Neurocomputing 388 (2020) 60–69

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Deterministic generative adversarial imitation learning☆

Guoyu Zuo[a,b,*], Kexin Chen[a,b], Jiahao Lu[a,b], Xiangsheng Huang[c]

[a] Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China
[b] Beijing Key Laboratory of Computing Intelligence and Intelligent Systems, Beijing 100124, China
[c] Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China

ARTICLE INFO

Article history:
Received 3 June 2019
Revised 5 December 2019
Accepted 6 January 2020
Available online 12 January 2020

Communicated by Dr. Li Sheng

Keywords:
Robot learning
Imitation learning
Reinforcement learning
GAN
DGAIL

ABSTRACT

This paper proposes a deterministic generative adversarial imitation learning method which allows the robot to implement the motion planning task rapidly by learning from the demonstration data without reward function. In our method, the deep deterministic policy gradient method is used as the generator for learning the action policy on the basis of discriminator, and the demonstration data is input into the generator to ensure its stability. Three experiments on the push and pick-and-place tasks are conducted in the gym robotic environment. Results show that the learning speed of our method is much faster than the stochastic generative adversarial imitation learning method, and it can effectively learn from the demonstration data in different states of the task with higher learning stability. The proposed method can complete the motion planning task without environmental reward quickly and improve the stability of the training process.

1. Introduction

With the continuing development of artificial intelligence technology, deep learning and reinforcement learning has been increasingly used in robot learning field, which allows robots acquire novel skills to accomplish various tasks, and much work has been conducted effectively in recent years. Mahler et al. [13–15] propose a deep learning method of multi-view convolutional neural network (MV-CNN) to complete the robot grasping task. This method extracts geometric features suitable for grasping from the 3D point cloud model and calculates the global similarity, and it can finally implement the prediction of the grasping positions of unknown objects and improve the stability of robot grasping. Levine et al. [11] adopt the Guided Policy Search (GPS) method to accomplish the complex robot control tasks in an end-to-end form, such as hanging clothes and opening bottle cap. To complete the robot task quickly, Hester et al. [8] and Vecerík et al. [23] combine the ideal of learning from Demonstrations (LfD) with reinforcement learning and fill demonstration data into the experience replay buffer.

The deep learning and reinforcement learning methods can effectively solve the problems of the traditional methods for the robot control tasks, such as sensitivity to physical models, poor real-time performance, and poor generalization. However, during the learning process, deep learning requires a large number of calibrated sample data, but in general a large amount of negative sample data cannot be easily obtained for the robot to learn, therefore the generalization of the method cannot be guaranteed. Reinforcement learning does not need sample data and it has good generalization ability, and so it can make up for the deficiency of deep learning. This method has some limitations yet: whether the robot can obtain the optimal policy depends on the reward function completely, which means that the reward function should not only better reflect the tasks that the robot needs to accomplish, but also should be carefully shaped to guide the gradient descent. However, it is often difficult to quantitatively evaluate the action policy taken by the robot in complex tasks, which will lead to a failure in determining the reward function.

From the human point of view, human often uses imitation ability to learn new skills, that is, human learns new skills by observing the behaviors of other individuals and reacting in the same ways. Robots are a kind of bionic systems, and they are designed and expected to have the ability of learning new skills in the way that human beings acquire skills. Imitation learning can be used to solve the problem of robot learning new tasks by using a demonstration policy. Abbeel and Ng [1] propose the apprenticeship learning method which creates the field of inverse reinforcement learning. According to the given demonstration policy, the method uses the maximum marginal method to solve the reward function, in which a hyperplane is found to divide the

☆ This document is the results of the research projects funded by the National Science Foundation of China (61873008, 61573356) and Beijing Natural Science Foundation (4182008, 4192010).

* Corresponding author at: Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China.
E-mail address: zuoguoyu@bjut.edu.cn (G. Zuo).

https://doi.org/10.1016/j.neucom.2020.01.016
0925-2312/

demonstration policy and ordinary policy into two categories and maximize the marginal between them. Xia and El Kamel [25] use neural networks to replace the artificially specified reward function base to improve the expression ability of the reward function. In order to solve the problem of ambiguous solutions in the above methods, Ziebart et al. [26] propose an inverse reinforcement learning method based on the maximum entropy model. The method uses the principle of maximum entropy to obtain the probability model that satisfies the conditional constraints, and transforms the inverse reinforcement learning problem into the optimization problem with the maximum entropy. Inverse reinforcement learning can solve the problem that the reward function can not be directly determined by using the demonstration data, but there are still some shortcomings such as large computation, slow learning speed, and artificial setting of reward function.

To complete the robot imitation learning task quickly and with high quality, Ho and Ermon [9] propose a generative adversarial imitation learning (GAIL) method, which is called stochastic generative adversarial imitation learning (SGAIL) in our paper, by combining the inverse reinforcement learning with the generative adversarial network (GAN) [6]. The method takes the deep reinforcement learning method as the generator to generate the action policy of robot, and take advantage of the discriminator to calculate the difference between the generation policy and the demonstration policy. Through adversarial training, the generation policy is increasingly close to the demonstration policy, and finally the imitation learning task is fulfilled. Stadie et al. [22] improve the GAIL method in imitating the demonstration policy from the perspective of the third person. Wang et al. [24] integrate variational auto-encoder (VAE) with GAIL to improve the robustness of GAIL. Now, the GAIL method often adopts the methods based on the stochastic policy, such as trust region policy optimization (TRPO) [19] and the proximal policy optimization (PPO) [7,20], as the generator. The method requires a large number of samples from the environment, and there is extremely a long training time to train a complex model for imitation tasks. Some works has begun to address the problem such as the slow training and large data demand of GAIL. Kostrikov et al. [10] used an preprocessing technique on the demonstrations to solve the problem of reward bias to overcome the data inefficiency of GAIL. Another natural idea is to combine the deterministic policy method with GAIL to address the problem of data inefficiency and instability in training process. However, directly implementing GAIL with deterministic policy gradient will make the learning process unstable and the algorithm hard to converge. In recent years, some studies which combine the deterministic policy with GAIL have noticed this problem and tried to improve the stability of deterministic GAIL. Barth-Maron et al. [4] propose a state screening function (SSF) to reduce the noisy policy, which makes the policy generator network avoid noisy policy updates with the states that are not typical of those appeared on the demonstrations. Schroecker et al. [18] design an off-policy learning procedure to stabilize the learning process of the off-policy generator in GAIL, which depends on the use of retained past experiences and current experience during one gradient update step. Unlike these methods, the method proposed in our paper improves the stability of the deterministic policy based GAIL by introducing the idea of LfD into the generator.

This paper proposes a deterministic generative adversarial imitation learning method (DGAIL), which adopts the deterministic policy and the adversarial training fashion of GAIL. In our method, the discriminator learns a cost function to explain the expert behavior and the generator directly learns the policy under the guidance of discriminator by combining the deterministic policy gradient algorithm with LfD. This paper is organized as follows: Section 1 introduces the background of our work and the related work, and some premilaries are described in Section 2. Section 3 details the DGAIL method and the training steps. The experiments and their result analyses are conducted in Section 4. Section 5 concludes this paper.

2. Premilaries

2.1. Policy gradient

For the stochastic policy, when the robot faces the state s, the action policy adopted by the robot obeys the distribution π. Normal distribution is the most common distribution of policy gradient method, whose mean and variance are ξ and σ^2, respectively. The action policy can be expressed by (1)

$$\pi_\theta(a|s) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(a-\xi)^2}{2\sigma^2}\right) \tag{1}$$

where the mean ξ and variance σ^2 represent the policy distribution which are the output of neural network with parameter θ; $\pi_\theta(a|s)$ is the probability density function for the robot to take action a under state s, which is subject to the normal distribution with parameter θ. It can be seen from (1) that for the stochastic policy, when the robot faces the same state s, the output of action by the robot is not exactly the same, so the gradient for the stochastic action policy can be expressed as (2)

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)] \tag{2}$$

where ρ^π is the state distribution depends on the policy π_θ and $Q^\pi(s, a)$ is the action-value function. When updating the gradient of the stochastic policy, it is necessary for the robot to sample the environment sufficiently, and then integrate the state distribution and action distribution, so that the mean is as close as possible to the expectation.

The deterministic policy [12,21] extended the stochastic police gradient framework to deterministic policies and provided that the deterministic policy gradient theorem is in fact a limiting case of the stochastic policy gradient theorem, the action policy and gradient are shown in as follows:

$$a = \mu_\theta(s) \tag{3}$$

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^\mu}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta}] \tag{4}$$

where ρ^μ is the state distribution depends on the deterministic policy μ_θ.

Compared with the stochastic policy, the deterministic policy directly outputs the action a rather than the probability distribution of the action when the robot faces the state s, so that the robot outputs exactly the same action when facing the same state of s. Therefore, when the gradient is updated for the deterministic policy, there is less need in sampling the environment and the algorithm is more efficient.

2.2. Deep deterministic policy gradients

Deep deterministic policy gradient (DDPG) [12] is proposed by DeepMind in 2016 based on deep Q-network (DQN) [16] and deterministic policy gradient (DPG) [21]. It is an off-policy model-free reinforcement learning method. Because it outputs the deterministic action rather than the probability of the selected action, it has a faster speed than the other reinforcement learning methods, and so it is widely used in the continuous control task of reinforcement learning.

Since the directly implementing Q-learning with neural networks proved to be unstable in many environments, DDPG uses the Actor-Critic (AC) framework and creates a copy of the actor and critic networks for value calculations and network parameter updates. The actor network consists of the main network

$\mu(s;\theta^{\mu})$ and the target network $\mu'(s;\theta^{\mu'})$, and it uses the soft target updates rather than directly copying the weights. The main role of the actor part is to learn the action policy in order to maximize the accumulated rewards. In the exploratory phase, in order to facilitate exploration and obtain the optimal action, the action $a_t = \mu(s_t;\theta^{\mu}) + \mathcal{N}$ is used as the output of the actor network, where $\mathcal{N}$ is the Ornstein–Uhlenbeck (OU) process. Define the performance objective $J(\mu)$ as the value function, which is obtained by averaging over the state distribution of the action policy, and update the actor parameters by the following policy gradient:

$$\nabla_{\theta} J(\mu_{\theta}) \approx \frac{1}{N}\sum \nabla_a Q(s,a;\theta^{Q})|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s;\theta^{\mu})|_{s=s_i} \tag{5}$$

Based on the idea of double Q-learning in [16], a copy of the main network is used as the target network including the target actor network $\mu'(s|g;\theta^{\mu'})$ and target critic network $Q'(s|g,a;\theta^{Q'})$, which are used to stabilize the update process of the action value function and make the network converge stably. The main role of the critic part is to evaluate the values of the actions which are taken in the current state, and thereby guide the actor network to select the optimal action. The loss function of this network is shown in as follows:

$$L = \frac{1}{N}\sum (y_i - Q(s_i,a_i;\theta^{Q}))^2 \tag{6}$$

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1};\theta^{\mu'});\theta^{Q'}) \tag{7}$$

2.3. Stochastic generative adversarial imitation learning

GAN is an unsupervised learning method proposed by Goodfellow in 2014. GAN consists of two parts: generator G and discriminator D. The G and D form a dynamic gaming process and finally reach the Nash equilibrium point. G uses noise Z as input, and the output approximates the real samples in the training set as much as possible. The input of D is the real sample or the output of G. The goal of D is to distinguish the output of G from the real sample as far as possible. In the training process, the main task of G is to make D discriminate the output of G as real sample data as far as possible. The generator parameters θ_g can be updated by the gradient as follows:

$$\nabla_{\theta_g} \frac{1}{m}\sum_{i=1}^{m} \log(1 - D(G(z^{(i)};\theta_g);\theta_d)) \tag{8}$$

where $Z = \{z^{(1)}, z^{(2)}, \ldots, z^{(m)}\}$ is noise data.

The main task of D is to distinguish the output data of G and the sample data X accurately, so as to guide G to perform optimization. The discriminator parameters θ_d can be updated by the following gradient:

$$\nabla_{\theta_d} \frac{1}{m}\sum_{i=1}^{m} [\log D(x^{(i)};\theta_d) + \log(1 - D(G(z^{(i)};\theta_g);\theta_d))] \tag{9}$$

where $X = \{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ is the sample data.

Based on the idea of GAN, Ho and Ermon [9] propose a generative adversarial imitation learning (GAIL) method by combining the inverse reinforcement learning with the generative adversarial network (GAN) [6]. The SGAIL method uses the stochastic policy reinforcement learning method as generator G_{θ} and uses a fully connected network as discriminator D_{ω}, where θ and ω are the parameters of the generator and the discriminator, respectively. In this method, the generator is to output the action policy $\pi_{\theta}(a|s)$ of the robot based on the current state s of the robot, and then the action policy is sampled to get the action a_t that the robot should perform. The discriminator is to determine the action a_t in the current state s_t as a demonstration action or an action policy by the generator, and output the similarity between the action a_t and the demonstration action. The optimization objective of the discriminator D_{ω} is to minimize

$$\hat{\mathbb{E}}_{\tau_E}[\nabla_{\omega}\log(D_{\omega}(s,a))] + \hat{\mathbb{E}}_{\tau_D}[\nabla_{\omega}\log(1 - D_{\omega}(s,a))] \tag{10}$$

where $\tau_E \sim \pi_{\theta_i}$ is the state-action pair provided by the generator, $\tau_D \sim \pi_D$ is the state-action pair provided by the demonstration data.

By updating the discriminator, it can better distinguish the demonstration action and the generation action so that better guide the generator. The optimization objective of the discriminator G_{θ} is to maximize

$$\hat{\mathbb{E}}_{\tau_E}[\nabla_{\theta}\log(D_{\omega_{i+1}}(s,a))] \tag{11}$$

By updating the generator, it outputs the action that can narrow the gap between generative action and demonstration action, in order to deceive the discriminator. Through adversarial training of generator G_{θ} and discriminator D_{ω}, the Nash equilibrium point is finally achieved, the generation action is close to the demonstration action, and the imitation learning task of the robot is completed.

3. Deterministic generative adversarial imitation learning

In this section, we describes a deterministic generative adversarial imitation learning (DGAIL) method. Different from the SGAIL method, the DGAIL method integrates the deep deterministic policy gradient (DDPG) [12] method into the GAIL method. The purpose of this method is to accelerate the GAIL learning process by using deterministic policy gradients, and learn a reward function to evaluate the policy adopted demonstration data. Fig. 1 shows the learning structure of our DGAIL method.

In this framework, the demonstration experience replay buffer D and the agent exploration experience replay buffer E are used to store the expert demonstration data and the agent exploration data, respectively. During training, the experiences are obtained from the above two replay buffers. Here, we use the DDPG method as the generator to output the action policy. The discriminator distinguishes whether the input action policy is demonstration policy or generation policy clearly. In the generator, the actor network $\mu(s|g;\theta^{\mu})$ is used to generate above action policy, and the critic network $Q(s|g,a;\theta^{Q})$ is used to assess the value of the output action more accurately, so that the action policy generated by the actor network converges to the demonstration policy.

In the discriminator $D(s|g,a;\theta^{D})$, the differences are mainly compared between the actions taken by the expert and those taken by the robot at the current time. Since our experimental environment can not provide the agent with real-time reward signals to learn effectively, the discriminator is used to learn a reward function that evaluates whether a state-action pair originates from the demonstration data or the generator. This learned reward function can be used to guide the generator to converge to the demonstration policy. The optimization objective of the discriminator is to minimize

$$J_{\theta}(D_{\theta}) = \hat{\mathbb{E}}_{\tau_E}[\log(D(s_i|g_i,a_i))] + \hat{\mathbb{E}}_{\tau_D}[\log(1 - D(s_i|g_i,a_i))] \tag{12}$$

where τ_E is the state-action pair provided by the generator, τ_D is the state-action pair provided by the demonstration data.

In the generator, the target network calculates the target value to make the network converge stably. In the case where the main network and the target network are unknown, the network parameters can be updated by using the TD method in the form of (6) and (7). Based on the idea of learning from demonstrations, the demonstration data stored in the replay buffer is used to improve the learning speed. In particular, in order to ensure the exploratory ability of robot, set $a_t = \mu(s_t|g_t;\theta^{\mu}) + \mathcal{N}_t$, where $\mathcal{N}_t$ is the Gaussian noise. But for the generator, simply increasing the output

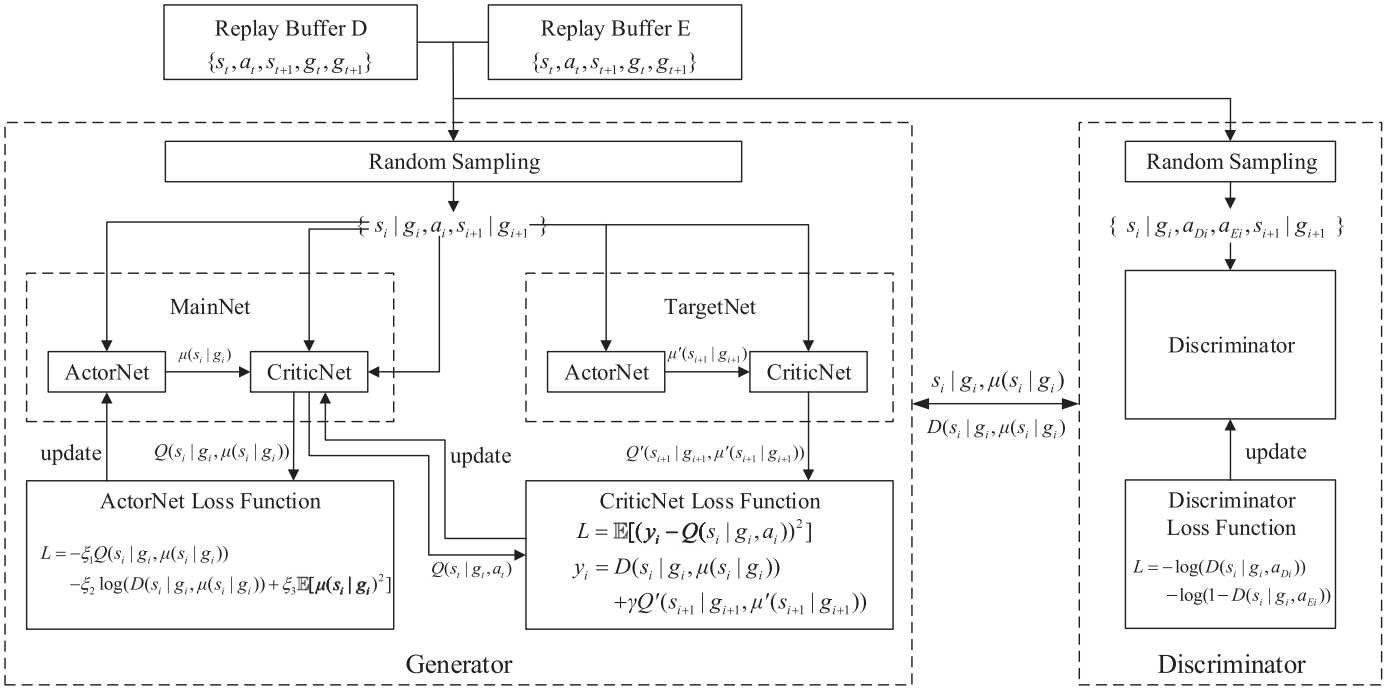

Fig. 1. Structure diagram of DGAIL.

$D(s_t|g_t, a_t; \theta^D)$ of the discriminator will cause the robot to pay more attention to the action $\mu(s_t|g_t; \theta^\mu)$ in current state $s_t|g_t$ and ignore the influence of current actions on the future. As a result, it has poor generalization ability. In order to alleviate this problem, DGAIL uses the critic network $Q(s|g, a; \theta^Q)$ and the discriminative network $D(s|g, a; \theta^D)$ to guide the generator at the same time, so that the output of the generator has better performance.

For critic network, according to the current state $s_t|g_t$ of the robot and the action $\mu(s_t|g_t; \theta^\mu)$ taken by the robot, it predicts the action value $Q(s_t|g_t, a_t; \theta^Q)$, that is, the action-value function. At the same time, DGAIL inputs the demonstration data into the critic network, and the critic network can use the demonstration data for training, and improve the stability of the DGAIL method. The optimization objective of the critic network is to minimize

$$L(Q|_{s_i|g_i,a_i}) = \hat{\mathbb{E}}_{\tau_E,\tau_D}[y_i - Q(s_i|g_i, a_i; \theta^Q)]^2 \tag{13}$$

$$y_i = r_i + \gamma Q'(s_{i+1}|g_{i+1}, \mu'(s_{i+1}|g_{i+1}; \theta^{\mu'}); \theta^{Q'}) \tag{14}$$

$$r_i = D(s_i|g_i, \mu(s_i|g_i; \theta^\mu); \theta^D) \tag{15}$$

The actor network is updated by maximizing the action value function that the critic network outputs. However, for the main critic network, we use the target network to calculate the target value to prevent the divergence of action value, but the soft target updates method will reduce the optimization speed of the main network. To solve this problem, based on idea of imitation learning, we add the $-\log(D)$ alternative [3] of the exploration state-action pairs to the loss function of the actor network, which is called imitation learning loss (IM Loss) in this paper. By minimizing the IM Loss, the output policy of actor network will be closer to the demonstration data. To reduce the fluctuation of action, we add the regularization item of the output action to the loss function, which is called action loss in our method, and use a set of hyperparameters $\{\zeta_1, \zeta_2, \zeta_3\}$ to balance the effect of the regularization item. Specifically, the discriminative network predicts the reward function $D(s_t|g_t, a_t; \theta^D)$, the critic network outputs the action-value function $Q(s_t|g_t, a_t; \theta^Q)$, and the optimization objective of the actor network is to minimize

$$\begin{aligned} L(\mu|_{s_i|g_i}) = &-\zeta_1 \hat{\mathbb{E}}_{\tau_E}[Q(s_i|g_i, \mu(s_i|g_i; \theta^\mu); \theta^Q)] \\ &-\zeta_2 \hat{\mathbb{E}}_{\tau_E}[\log(D(s_i|g_i, \mu(s_i|g_i; \theta^\mu); \theta^D))] \\ &+\zeta_3 \hat{\mathbb{E}}_{\tau_E}[\mu(s_i|g_i; \theta^\mu) - \bar{\mu}_{\tau_E}]^2 \end{aligned} \tag{16}$$

where $\bar{\mu}_{\tau_E}$ is the mean of actions in the state-action pair τ_E, the first item is the action value loss, the second one is the IM loss and the third one is the action loss.

Through optimization, the actions output by the actor network are expected to (1) increase the action-value function that the critic network outputs, and output more valuable action policy; (2) reduce the difference between the distribution of action $\mu(s_t|g_t; \theta^\mu)$ and the distribution of demonstration data under the current state $s_t|g_t$; (3) reduce the fluctuation of the output action of the actor network to make the output action more smooth. Algorithm 1 details the training steps for the DGAIL method.

4. Experiments

This section mainly uses the robotics environment [17] in gym [5] to verify the effect of the DGAIL method. The experimental environment and the tasks to be completed by the agent and the structure and related hyperparameters of the DGAIL method are elaborated in the beginning part of this section. Then the push and pick-and-place tasks are conducted in the experiment to analyze the performance and the key factors of the method proposed in Section 3 through the following experiments.

- Experiment I: The effectiveness of DGAIL.
- Experiment II: The imitation effect of DGAIL.
- Experiment III: The stability of DGAIL.

4.1. Experimental environment

The Fetch robot, which is in robotics environment, using the MuJoCo physics engine has a 7-DOF arm and a two-finger gripper. Fig. 2 illustrates the learning of the push and pick-and-place

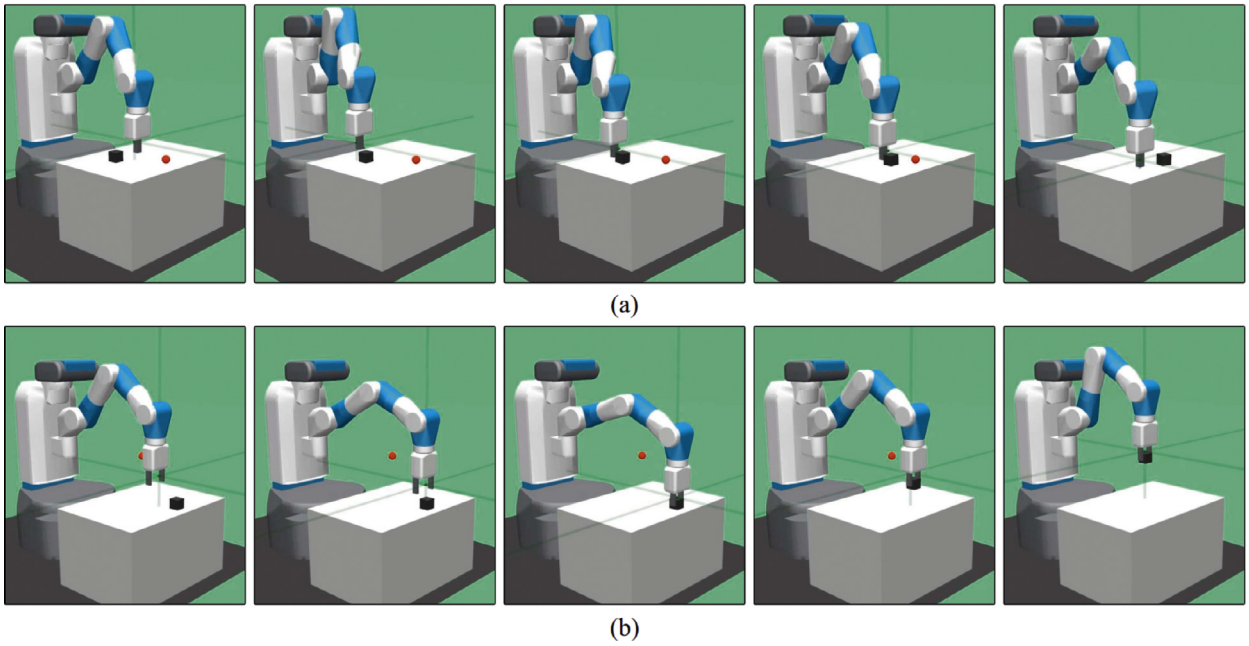

Fig. 2. Experimental tasks: push (a), pick-and-place (b).

Algorithm 1: Deterministic generative adversarial imitation learning method.

Input: Demonstration data $\tau_D \sim \pi_D$
Randomly initialize actor network $\mu(s|g;\theta^{\mu})$, critic network $Q(s|g,a;\theta^{Q})$, and discriminative network $D(s|g,a;\theta^{D})$, with weights θ^{μ}, θ^{Q} and θ^{D}
Initialize target actor network μ' and target critic network Q', with weights $\theta^{\mu'} \leftarrow \theta^{\mu}$ and $\theta^{Q'} \leftarrow \theta^{Q}$
Initialize replay buffer R_E and R_D
Store demonstration data τ_D in R_D
for *episode* $= 1, \ldots, M$ **do**
  for $t = 1, \ldots, T$ **do**
    Select action $a_t = \mu(s_t|g_t;\theta^{\mu}) + \mathcal{N}_t$
    Execute a_t and observe new state s_{t+1}, g_{t+1}
    Store transition $(s_t, g_t, a_t, s_{t+1}, g_{t+1})$ in R_E
  end
  Sample a random minibatch of *BATCH_SIZE* transitions $(s_i, g_i, a_i, s_{i+1}, g_{i+1})$ from R_E and R_D
  Calculate the reward r_i using (15)
  for *learning_num* $= 1, \ldots, N$ **do**
    for *learning_num_d* $= 0, \ldots, N_d$ **do**
      Update the discriminative network by (12)
    end
    Update the critic network by (13),(14),(15)
    Update the actor network by (16)
  end
  Update the target network :
  $\theta^{Q'} \leftarrow \delta\theta^{Q} + (1-\delta)\theta^{Q'}$
  $\theta^{\mu'} \leftarrow \delta\theta^{\mu} + (1-\delta)\theta^{\mu'}$
end

tasks. For our method, the demonstration trajectories can be collected in different ways, including but not limited to human teachers, demonstration trajectories in the simulation environment, and the agents that have completed RL-training. In our experiments, we trained an agent in advance by using traditional DDPG as an expert to provide the demonstration trajectories in the simulation environment, and record the state-action pairs through the API interface of the MuJoCo physics engine.

Push task: The fixture of robot is locked and the robot needs to utilize it to push the block to the target position. In this task, the success rate of demonstration data is 99.555%.

Pick-and-Place task: The fixture of robot is unlocked, and the robot needs to clamp the block and place the block to the target position. In this task, the success rate of the demonstration data is 98.495%.

State: The state $s_t \in \mathbb{R}^{25}$ mainly includes the angles and velocities of all robot joints as well as positions, rotations and velocities (linear and angular) of all objects.

Goal: The goal $g_t \in \mathbb{R}^3$ is the target position that the robot operates on or moves to, and the position is represented by three-dimensional Cartesian coordinates.

Action: The action $a_t \in \mathbb{R}^4$ includes four types of control information. The first three control the three-axis speed and direction of the end of the robot, respectively, and the last controls the linear velocity and direction of the two-finger gripper.

Reward: In DGAIL, the robot does not use the reward function of the environment, but uses (15) to calculate the reward function. However, in order to better verify the effect of DGAIL, we uses the reward function of the environment to evaluate this method, which is shown as follows:

$$r_t = \begin{cases} -1 & d > 5cm \\ 0 & d \le 5cm \end{cases}$$

where d represents the distance between the current position and the target position of the object.

All the experiments in this paper are carried out on the following configurations: CPU: AMD Threadripper 1950x; GPU: Nvidia GTX 1080Ti.

4.2. Training details

Fig. 3 shows the main network used in the DGAIL method. In order to complete the robot imitation learning task, in this

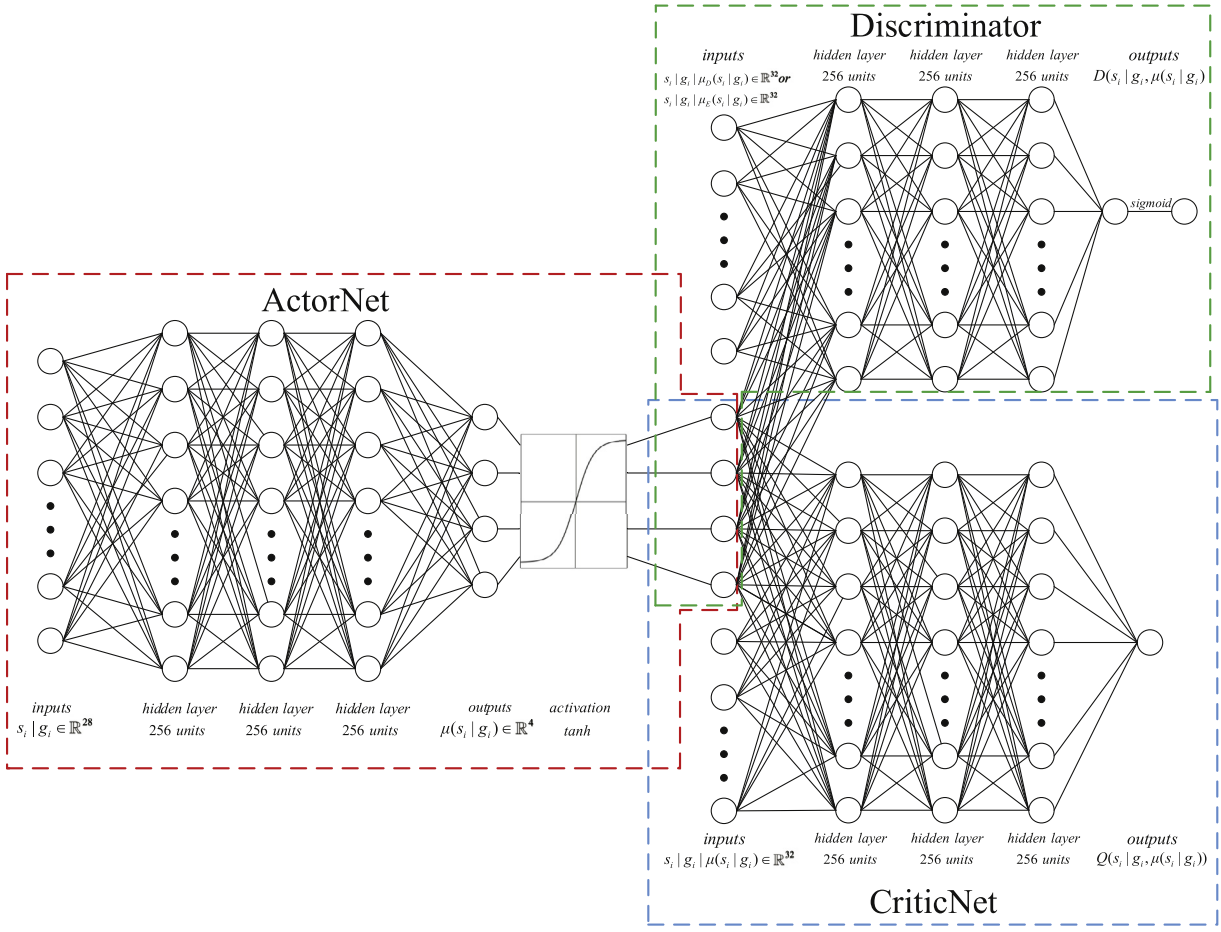

Fig. 3. The network structure of DGAIL.

method, the actor network, the critic network, and the discriminative network all use low-dimensional data as the network input. The actor network uses $s_t|g_t \in \mathbb{R}^{28}$ as its input in order to output the action $\mu(s_t|g_t)$ in the current state $s_t|g_t$. The critic network uses $s_t|g_t|\mu(s_t|g_t) \in \mathbb{R}^{32}$ as its input to output the value $Q(s_t|g_t, \mu(s_t|g_t))$ of action $\mu(s_t|g_t)$ in the current state $s_t|g_t$. The discriminative network also uses $s_t|g_t|\mu(s_t|g_t) \in \mathbb{R}^{32}$ as its input, and outputs the difference $D(s_t|g_t, \mu(s_t|g_t))$ between the action $\mu(s_t|g_t)$ and the demonstration action in the current state $s_t|g_t$. The actor network, the critic network and the discriminative network all adopt the fully-connected neural network structure which consists of an input layer, three hidden layers and an output layer. Rectified Linear Unit (*ReLU*) is used as the activation functions between the layers. Each of the three hidden layers contain 256 neurons, the output layer of the actor network contains 4 neurons, and the output layers of the critic network and the discriminative network contain 1 neuron, respectively. The actor network processes the output using the activation function *tanh* due to the action $\mu(s_t|g_t) \in [-1, 1]$ taken by the agent. The discriminative network uses the *sigmoid* function to limit the output $D(s_t|g_t, \mu(s_t|g_t))$ to [0,1]. The output layer of the critic network does not use any activation functions.

In terms of the selection of hyperparameters, DGAIL uses the Adam optimizer to optimize the neural network, where $\beta_1 = 0.9$, $\beta_2 = 0.99$. The learning rate of the actor network α_a and the critic network α_c is 0.001. The learning rate of the discriminative network α_d is 0.002. The mini-batch is 256; The discount factor γ is 0.98. The update parameters $\zeta_1, \zeta_2, \zeta_3$ of the actor network in (16) are 1, 20 and 1, respectively.

4.3. Experimental analyses

4.3.1. The effectiveness of DGAIL

In order to verify that DGAIL can effectively use the demonstration data to learn the reward function and guide the generator to complete the task, we use DDPG as the baseline experiment, and at the same time, we compare our DGAIL method with Deep Deterministic Policy Gradient from Demonstrations (DDPGfD) [23] and SGAIL, respectively. In experiment, DDPG uses the HER mechanism [2] to replay the experience data, and the reward function uses the sparse reward of the environment. DDPGfD uses the HER mechanism to replay the experience data and the demonstration data respectively. The ratio of demonstration data to exploration data is 1:1, and the reward function uses the sparse reward of the environment. PPO is used as the generator for SGAIL, and the artificially set reward function is not used. DGAIL uses random sampling to replay the exploration data and the demonstration data, respectively, the ratio of demonstration data to exploration data is 1:1, and the artificially set reward function is not used. It should be noticed that, for the model-agnostic reinforcement learning task, the methods such as DDPG and DDPGfD use the Monte Carlo method to sample the reward to estimate the state value. Simultaneously, DDPGfD combines the idea of LfD with a host of other techniques to improve the training speed and stability of the algorithm. However, for the GAIL-based methods, the agent needs to face the model-agnostic reward-agnostic task. The information that can be accessed by agent is only the state-action pairs from demonstration and exploration data. The discriminator needs to

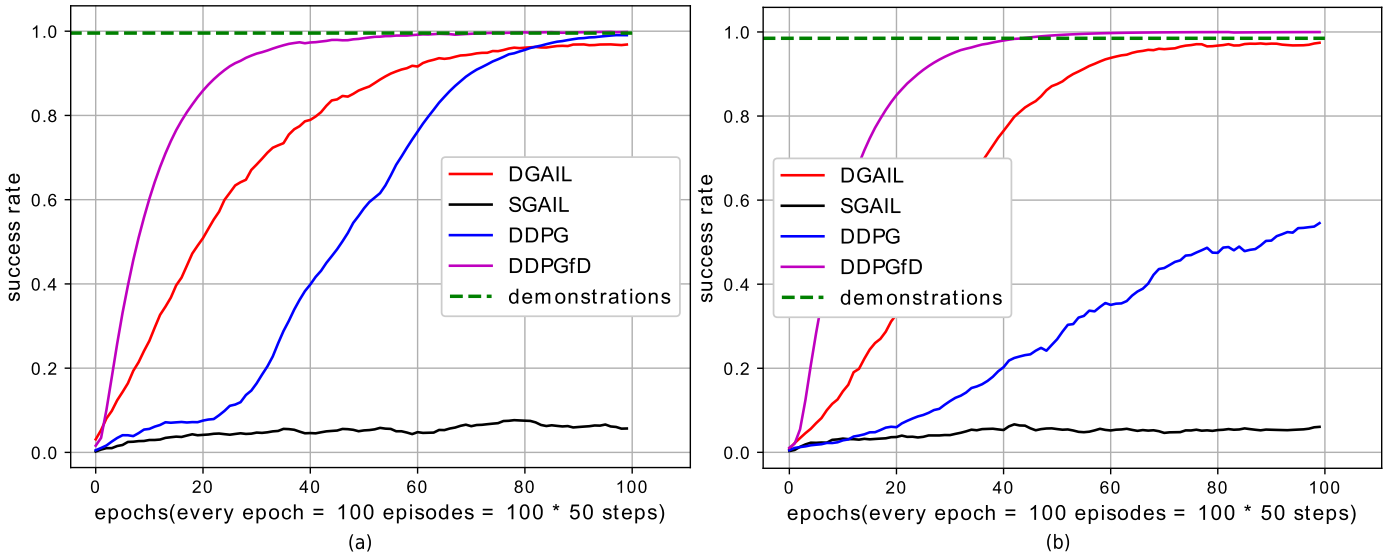

Fig. 4. Learning curves of push and pick-and-place tasks.

be first updated to determine whether the current policy is the expert policy, and then the state value function of the generator is further updated by using the output of discriminator. In this paper, we only show the training process of DDPGfD as a reference for the learning methods using both environment reward and demonstration data, but do not conduct further comparative analyses on DDPGfD with our method and the other GAIL-based methods. In the GAIL-related studies such as [4] and [18], DDPGfD has not been mentioned yet, except for the other GAIL-based methods.

Fig. 4(a) and (b) shows the learning curves of the push and pick-and-place tasks, respectively. There are four curves of different colors: purple, blue, red and black, which represent the results of four methods: DDPGfD, DDPG, DGAIL and SGAIL, respectively. The green dashed line represents the success rate of demonstration data. The horizontal and vertical coordinates in the figures are the number of epochs and the success rate of robot in each epoch. In experiment, each epoch includes 100 episodes, each episode is 50 steps. In particular, in order to make the success rate curve smoother, the experiments in this paper has carried out exponential weighted average processing on the experimental data:

$$\tilde{S} = \beta\tilde{S} + (1-\beta)S \tag{17}$$

where, $\tilde{S}$ represents the exponential weighted average success rate, S represents the success rate of this epoch, β is the weight coefficient. Here, β is 0.9.

In the push task, the success rate of demonstration data is 99.555%, and the results show that DDPGfD achieves 95% success rate in the 31st epoch, DDPG achieves 95% success rate in the 80th epoch, DGAIL achieves 95% success rate in the 80th epoch, and the SGAIL method achieves 5% success rate in the 100th epoch. In the pick-and-place task, the success rate of demonstration data is 98.495%, DDPGfD achieves 95% success rate in the 36th epoch, DDPG only achieves 56% success rate in 100th epoch, DGAIL achieves 95% success rate in the 65th epoch, and SGAIL achieves 6% success rate in the 100th epoch.

Through this experiment, we can see that the DGAIL method can effectively learn the reward function from the demonstration data and guide the generator to complete the learning tasks. Through the training of DGAIL, the final success rate of the agent is only about 2% different from the demonstrations. Meanwhile, DGAIL performs better in learning speed. It is obvious from the push task and the pick-and-place task that the learning speed of DGAIL, which does not use the environmental rewards, is basically between DDPGfD and DDPG, but it is much faster than the SGAIL method. Moreover, compared with DDPG, the learning speed of DGAIL is less affected by the difficulty of the task. Both in the push task and pick-and-place task, the learning effect of DGAIL differs slightly, but for DDPG, the change of task difficulty will lead to great difference in learning effect.

4.3.2. The imitation effect of DGAIL

In order to prove that the actor loss of our method can effect improve the learning effectiveness of the agent, this experiment compares the learning results of DGAIL with different actor loss parameters. Tables 1 and 2 show the learning results of DGAIL after 100 training epochs in the push and pick-and-place tasks, which are represented by the average values of 500 groups of tasks. We use the mean square error (MSE) between the demonstration trajectory and the action trajectory of DGAIL to represent the imitation effect of the agent, and the average time of completing target task is used to express the completion effect of target task. The trajectory of the robot is determined by the position of the end of the manipulator, which is represented by a three-dimensional vector, so the MSE distance at each time-step is calculated as the mean square error of two three-dimensional vectors. The average time of completing target task is the average number of steps required for the robot to complete 500 groups tasks. Fig. 5 shows a comparison between the output trajectories of the robot and the expert trajectories during the pick-and-place task, in which the blue curve is the trajectory of the expert, the green curve is the trajectory of the robot, and the red pentagram represents the target location where the object is to arrive.

In experiment, the IM loss can effectively accelerate the learning speed of the agent and improve the learning effect of the DGAIL method. However, when the weight hyperparameter ζ_2 increases from 20 to 50, the agent will more focuses on the behavioral cloning of demonstration data rather than the exploration of RL, and the completion time will increases. For the parameter ζ_3, the MSE of our method with action loss is smaller than that without action loss. Therefore, the action loss can effectively reduce the fluctuation of the output policy. However, we also can see that there is no obvious trend of increasing or decreasing. So, the results are relatively insensitive to the hyperparameter settings.

The results show that the discriminator can learn the demonstration action and predict the reward function to guide the

Table 1
The learning result of DGAIL with different hyperparameters in the push task.

ζ_2 of IM loss	$\zeta_2 = 50$	$\zeta_2 = 20$	$\zeta_2 = 10$	$\zeta_2 = 5$	$\zeta_2 = 1$	$\zeta_2 = 0.5$	$\zeta_2 = 0$
MSE	3.656	2.101	3.652	4.391	2.980	3.515	8.376
Completion time (steps)	21.371	18.581	20.867	22.463	19.152	21.839	47.756

ζ_3 of action loss	$\zeta_3 = 10$	$\zeta_3 = 5$	$\zeta_3 = 1$	$\zeta_3 = 0.5$	$\zeta_3 = 0$
MSE	3.405	4.221	2.048	3.260	3.331
Completion time (steps)	19.531	29.202	17.960	18.359	18.551

Table 2
The learning result of DGAIL with different hyperparameters in the pick-and-place task.

ζ_2 of IM loss	$\zeta_2 = 50$	$\zeta_2 = 20$	$\zeta_2 = 10$	$\zeta_2 = 5$	$\zeta_2 = 1$	$\zeta_2 = 0.5$	$\zeta_2 = 0$
MSE	2.833	1.741	1.956	2.054	2.428	3.563	8.692
Completion time (steps)	16.934	13.362	14.365	15.440	15.328	15.591	47.440

ζ_3 of action loss	$\zeta_3 = 10$	$\zeta_3 = 5$	$\zeta_3 = 1$	$\zeta_3 = 0.5$	$\zeta_3 = 0$
MSE	1.802	2.214	1.614	2.301	2.823
Completion time (steps)	14.229	14.412	13.081	15.179	17.069

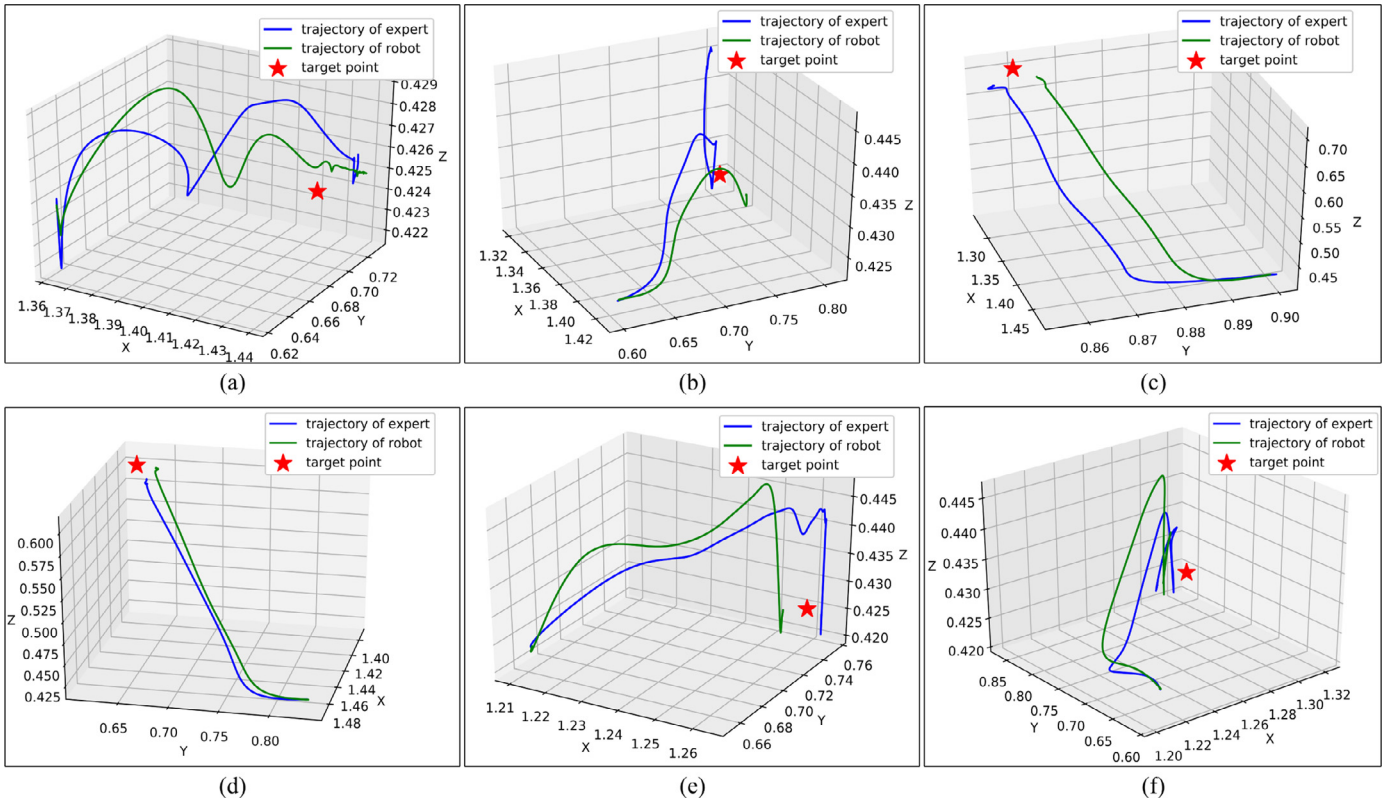

Fig. 5. Contrast diagram of robot trajectories and expert trajectories.

generator, so the output of the robot is close to the demonstration action. Moreover, because of the action loss of actor network, the motion smoothness of the robot is better than the demonstration action as shown in (b) and (e) of Fig. 5.

4.3.3. The learning result of DGAIL with different random seed

Compared with the on-policy reinforcement learning methods such as PPO and TRPO, DDPG is prone to unstable in the training process, and the combination of DDPG and GAN will be more unstable. To alleviate the instability of training process, in DGAIL, the demonstration data are input into discriminator to obtain the reward function, and they are also input into the experience replay buffer of the generator, so that DGAIL can learn better from the demonstration data rather than simply from the exploration data. In order to verify the stability of DGAIL, we compare it with the traditional DDPG method and the DDPG-GAIL with different random seeds. In this experiment, DDPG-GAIL and DGAIL use random sampling to replay the exploration data and the demonstration data, the ratio between them is 1:1, and the artificially set reward function is not used.

Figs. 6 and 7 show the learning results of the push and pick-and-place tasks, which represent the learning effect of the robot with different methods and random seeds. The black, blue and red curves show the learning effect of DGAIL, DDPG-GAIL and DDPG, respectively. The green dashed line represents the success rate of demonstration data.

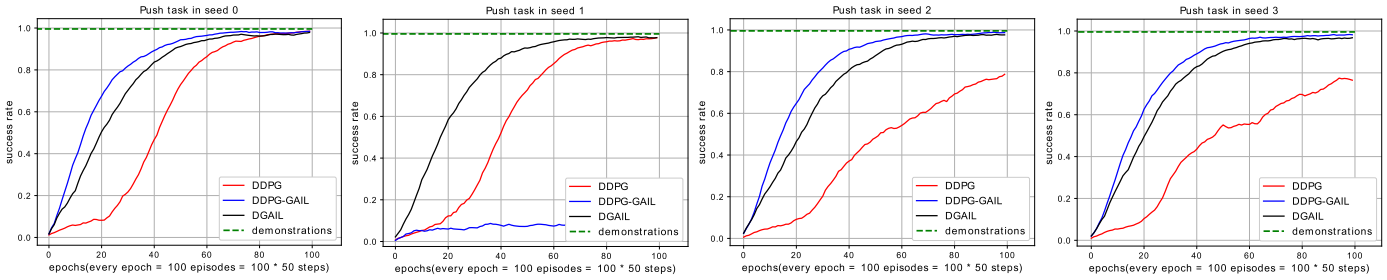

Fig. 6. Learning curves of the push task with different random seeds.

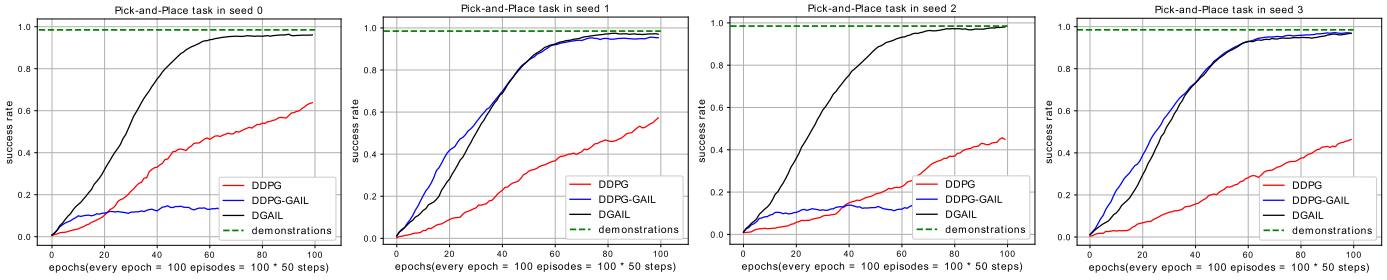

Fig. 7. Learning curves of the pick-and-place task with different random seeds.

The success rate of demonstration data in push task is 99.56% with 98.50% in pick-and-place task. For the DDPG-GAIL method, under the guidance of discriminator, the agent can complete the push task three times and complate the pick-and-place two times with four different random seeds. The agent in the DDPG method can complete both the two tasks with different random seeds, but the curves show that the learning speed of DDPG is much lower than our DGAIL method. For both two tasks, DGAIL completes the tasks under the guidance of the discriminator and ultimately achieves a success rate about 96%. All the result curves of DGAIL method can almost ignore the impact of different random seeds.

We can see that, in the face of different states of the tasks, DGAIL can effectively train from the demonstration data and guide the generator to complete the task. Compared with DDPG-GAIL, DGAIL has higher stability. Furthermore, DGAIL can improve the utilization of demonstration data and makes the demonstration data play a greater role in robot learning. DGAIL can better integrate GAN with the deterministic policy reinforcement learning method, especially the deterministic off-policy reinforcement learning method.

5. Conclusions

In order to solve the problem that the SGAIL method needs long training time for robot, the DGAIL method, which combines DDPG and GAN, is proposed in this paper. First, the discriminator is used to learn the reward function from demonstrations, which can guide the generator to complete the robot grasping task. Then, DDPG is used as the generator for learning action policy on the basis of discriminator. In particular, the demonstration data is also input into the generator to ensure its performance. The experimental results show that, first of all, the DGAIL method can complete the robot motion planning task without environmental reward quickly. In pick-and-place task, the learning speed of DGAIL is about 3 times that of DDPG. Moreover, the difficulty of tasks has less affect on the learning speed of DGAIL. Secondly, the output trajectory of DGAIL has high consistency with the expert trajectory. Thirdly, the DGAIL method improves the stability of the training process. In the following work, we will extend the full connected network structure into the convolution neural network or the LSTM network, in order to complete more complex tasks, such as end-to-end reinforcement learning task or robot complex operation task.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Guoyu Zuo: Conceptualization, Methodology, Writing - review & editing. **Kexin Chen:** Investigation, Software. **Jiahao Lu:** Investigation, Software, Writing - original draft. **Xiangsheng Huang:** Writing - review & editing.

References

[1] P. Abbeel, A.Y. Ng, Apprenticeship learning via inverse reinforcement learning, in: Proceedings of the Twenty-first International Conference on Machine Learning, ACM, 2004, p. 1.
[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O.P. Abbeel, W. Zaremba, Hindsight experience replay, in: Proceedings of Advances in Neural Information Processing Systems, 2017, pp. 5048–5058.
[3] M. Arjovsky, L. Bottou, Towards principled methods for training generative adversarial networks, in: Proceedings of the 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017.
[4] G. Barth-Maron, M.W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, T.P. Lillicrap, Distributed distributional deterministic policy gradients, in: Proceedings of 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, 2018. https://openreview.net/forum?id=SyZipzbCb.
[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai Gym, arXiv:1606.01540 (2016).
[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Proceedings of Advances in Neural Information Processing Systems, 2014, pp. 2672–2680.
[7] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller, et al., Emergence of Locomotion Behaviours in Rich Environments, arXiv:1707.02286 (2017).
[8] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, et al., Deep q-learning from demonstrations, in: Proceedings of Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
[9] J. Ho, S. Ermon, Generative adversarial imitation learning, in: Proceedings of Advances in Neural Information Processing Systems, 2016, pp. 4565–4573.

[10] I. Kostrikov, K.K. Agrawal, D. Dwibedi, S. Levine, J. Tompson, Discriminator-actor-critic: addressing sample inefficiency and reward bias in adversarial imitation learning, in: Proceedings of 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6–9, 2019, 2019. https://openreview.net/forum?id=Hk4fpoA5Km.
[11] S. Levine, N. Wagener, P. Abbeel, Learning contact-rich manipulation skills with guided policy search, in: Proceedings of 2015 IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 156–163, doi:10.1109/ICRA.2015.7138994.
[12] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, in: Proceedings of 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2–4, 2016, 2016 arXiv:1509.02971.
[13] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. Aparicio, K. Goldberg, Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics, in: Proceedings of Robotics: Science and Systems, Cambridge, Massachusetts, 2017, pp. 1957–1964, doi:10.15607/RSS.2017.XIII.058.
[14] J. Mahler, M. Matl, X. Liu, A. Li, D. Gealy, K. Goldberg, Generative adversarial nets, in: Proceedings of 2018 IEEE International Conference on Robotics and Automation (ICRA), 2018, pp. 1–8.
[15] J. Mahler, F.T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kröger, J. Kuffner, K. Goldberg, Dex-net 1.0: a cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards, in: Proceedings of 2016 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2016, pp. 1957–1964.
[16] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature (7540) (2015) 518–529.
[17] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, et al., Multi-goal Reinforcement Learning: Challenging Robotics Environments and Request for Research, arXiv:1802.09464 (2018).
[18] Y. Schroecker, M. Vecerík, J. Scholz, Generative predecessor models for sample–efficient imitation learning, in: Proceedings of 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, 2019.
[19] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: Proceedings of International Conference on Machine Learning, 2015, pp. 1889–1897.
[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal Policy Optimization Algorithms, arXiv:1707.06347 (2017).
[21] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, Deterministic policy gradient algorithms, in: Proceedings of the 31th International Conference on Machine Learning, ICML, Beijing, China, 2014, pp. 387–395.
[22] B.C. Stadie, P. Abbeel, I. Sutskever, Third person imitation learning, in: Proceedings of 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings, 2017. https://openreview.net/forum?id=B16dGcqlx.
[23] M. Vecerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, M.A. Riedmiller, Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards, CoRR arXiv:1707.08817 (2017).
[24] Z. Wang, J.S. Merel, S.E. Reed, N. de Freitas, G. Wayne, N. Heess, Robust imitation of diverse behaviors, in: Proceedings of Advances in Neural Information Processing Systems, 2017, pp. 5320–5329.
[25] C. Xia, A. El Kamel, Neural inverse reinforcement learning in autonomous navigation, Robotics and Autonomous Systems 84 (2016) 1–14.
[26] B.D. Ziebart, A.L. Maas, J.A. Bagnell, A.K. Dey, Maximum entropy inverse reinforcement learning., in: Proceedings of AAAI, 8, Chicago, IL, USA, 2008, pp. 1433–1438.

Guoyu Zuo received his Ph.D. degree from Beijing University of Technology in 2005, and from then he was Lecture at Beijing University of Technology. He was Associate Professor at Beijing University of Technology from 2012. He is currently Head of Intelligent Robot Laboratory of Beijing University of Technology. Guoyu Zuo is the author of over 30 academic articles and over 20 patents. His research interests include robot control, robot learning, and pattern recognition.

Kexin Chen received his Bachelor's degree from Beijing University of Technology in 2017, and he is currently a master student at the Institute of Intelligent Robotics in Beijing University of Technology. His main research interests include Pattern Recognition and robot learning.

Jiahao Lu received his Bachelor's degree from Beijing University of Technology in 2016, and he is currently a master student at the Institute of Intelligent Robotics in Beijing University of Technology. His main research interests include Pattern Recognition and robot learning.

Xiangsheng Huang received the B.S. degree in material science and the M.S. degree in computer science from Chongqing University, Chongqing, China, in 1998 and 2002, respectively, and the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2005. He is currently an Associate Professor with the Institute of Automation, Chinese Academy of Sciences. From 2005 to 2010, he was with the Samsung Advanced Institute of Technology. He has authored or co-authored over 30 papers in international journals and conferences. He holds ten patents. His current research interests include face technology, 3-D imaging and registration, machine learning, and wavelet and filter banks.