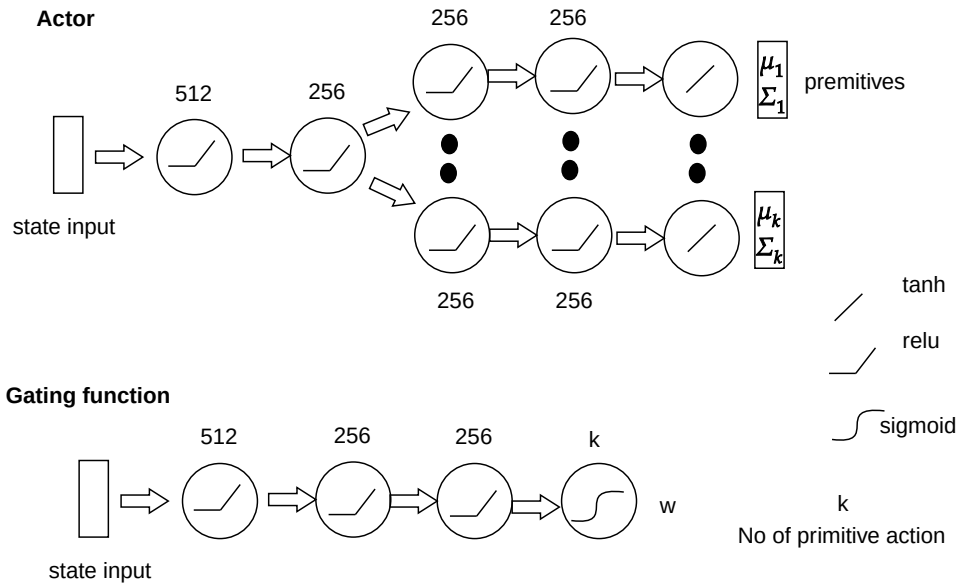


Fig. 5.3 Generator architecture for SAC-MCP.

5.3 Performance in transfer learning

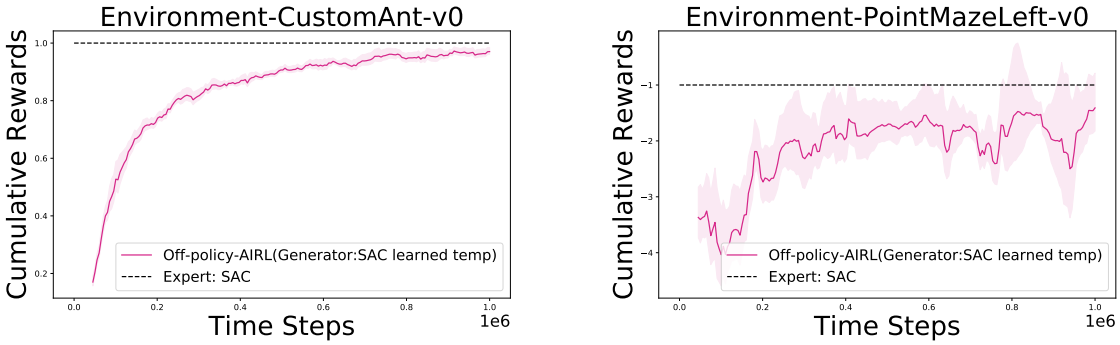
Off-policy-AIRL allows an agent to learn the policy and the reward function for continuous control tasks. In this Section, we experimentally demonstrate the advantage of our proposed off-policy-AIRL over AIL algorithms in transfer learning tasks, under criterion (1) and (2), through retraining the policy using the learned reward function.

During training, we allow the agent to take a trajectory of a maximum of 500 steps for Customized-Ant and 100 steps for Shifting-Maze. We run experiments for in total 10^6 iterations for imitation task and for 20^6 iterations during transfer learning tasks. After every 5000 steps, we evaluate the agent for 10 times and plot the graph using the average performance. Other than these, we use the same hyper-parameter and update rules described Chapter 3.

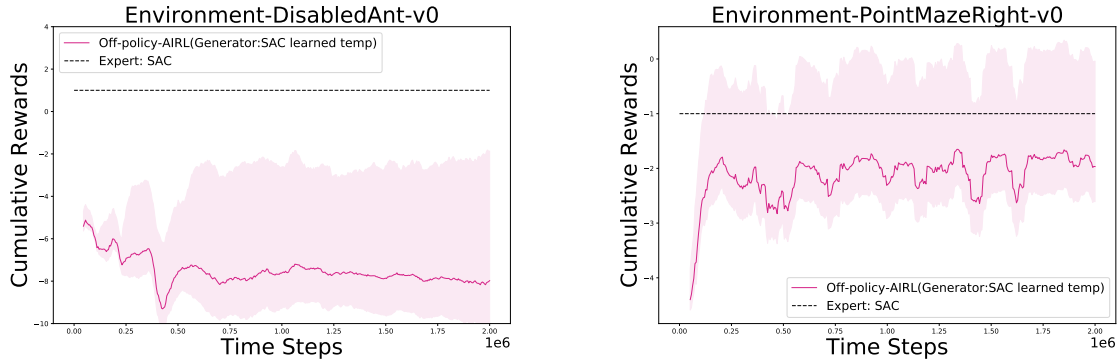
Figure 5.4a shows the imitation performance of off-policy-AIRL, where we use SAC as the generator. Off-policy-AIRL learns expert like behavior in the Customized-Ant environment but suffers from variance in shifting-maze task. We do not tune separate hyper-parameter for the individual environment, but doing so may reduce performance variance for the Shifting-maze environment.

During transfer learning under criterion-1, the agent's gait changes significantly (see Figure 5.1), and as can be seen in Figure 5.4b, off-policy-AIRL fails to relearn the policy using the learned reward function. But we do not conclude that off-policy-AIRL fails under this criterion. It is demonstrated in [56] that changing a better structure of agent's body not only is better suited for the task but also facilitates policy learning. The environment that we use in criterion-1 is introduced in [17], where the structure of the body of the quadrupedal Ant is changed [17] for transfer learning by shrinking two legs and making other two larger. But the specific increment or decrement of the legs may have favored for [17] to relearn policy when two legs are paralyzed in transfer learning. A hyper-parameter tuning of the structural change may work for our proposed algorithm as well. Experiments with more structural flexibility are required to come to a solid conclusion under this criterion but we do not conduct such experiments in this thesis.

On the other hand, for the Shifting-maze environment (see Figure 5.4b), off-policy-AIRL successfully completes the task but suffers from high variance in performance.



(a) Imitation performance of off-policy-AIRL over continuous control task



(b) Performance off-policy-AIRL during transfer learning task

Fig. 5.4 Performance of off-policy-AIRL (Generator: SAC) during transfer learning.

5.3.1 Performance comparison for using MCP

We modify the single actor-network of SAC with multiple actors ($k = \{4, 8\}$) for SAC-MCP implementation (see Figure 5.3). To validate the SAC-MCP implementation, we compare the imitation performance of off-policy AIRL algorithm for different generators, which is shown in Figure 5.5 and Figure 5.6. Using SAC-MCP as the generator in off-policy-AIRL, improves the imitation performance in multiple (HalfCheetah-v2 and Ant-v2) MuJoCo environments.

Performance of the transfer learning tasks are shown in Figure 5.7. During the transfer learning, we do not load prior gating function. We compare the performance with and without retraining the actor. Our initial hypothesis is that having the multiplicative composition of Gaussian primitives should allow us with diverse behavior by only training the gating function during the transfer task. But Figure 5.7a shows the agent is not able to

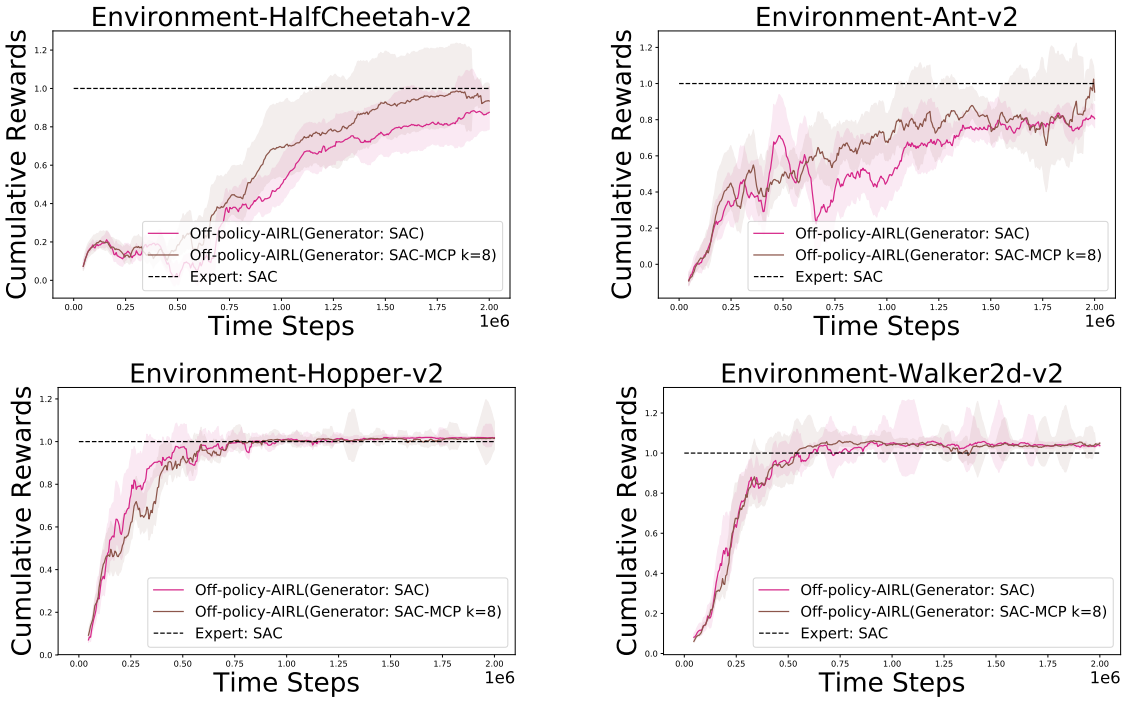


Fig. 5.5 Performance Comparison of SAC and SAC-MCP as generator in off-policy-AIRL over continuous control task.

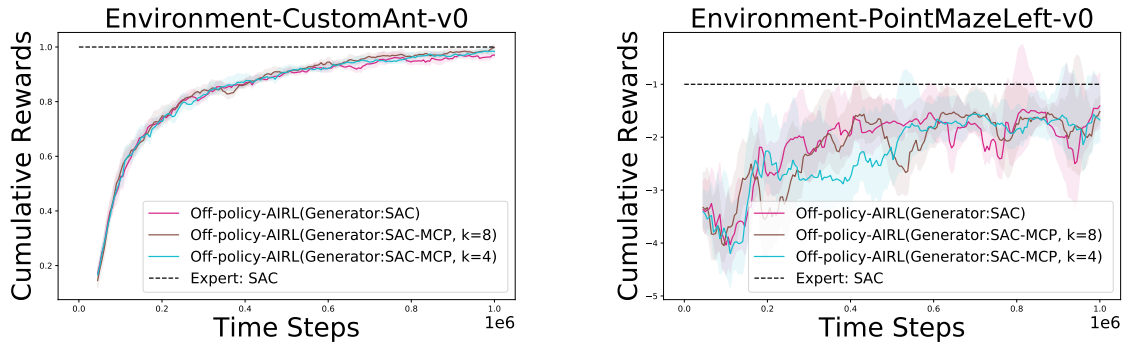
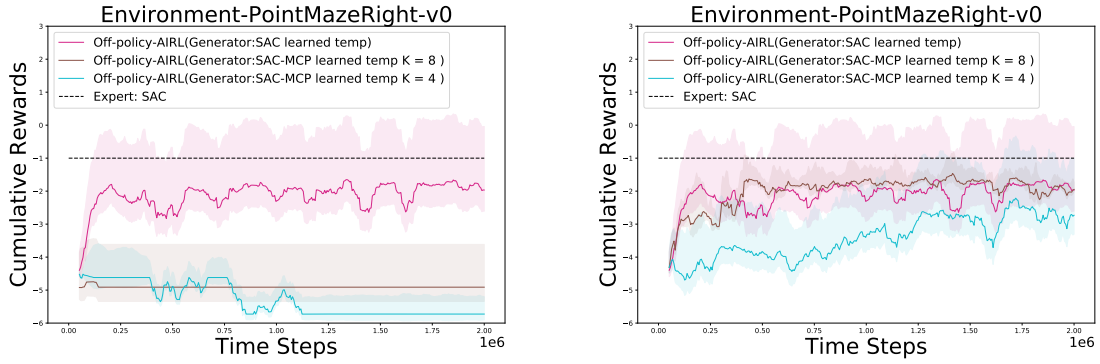


Fig. 5.6 Performance of SAC-MCP as generator in off-policy-AIRL over continuous control task.

learn the shifting-maze task without retraining the actor networks. We compare the performance of SAC-MCP for $k = \{4, 8\}$ primitives. For $K = 8$ it reduces the performance variance suffered by SAC on shifting-maze task (see Figure 5.7b).



(a) Train only gating function

(b) Train both policy and gating function

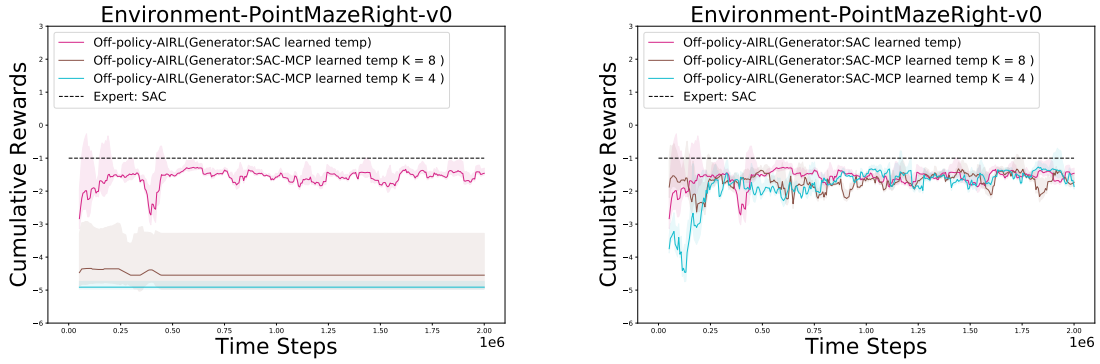
Fig. 5.7 Performance of off-policy-AIRL (Generator: SAC-MCP) during transfer learning with fixed state initialization.

5.3.2 Random state initialization

Here, we initialize the environment from a random state. As the policy is already trained over one task, it does not give much exploratory action. Thus initializing the environments at different states help to explore the new task. We use an expert agent to run for 0-100 steps and then stop at random, thus enabling the learning agent to start at random state every time the environment resets. It allows the agent to explore from a state that is expected to be visited by the expert. At the same time, as the objective/goal is fixed in the transfer task, a trained policy from prior imitation task takes better sequential decisions. We see in Figure 5.8 for both SAC and SAC-MCP, random initialization reduces performance variance for the shifting-maze environment.

5.4 Discussion

In this Chapter, we experimentally demonstrate the advantage of learning off-policy-AIRL over AIL algorithms. Off-policy-AIRL approximates a reward function along with learning expert-like policy during imitation learning. We hypothesize that the approximated reward function captures the expert's objective, and thus it can be used to retrain policy under dynamic changes where expert demonstrations are not available. In the experiments, we train the agent on one task through imitation and expect to relearn the expected policy in another task under two transfer learning setting, where either (1) the dynamics of the agent



(a) Train only gating function (b) Train both policy and gating function

Fig. 5.8 Performance of off-policy-AIRL during transfer learning with random state initialization.

or (2) the dynamics of the environment changes while in both cases the goal/objective and action dimension remain the same.

.

Furthermore, we propose using MCP model to learn single primitive to achieve more flexibility in retraining policy during transfer learning. We observe an improved performance for using SAC-MCP on both imitation performance and transfer learning under criterion 2. We also conduct experiments under random state initialization, which helps the agent to explore from a state that is expected to be visited in the new environment and reduces performance variance over prior experiments.

But unfortunately, off-policy-AIRL algorithm fails when the (criterion 1) dynamics of the agent is changed in transfer learning. The quadrupedal Ant has a very specific increment and decrement of the legs, which may have favored for [17] to re-learn policy when two legs are paralyzed in transfer learning. As specific body structure of an agent can facilitate learning policy over a task [56], experiments using environments with more structural flexibility is required to come to a conclusion under this criterion.

Chapter 6

Conclusions

Generative Adversarial Imitation Learning (GAIL) [11], discussed in Section 2.5.1, introduces the concept *Adversarial imitation learning* (AIL) where a generator (policy) is used to compute trajectory, and a binary classifier called discriminator classifies generated behavior from the expert. It is the first algorithm to provide expert-like performance on continuous control tasks in a model-free setting. *Discriminator Actor Critic* (DAC) [26] (discussed in Section 3.1) further solves sample inefficiency of imitation learning with respect to policy transitions in the environment. But these algorithms only learn policy under fixed dynamics. Thus fail to provide expected behavior when there is a change in the environment dynamics [17].

Inverse reinforcement learning (IRL) [12, 13, 5, 17, 25] addresses the importance of learning reward function and thus learning reward function approximator along with the policy. *Adversarial Inverse Reinforcement Learning* (AIRL) [17] (see Section 2.5.2) proposes an IRL algorithm in adversarial learning, which shows promising results when there is considerable variability in the environment from the demonstration setting, thus very useful in transfer learning tasks. But compared to AIL (i.e., GAIL, DAC) algorithms, AIRL does not perform well in imitation performance [26, 17]. It is also important to note the best performing reward function approximator configuration for imitation and transfer learning tasks in the AIRL algorithm are different.

Inspired from DAC [26], in Chapter 3, we propose an *Off-Policy Adversarial Inverse Reinforcement Learning* (off-policy-AIRL) algorithm that takes the best out of imitation learning algorithm and tries to understand the objective of expert by learning a reward

function. We use *Soft Actor-Critic* (SAC) as the generator, which makes our algorithm more sample efficient. In our experiment, we show that off-policy-AIRL provides comparable performance to DAC (see Figure 3.8), which is the state-of-the-art AIL algorithm on continuous control tasks.

We use an off-policy actor-critic algorithm as our generator. Similar to *policy gradient algorithms* (discussed in Section 2.2), off-policy actor-critic algorithms suffer from high variance in performance (discussed in 2.2.1) because in both algorithms, the policy gradient update depends on the value estimation (see policy gradient objective 2.21). Using different control-variates/baselines have been proposed in policy gradient update [20, 21, 22, 23], but choosing the right baseline is not obvious [24]. Thus in Chapter 4, we further look into improving the performance of the generator by reducing the variance issue. As actor performance vastly depends on critic estimation [37], we propose using a *doubly robust estimation method* (DR) to estimate critic. DR estimator is proven to be unbiased and reduces variance in off-policy evaluation [52, 53]. Our hypothesis is that using the DR estimator leads to lower variance in critic estimation, and thus a more controlled policy update results in an improved policy performance in both RL and imitation learning settings. We implement a model-free DR estimator on *deterministic* off-policy actor-critic algorithms (DDPG [36] and TD3 [37]) as the deterministic policy reduces the DR estimation complexity (discussed in Section 4.3). Even though our implementation improves the DDPG-baseline performance, we see a decline in TD3 performance. There are two important things that further study needs to investigate. Firstly, TD3 uses double-Q (discussed in Section 2.3.2) to reduce the overestimation bias of the critic. It seems important to see if the DR estimator, along with double Q-learning remains unbiased. Secondly, we also approximate a reward function, which predicts the reward $\hat{R}$ to estimate the DR terms $(\hat{Q}, \hat{V})$. It is shown in [55] that corrupt rewards can yield a high variance in learning. Thus, further experiments and theoretical analysis on bias-variance trade-off in critic evaluation and reward approximation may give a more generalized implementation of doubly robust in the actor-critic algorithm.

In Chapter 5, we show the advantage of learning off-policy-AIRL over AIL by reusing the learned reward function to train policy under certain changes in the dynamics where the expert demonstrations are not available. Thus we experiment on *transfer learning* setting, where we train an agent using off-policy-AIRL on one task and use the learned reward function to retrain on another task, which has significant dynamic changes. We also define these *dynamic change* over two criteria, where either (1) the dynamics of the agent or (2)

the dynamics of the environment changes while the goal/objective and action dimension remain the same. As AIL algorithms only learn the policy, there is no way to adapt the policy over the new task without expert demonstrations. We also show using *Multiplicative Compositional Policies* (MCP) architecture on SAC allows more flexibility in retraining policy in transfer task (Figure 5.7b) and at the same time improves imitation performance (Figure 5.5). A further performance variance reduction is achieved in the transfer task through the random initialization of the environment. As the agent is already trained over one task, it does not provide exploratory behavior anymore. Thus we initialize the agent to explore from a state that is expected to be visited in the new environment, and this reduces performance variance over prior experiments (Figure 5.8). But unfortunately, our algorithm (using both SAC and SAC-MCP as the generator) fails when the (criterion 1) dynamics of the agent is changed in transfer learning. The quadrupedal Ant environment used under criteria-1 (see Figure 5.1) is introduced in AIRL [17], which has a very specific increment and decrement of the legs, which may have favored for AIRL to re-learn policy when two legs are paralyzed in transfer learning. As specific body structure of an agent can facilitate learning policy over a task [56], experiments using environments with more structural flexibility is required to come to a conclusion under this criterion.

References

[1] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.

[2] V. L. Sheldon Benard and S. Y. Arnob, "Iclr 2019 reproducibility-challenge discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning," *ICLR 2019 REPRODUCIBILITY-CHALLENGE*.

[3] C. Finn, "Inverse reinforcement learning," *Deep RL bootcamp*.

[4] U. berkley, "Actor critic algorithm," *UC berkley: Deep Reinforcement Learning*.

[5] U. berkley, "Inverse reinforcement learning algorithm," *UC berkley: Deep Reinforcement Learning*.

[6] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[7] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[8] H. M. Le, C. Voloshin, and Y. Yue, "Batch policy learning under constraints," *arXiv preprint arXiv:1903.08738*, 2019.

[9] R. Cheng, A. Verma, G. Orosz, S. Chaudhuri, Y. Yue, and J. W. Burdick, "Control regularization for reduced variance reinforcement learning," *arXiv preprint arXiv:1905.05380*, 2019.

[10] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[11] J. Ho and S. Ermon, "Generative adversarial imitation learning," *CoRR*, vol. abs/1606.03476, 2016.

[12] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, vol. 99, pp. 278–287, 1999.

[13] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, p. 1, ACM, 2004.

[14] M. Bain and C. Sammut, "A framework for behavioural cloning.," in *Machine Intelligence 15*, pp. 103–129, 1995.

[15] 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*, pp. 627–635, 2011.

[16] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9329–9338, 2019.

[17] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adverserial inverse reinforcement learning," in *International Conference on Learning Representations*, 2018.

[18] C. Finn, P. F. Christiano, P. Abbeel, and S. Levine, "A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models," *CoRR*, vol. abs/1611.03852, 2016.

[19] R. S. Sutton, A. G. Barto, *et al.*, *Introduction to reinforcement learning*, vol. 2. MIT press Cambridge, 1998.

[20] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[21] L. Weaver and N. Tao, "The optimal reward baseline for gradient-based reinforcement learning," *arXiv preprint arXiv:1301.2315*, 2013.

[22] C. Wu, A. Rajeswaran, Y. Duan, V. Kumar, A. M. Bayen, S. Kakade, I. Mordatch, and P. Abbeel, "Variance reduction for policy gradient with action-dependent factorized baselines," *arXiv preprint arXiv:1803.07246*, 2018.

[23] H. Liu, Y. Feng, Y. Mao, D. Zhou, J. Peng, and Q. Liu, "Action-depedent control variates for policy optimization via stein's identity," *arXiv preprint arXiv:1710.11198*, 2017.

[24] G. Tucker, S. Bhupatiraju, S. Gu, R. E. Turner, Z. Ghahramani, and S. Levine, "The mirage of action-dependent baselines in reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 5022–5031, 2018.

[25] A. H. Qureshi, B. Boots, and M. C. Yip, "Adversarial imitation via variational inverse reinforcement learning," *arXiv preprint arXiv:1809.06404*, 2018.

[26] I. Kostrikov, K. K. Agrawal, D. Dwibedi, S. Levine, and J. Tompson, "Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning," in *International Conference on Learning Representations*, 2019.

[27] R. Bellman, "The theory of dynamic programming," tech. rep., Rand corp santa monica ca, 1954.

[28] B. D. Ziebart, J. A. Bagnell, and A. K. Dey, "Modeling interaction via the principle of maximum causal entropy," 2010.

[29] A. Markov, "Theory of algorithms,"

[30] 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.

[31] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[32] P. W. Glynn, "Likelihood ratio gradient estimation for stochastic systems," *Communications of the ACM*, vol. 33, no. 10, pp. 75–84, 1990.

[33] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning*, 2014.

[34] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, pp. 1008–1014, 2000.

[35] D. Precup, "Eligibility traces for off-policy policy evaluation," *Computer Science Department Faculty Publication Series*, p. 80, 2000.

[36] 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.

[37] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *CoRR*, vol. abs/1802.09477, 2018.

[38] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems 23* (J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, eds.), pp. 2613–2621, Curran Associates, Inc., 2010.

[39] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," *CoRR*, vol. abs/1812.05905, 2018.

[40] A. Y. Ng, S. J. Russell, *et al.*, "Algorithms for inverse reinforcement learning.," in *Icml*, vol. 1, p. 2, 2000.

[41] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, (San Francisco, CA, USA), pp. 278–287, Morgan Kaufmann Publishers Inc., 1999.

[42] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[43] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," *CoRR*, vol. abs/1604.06778, 2016.

[44] K.-E. Kim and H. S. Park, "Imitation learning via kernel mean embedding," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[45] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, pp. 1310–1318, 2013.

[46] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in neural information processing systems*, pp. 5767–5777, 2017.

[47] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[48] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015.

[49] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[50] S. Gu, T. P. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-prop: Sample-efficient policy gradient with an off-policy critic," *CoRR*, vol. abs/1611.02247, 2016.

[51] M. Dudík, D. Erhan, J. Langford, and L. Li, "Doubly robust policy evaluation and optimization," *CoRR*, vol. abs/1503.02834, 2015.

[52] N. Jiang and L. Li, "Doubly robust off-policy value evaluation for reinforcement learning," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 652–661, 2016.

[53] P. S. Thomas and E. Brunskill, "Data-efficient off-policy policy evaluation for reinforcement learning," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 2139–2148, 2016.

[54] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 3207–3214, 2018.

[55] J. Romoff, A. Piché, P. Henderson, V. François-Lavet, and J. Pineau, "Reward estimation for variance reduction in deep reinforcement learning," *CoRR*, vol. abs/1805.03359, 2018.

[56] D. Ha, "Reinforcement learning for improving agent design," 2018. `https://designrl.github.io`.

[57] X. B. Peng, A. Kanazawa, S. Toyer, P. Abbeel, and S. Levine, "Variational discriminator bottleneck: Improving imitation learning, inverse rl, and gans by constraining information flow," *arXiv preprint arXiv:1810.00821*, 2018.

[58] X. B. Peng, M. Chang, G. Zhang, P. Abbeel, and S. Levine, "MCP: learning composable hierarchical control with multiplicative compositional policies," *CoRR*, vol. abs/1905.09808, 2019.

[59] P. Faloutsos, M. Van de Panne, and D. Terzopoulos, "Composable controllers for physics-based character animation," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 251–260, ACM, 2001.

[60] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[61] M. I. Jordan and R. A. Jacobs, "Hierarchies of adaptive experts," in *Advances in neural information processing systems*, pp. 985–992, 1992.

[62] X. B. Peng, G. Berseth, and M. Van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 81, 2016.