Hindawi
Mathematical Problems in Engineering
Volume 2019, Article ID 4834516, 8 pages
https://doi.org/10.1155/2019/4834516

Research Article

A Study of Continuous Maximum Entropy Deep Inverse Reinforcement Learning

Xi-liang Chen, Lei Cao, Zhi-xiong Xu, Jun Lai, and Chen-xi Li

Command & Control Engineering College, Army Engineering University, No. 1, Hai Fu Road, Guang Hua Road, Qin Huai District, Nanjing City, Jiangsu Province, 210007, China

Correspondence should be addressed to Xi-liang Chen; 383618393@qq.com

Received 24 August 2018; Revised 19 March 2019; Accepted 26 March 2019; Published 8 April 2019

Academic Editor: Ali Ramazani

The assumption of IRL is that demonstrations are optimally acting in an environment. In the past, most of the work on IRL needed to calculate optimal policies for different reward functions. However, this requirement is difficult to satisfy in large or continuous state space tasks. Let alone continuous action space. We propose a continuous maximum entropy deep inverse reinforcement learning algorithm for continuous state space and continues action space, which realizes the depth cognition of the environment model by the way of reconstructing the reward function based on the demonstrations, and a hot start mechanism based on demonstrations to make the training process faster and better. We compare this new approach to well-known IRL algorithms using Maximum Entropy IRL, DDPG, hot start DDPG, etc. Empirical results on classical control environments on OpenAI Gym: MountainCarContinues-v0 show that our approach is able to learn policies faster and better.

1. Introduction

In reinforcement learning, an agent aims to learn a policy for acting in environment [1]. Compared with supervised learning and semisupervised learning, reinforcement learning can perceive the feedback status of the environment in real time and do it without labels, using continuous trial-and-error mechanism and exploitation-exploration strategy. Finally, by adjusting the parameters continuously, the optimal policy for a task can be selected. In 2013, DeepMind proposed the DQN algorithm, using experience memory mechanism to break the relevance of reinforcement learning samples and achieve a state-of-the-art performance in Atari games [2]; in 2015, DeepMind proposed to improve the DQN algorithm by using the target separation mechanism. Besides, the success in AlphaGo zero and Alpha zero provides a practical solution for solving the high computational complexity problems in complex system optimization by means of deep reinforcement learning [3]. However, the formation speed and quality of the optimal strategy of deep reinforcement learning depend heavily on the setting of reward function. In practice, there are some problems as follows: first, the sample utilization rate of deep reinforcement learning is very low; second, in practical multistep reinforcement learning, it is very difficult to design an appropriate reward function.

The goal of inverse reinforcement learning is to generate the structure of potential reward function based on demonstrations. This approach to modeling the reward function provides a method for agents to imitate the specific behavior from the demonstrator [4]. Most of the current methods are based on the parameterization of the reward function. Abbeel, Ratliff, Ziebart, etc. [5–7] adopt the feature of weighted linear combination reward function to achieve better generalization performance. But the original IRL algorithms and many variants assume that the reward function has a linear combination of features. This assumption is usually unreasonable for practical tasks because it has been shown that the quality of learning policy is greatly jeopardized by the error in value estimation [8].

To overcome the inherent limitations of linear models, Choi et al. [9] extend this approach to a limited set of nonlinear reward functions by learning a set of composites of logical conjunctions of atomic features. Nonparametric methods, such as Gaussian Processes (GPs) are used to satisfy potentially complex, nonlinear reward functions [8]. However, this approach tends to require a large number of

reward samples to approximate the complex return function [10–12]. Even the sparse Gaussian process described in [8] makes the time complexity of the algorithm dependent on the number of action sets or the number of state reward pairs. Situations with increasingly complex reward function leading to higher requirements regarding the number of inducing points can quickly render this nonparametric approach infeasible in calculation. Besides, Fourier transformation, wavelet transformation, least square method, nonparametric representation of the value function by defining the augmented inverse propensity weighting estimator to approximate the value function Q, and other traditional function approximation methods have also achieved good results in some areas. Nevertheless, compared with other approximation methods, neural network function approximation represents a class of affine expansion of nonlinear functions. Its special advantage is that the degree of freedom of affine basis selection is large, and the expansion coefficient can be obtained by a unified training algorithm, which is more stable than the traditional algorithm. It can be said that the end-to-end function approximation ability of deep neural network provides a very useful attempt for the application of function approximation theory in practical problems.

The end-to-end learning method can map the original input directly to the reward value without compressing or preprocessing the input data. However, the traditional methods such as apprenticeship learning, maximum marginal method, maximum entropy, and cross entropy cannot be well extended to the tasks with a large number of states [13–15]. Another perspective is learning a loss function to help the learning process.

Therefore, by combining these algorithms with deep neural network, the agent can learn the reward of state action pairs in the neural network, which is complex or large according to the need. The combination of deep neural network and inverse reinforcement learning makes it possible to use the complex correlation of environment and state features to learn the reward function by deep neural network. Ziebart B. et al. [7] proposed a maximum entropy based deep inverse reinforcement learning method, and Sharifzadeh et al. [16] proposed the projection-based IRL algorithm in NIPS2016 with the help of DQN's powerful ability to solve MDP and got a good application in autonomous driving. However, on one hand, DQN cannot be used in the problem of continuous action space; on the other hand, these methods have to solve MDP iteratively, which is inefficient. Besides, the absence of reward shaping between the environmental reward function and the IRL generated reward function may also result in the absence of environmental feedback during the training process.

In this paper, we address a continuous maximum entropy deep inverse reinforcement learning algorithm, which realizes the depth cognition of the environment model by the way of reconstructing the reward function based on the demonstrations. In order to Increase the utilization of demonstrations and accelerate the convergence of deep reinforcement learning, this paper uses the hot start mechanism and uses this mechanism to improve the DDPG algorithm combined with the continuous maximum entropy deep inverse reinforcement learning algorithm, and it carries out experimental verification in the Open AI Gym environment. The results reveal that our approach performs significantly better than the original Maximum Entropy IRL model and the origin DDPG algorithm.

2. Related Work

2.1. Inverse Reinforcement Learning. The most intuitive representation of an expert's intention is the demonstrations generated from its policy. But using methods of supervised learning directly learn a policy from expert's demonstrations usually suffering from insufficient number of training samples and poor generalization ability while the reward function can succinctly represent the expert's knowledge, and this knowledge is transferable to other scenarios [17].

In many tasks, the sequence of tasks performed by experts is considered to obtain relatively high cumulative rewards. When an agent has done a complex task well, without considering the reward function explicitly, this does not mean that the agent has no reward function. To some extent, the agent has potential reward functions in completing specific tasks. Ng et al. proposed that [5, 14] when an agent is performing a task, its decisions are often optimal or near optimal. It can be assumed that when the cumulative expectation of reward generated by all the policies is not better than the cumulative expectation of reward generated by the demonstrations, the corresponding reward function is the potential reward function by demonstrations.

Therefore, inverse reinforcement learning can be defined as a reverse procedure of RL problems, which assumes that the demonstrations from experts are generated from optimal policy π^*. IRL considers the case in MDP where the reward function is unknown. Correspondingly, there is a demonstration set $D = \{\xi_1, \xi_2, \ldots, \xi_n\}$, which is composed of expert demonstration trajectories. Each demonstration trajectory includes a set of state action pairs, which are $\xi_i = \{(s_0, a_0), (s_1, a_1), \ldots, (s_k, a_k)\}$. Thus, we define an MDP/R process with no reward function, defined as tuple $\{S, A, T, \gamma, D\}$ [5]. The aim of inverse reinforcement learning is to learn the potential reward function R.

2.2. Reward Function Approximation with Deep Neural Network. The inverse reinforcement learning assumes that the state space S and the action space A are known. When an agent acts in the decision space according to a policy, a decision trajectory $(s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_n, a_n, r_n,)$ will be generated. In order to make the algorithm produce the behavior consistent with the demonstrations' decision trajectory, it is equivalent to solving the optimal policy in the environment of a reward function. The decision trajectory of the optimal policy is consistent with the demonstrations.

The output of inverse reinforcement learning algorithms is the reward function. The reward function is learned so as to avoid setting the reward function artificially. However, when learning the reward function, we introduce the characteristic

function which needs to be specified artificially; that is, we have assumed the form of the reward function is

$$r_\theta(s) = \theta^T \phi(s) \tag{1}$$

$\phi(s)$ is an artificial characteristic function. For large-scale tasks, the artificially specified eigenfunctions cannot be expressed well and can only cover part of the reward function form, and it is difficult to generalize to other state spaces. One solution is to use neural networks to represent the reward function. At this point, the reward function can be expressed as $r(s) = \theta^T f(s)$, where $f(s)$ is the characteristic function, as shown in Figure 1.

Then the expectation of cumulative reward by policy π is

$$V^\pi = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \mid \pi\right] = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t \theta^T f(s) \mid \pi\right] \tag{2}$$

2.3. Deep Deterministic Policy Gradient. The value function approximation method represented by DQN algorithms has made a breakthrough progress, but the method based on value function still has many limitations: it is difficult to find the randomness policy, and it is not suitable to solve the problem of continuous action. Therefore, the policy search method based on deep neural network is an indispensable part of deep reinforcement learning.

David Silver proved the gradient formula and proposed the DPG algorithm. DPG algorithm contains a parameterized actor policy function, which represents that the current policy can map state to an action deterministically [14]. The Critic value function is based on Q-Learning, and David Silver proves the equation:

$$\nabla \mathrm{J}(\theta) = E_{\pi_\theta}\left[\nabla_\theta \pi_\theta(s) \nabla_a Q^{\pi_\theta}(s, a) \mid a = \pi_\theta(s)\right] \tag{3}$$

On the basis of DPG, the DDPG algorithm combines Actor-Critic algorithm with deep neural network. DDPG uses the underlying idea of DQN in the continuous state-action space. It is an Actor-Critic Policy learning method with added target networks to stabilize the learning process. Besides, batch normalization is used to improve the training performance of deep neural network [15].

3. Continuous Maximum Entropy Deep Inverse Reinforcement Learning

3.1. Inverse Reinforcement Learning Based on Sequence Demonstration Samples. Section 2.1 introduces the inverse reinforcement learning algorithm. The output of the inverse reinforcement learning algorithm is the reward function. However, the goal of sequential decision-making is to find the optimal policy function for decision-making. When there are demonstration trajectories, we can use the inverse reinforcement learning algorithms to solve the reward function r_D according to the demonstration dataset D. When the reward function r_D is solved, the reward can be obtained by combining the reward function r_D with the inherent reward r_ϵ in the environment. After obtaining the reward function, the policy function can be improved by reinforcement

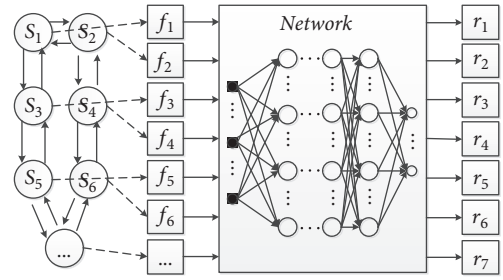

Figure 1: Reward function approximation based on deep natural network.

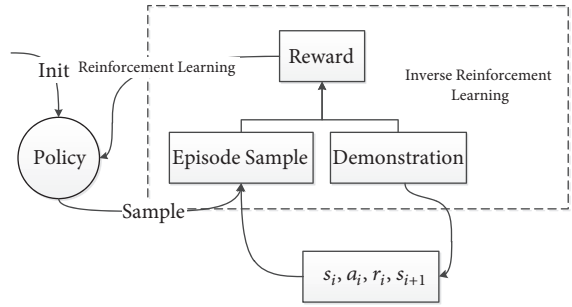

Figure 2: Inverse reinforcement learning process based on sequence experts' demonstrations.

learning according to the reward function. After the policy is improved, the optimal example trajectory of the current policy can be obtained according to the policy function. After sampling, the trajectory data set of the inverse reinforcement learning demonstration can be formed together with the expert example trajectory, and the reward function can be optimized iteratively. Thus, with the continuous optimization of the policy, the reward function becomes more and more close to the optimal reward function, so the estimation of the corresponding reward function will be more accurate. The process is shown in Figure 2.

3.2. Deep Inverse Reinforcement Learning Based on Maximum Entropy. The traditional maximum entropy inverse reinforcement learning can only be used in small-scale and discrete tasks because of the limitation of representational ability [7]. Inverse reinforcement learning with deep neural network architecture approximating the reward function enables it to characterize nonlinear functions by combining and reusing many nonlinear results in a hierarchical structure [12]. In addition, DNNs provide good computational complexity O (1) relative to the demonstrations, making it easy to extend to large state spaces and complex reward functions.

Under these circumstances, the final optimization problem under the continuous maximum causal entropy formulation is nonconvex. We need to use gradient based algorithms to train DNN. Gradient descent method updates model parameters based on the product of derivative and step size, so it is very easy to fall into local optimum. To alleviate this problem, many improved versions of gradient descent algorithms such as batch gradient descent, stochastic gradient descent, Nesterov accelerated gradient, Adagrad,

Adadelta, RMSprop, and Adam have been proposed. Overall, RMSprop is an extension of Adagrad to handle the rapidly decreasing learning rate in Adagrad. RMSprop is the same as Adadelta, but the difference is that Adadelta updates rules using the root mean square of parameters. Finally, Adam adds bias correction and momentum to RMSprop. In such cases, RMSprop, Adadelta, and Adam are very similar algorithms and perform well in similar environments. Kingma et al. pointed out that deviation correction could help Adam to outperform RMSprop slightly in the late optimization stage as gradients became more and more sparse. Some research work s focus on the widely used stochastic mirror descent (SMD) family of algorithms (which contains stochastic gradient descent as a special case) and proved that the last iterate of SMD converges to the set of problem solutions with probability 1. In problems with sharp minima especially (such as generic linear programs or concave minimization problems), SMD reaches a minimum point in a finite number of steps, even in the presence of persistent gradient noise. In general, Adam may be the best choice. Further, Ghadimi S et al. [18] proposed randomized stochastic gradient (RSG) method to solve nonlinear (possibly nonconvex) stochastic programming (SP) problems. In many machine learning problems where either the loss function or the regularization is nonconvex, convergence to stationary points is typically guaranteed

The task of solving the IRL problem can be limited to the maximum posterior probability of Bayesian reasoning. Under the given structure of the reward function and the model parameters θ, the joint posterior distribution of the observing demonstration D can be maximized.

$$L(\theta) = \log P(D, \theta \mid r) = \log P(D \mid r) + \log P(\theta) \tag{4}$$

with $l_D = \log P(D \mid r)$, $l_\theta = \log P(\theta)$

The joint logarithmic likelihood is differentiable from the network parameter θ, so the gradient descent method can be used for optimization. The maximum entropy-based objective function given by the data term L_D of the equation is differentiable from the reward r, so the weight of the target gradient can be propagated back to the network. The final gradient is the sum of the gradient of L_D and L_θ.

$$\frac{\partial L}{\partial \theta} = \frac{\partial L_D}{\partial \theta} + \frac{\partial L_\theta}{\partial \theta} \tag{5}$$

The gradient of the data item can be expressed by the derivative of the reward shown by the demonstrations and the derivative of the reward to the network weight θ, as follows:

$$\frac{\partial L_D}{\partial \theta} = \frac{\partial L_D}{\partial r} \bullet \frac{\partial r}{\partial \theta} = (\mu_D - E[\mu]) \bullet \frac{\partial}{\partial \theta} g(f, \theta) \tag{6}$$

In the above formula, $r = g(f, \theta)$. The derivative of L_D versus reward r is equal to the difference between the number of state visits of the demonstrations and the expected number of visits of the learning system trajectory distribution, which depends on the approximation of the reward function given by the corresponding optimal policy.

$$E[\mu] = \sum_{\xi:(s,a)\in\xi} P(\xi \mid r) \tag{7}$$

The computation of $E[\mu]$ involves summation over exponentially many possible trajectories. Ziebart et al. [7] proposed an effective algorithm based on dynamic programming which can calculate this quantity in polynomial time.

The loss function L_D^n can be described as $L_D^n = \log(\pi^n) \times \mu_D^\alpha$ and the gradient derivation can be denoted as

$$\frac{\partial L_D^n}{\partial r^n} = \mu_D - E[\mu^n] \tag{8}$$

$\partial L_D / \partial \theta$ first uses this algorithm to calculate the difference in the number of visits and then passes this value back to the network as an error signal [10].

$$\frac{\partial L_D^n}{\partial \theta_D^n} = nn\_backprop\left(\frac{\partial L_D^n}{\partial r^n}\right). \tag{9}$$

In order to calculate the loss function value, the frequency of state-action pairs is denoted as μ_D^a. And μ_D^a is obtained by summing the frequency of all actions $\mu_D = \sum_{a=1}^{A} \mu_D^a$.

3.3. Reward Shaping. Reward shaping is a commonly used technology to enhance learning performance in reinforcement learning tasks. The concept of reward shaping is to provide agents with complementary incentives to encourage the transfer to a state of high reward in the environment. If these rewards and punishments are arbitrarily applied, they may make agents deviate from their intended goals. In this case, the agents will converge to the optimal policy in the case of shaping reward, but they are suboptimal in the original task.

Reward shaping attempts to obtain more accurate rewards than the original environment rewards by introducing additional rewards to accelerate the convergence rate of reinforcement learning. Generally, the new reward function is expressed as follows:

$$r_P(s, a) = r_D(s, a) + r_\epsilon(s, a) \tag{10}$$

Where r_ϵ denotes the reward function of the environment in which the original task own. r_D denotes the reward function derived from the inverse reinforcement learning based on demonstrations. In general, the reward function of the environment is sparse and return functions are constructed based on prior knowledge, which has proved effective in many cases [19, 20].

3.4. Nonlinear IRL with Gradient Descent. We denoted $r_D(s, a : \theta_r)$ as the reward function represented by deep neural network with parameter θ_r and $D_{demo} = \{\zeta_1, \zeta_2, \ldots, \zeta_n\}$ as demonstration dataset and $\zeta_i = (s_{i1}, a_{i1}, r_{i1}, s_{i2}, a_{i2}, r_{i2}, \ldots, s_{in}, a_{in}, r_{in})$ as trajectories in demonstrations. The reward function generated by inverse deep reinforcement learning algorithms can be optimized by gradient $(dL_D^{\theta_r}/d\theta_r)(\theta_r)$. This method can be directly applied to the target decomposed by training samples, but the partition function cannot be decomposed by this method. Nevertheless, we find that our goal can still be optimized

Input: D_{demo}
Output: optimal reward function weight θ_r
(1) θ_r = initialize_all_variables()
(1) **for** iteration $k = 1$ to K **do**
(2) Sample demonstration batch $\widehat{D}_{demo} \subset D_{demo}$
(3) Sample background batch $\widehat{D}_{samp} \subset D_{samp}$
(4) Append demonstration batch to background batch: $\widehat{D}_{samp} \longrightarrow \widehat{D}_{Demo} \cup \widehat{D}_{samp}$
(5) set $L(\theta) = \log P(D, \theta \mid r)$ and Estimate $(dL_D^{\theta_r}/d\theta_r)(\theta_r)$ using $\widehat{D}_{Demo}$ and $\widehat{D}_{samp}$
(6) Update parameters θ using gradient $(dL_D^{\theta_r}/d\theta_r)(\theta_r)$
(7) **end for**
(8) **return** optimized parameters θ_r

Algorithm 1: Nonlinear IRL with gradient descent.

by stochastic gradient method. In each iteration, a subset from demonstrations $\widehat{D}_{demo}$ and a subset from background sample $\widehat{D}_{samp}$ are combined together. In Algorithm 1, the stochastic optimization process is given and implemented with neural network optimized based on backpropagation.

In the Algorithm 1, we summarize the IRL algorithm based on stochastic gradient descent. We call this method guided reward function generation learning because the policy optimization is used to guide the sample to train reward function to maximize the expectation of trajectory features. The algorithm consists of taking successive policy optimization steps. After sampling from the latest sample distribution D_{samp} in each step, the algorithm updates the parameters of the reward function by using all the samples collected so far and sampling from the set of demonstration trajectories. This method does not require additional background samples. This process returns a learnt reward function $r_D(s_t, a_t : \theta_r)$.

In addition, the demonstration trajectories can be used to pretraining the Critic network and Actor network in order to make them hot start. DAQN is used to learn the parameters of the Critic network and estimate the value function of the action in a certain state. The structure outputs softmax predictions for each possible action. Therefore, for each state, the next action of network prediction is to be taken. The loss function of this network is

$$J(w) = \sum_a \left[Q\left(s, a \mid \theta^Q\right) - \widehat{Q}(s, a) \right]^2 \tag{11}$$

Among them, θ^Q is the learned weight, $Q(s, a \mid \theta^Q)$ is the output of DAQN, and $\widehat{Q}(\mathrm{s}, \mathrm{a})$ is the actual action taken by demonstrations, using a one-hot array. Therefore, for input $s = s_t, a = a_t \in D_E$, if $a = a_t$ then $\widehat{Q} = 1$. Similarly, we can use the demonstrations to hot start the Actor network $\mu(\mathrm{s} \mid \theta^\mu)$. In this way, we initialize the Critic network and Actor network of DDPG algorithm with hot-start $Q(s, a \mid \theta^Q)$ and $\mu(\mathrm{s} \mid \theta^\mu)$ and update the reward function with reward shaping, as shown in Algorithm 2.

4. Experiments

4.1. Experimental Setup. So far, we have carried out our experiments on classical control environments on OpenAI Gym: MountainCarContinues-v0. Each task comes with a true reward function, defined in the OpenAI Gym [20]. We first generated expert behavior for these tasks by running DDPG [15] on these true reward functions to create expert policies. Expert demonstrations are the optimal trajectories obtained based on DDPG algorithm. To evaluate the performance of imitation learning and inverse reinforcement learning, we sample different data sets from demonstrations.

We trained the algorithms using Adaptive Moment Estimation (Adam) algorithm to minimize the loss with learning rate $\mu = 0.00001$ and set the batch size to 32. The summary of the configuration is provided below. The target network updated all 300 steps. The behavior policy during training was ε-greedy with ε annealed linearly from 1 to 0.01 over the first five thousands steps and fixed at 0.01 thereafter. We used a replay memory of ten thousands most recent transitions.

We independently executed each method 100 times, respectively, on every task. For each running time, the learned policy will be tested 100 times without exploration noise or prior knowledge by every 100 training episodes to calculate the average scores. We report the mean and standard deviation of the convergence episodes and the scores of the best policy.

4.2. Results and Analysis. Unlike supervised learning, deep reinforcement learning has no training data set and verification data set, so it is difficult to evaluate the training performance of the algorithm online. Therefore, there are two main ways to evaluate the training effect. One is to use the accumulative reward value. After a certain period of training, the larger the accumulative average reward value, the better the performance. The other is that the quicker and more stable the Q network, the better the convergence of the algorithm. As shown in Figure 3, the experiment selects the performance of five training methods when the number of expert sample trajectories N_T equals 200: (1)

Input : D_{demo} : initialized with demonstration data set, Initialize critic network $Q(s, a \mid \theta^Q)$ and actor $\mu(\mathrm{s} \mid \theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \longleftarrow \theta^Q$
Initialize replay buffer D
for episode = 1, M **do**
  Initialize a random process N for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
  Select action $a_t = \mu(s_t \mid \theta^\mu) + N_t$ according to the current policy and exploration noise
  Execute action a_t and observe reward r_t and observe new state s_{t+1}
  use D_{demo} to get reward function weight θ_r and $r_D(s_t, a_t : \theta_r)$
  reward shaping $r_t^*(s_t, a_t) = r_D(s_t, a_t : \theta_r) + r_t$ using Algorithm 1
  Store transition $(s_t, a_t, r_t^*, s_{t+1})$ in D
  Sample a random minibatch of N transitions (s_t, a_t, r_t, s_{t+1}) from D
  Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid \theta^{\mu'}) \mid \theta^{Q'})$
  Update critic by minimizing the loss: $L = (1/N)\sum_i (y_i - Q(s_i, a_i \mid \theta^Q))^2$
  Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q\left(s, a \mid \theta^Q\right)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu\left(s \mid \theta^\mu\right)|_{s_i}$$

  Update the target networks:
  $\theta^{Q'} \longleftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
  $\theta^{\mu'} \longleftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
 end for
end for

Algorithm 2: Continuous Maximum Entropy Deep Inverse reinforcement learning with Hot start.

the DDPG algorithm without expert demonstrations; (2) hot start DDPG (HS-DDPG) with initializing the Actor network and Critic network by supervised learning; (3) inverse maximum entropy method with reward shaping (ME DDPG). (4) ME DDPG with initializing the Actor network and Critic network by supervised learning (HS-ME-DDPG) is used. In addition, the average cumulative reward value of the expert demonstrations is shown as a comparison.

As shown in Figure 4, we can see that although all the algorithms have achieved near expert demonstration results after running about 120 episodes, it is obvious that HS-ME-DDPG achieves the most stable and fast convergence performance. The HS-DDPG algorithm, which initializes Actor network and Critic network by supervised learning only using expert demonstration trajectory, gets a rapid increase in average reward at the beginning, but the generalization ability of the trained model is poor because of the demonstrations only containing a small proportion of state and action space. The performance is no better than origin DDPG. But Inverse Reinforcement Learning (IRL) provides an efficient tool for generalizing the demonstration. It has achieved better performance than supervised learning.

Figure 4 shows the relationship between episode use before convergent and the number of Demonstrations N_T. Experimental results show the origin DDPG convergence after about 120 episode with no expert demonstrations used. Although HS-DDPG has improved by using the expert demonstrations to initialize the actor network and the critic network, the convergence rate was earlier entering the linear range with the increase of the amount of data. But ME-DDPG and HS-ME-DDPG use the maximum entropy deep inverse reinforcement learning to get the reward function for solving the optimal policy convergence after about 50 episodes.

Figure 5 shows the relationship between Average reward after convergent and the number of Demonstrations N_T. Experiments show that the maximum entropy deep inverse reinforcement learning algorithm not only improves the convergence speed, but also achieves better results than the expert strategy because the reward function obtained from the expert demonstrations can better reflect the information of the agent getting the optimal policy when completing the task.

5. Conclusion

We present a continuous maximum entropy deep inverse reinforcement learning algorithm, which realizes the depth cognition of the environment model by the way of reconstructing the reward function based on the demonstrations, provides a convex, computationally efficient procedure for

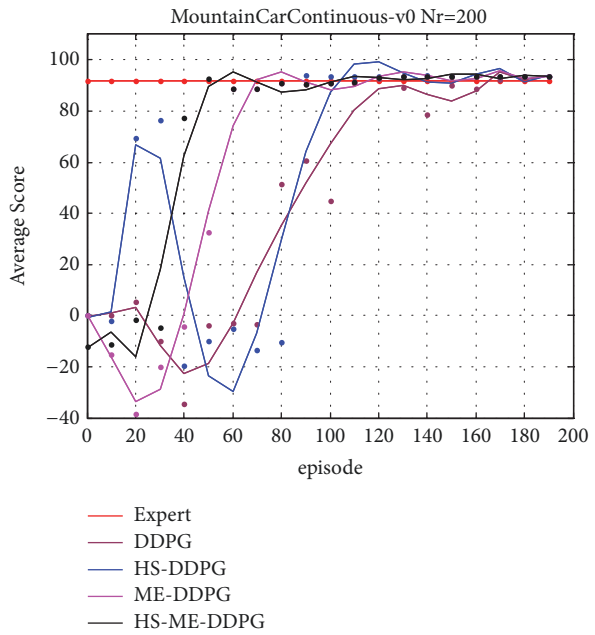

Figure 3: Performance comparison of all algorithms in terms of the average reward on MountainCarContinues-v0.

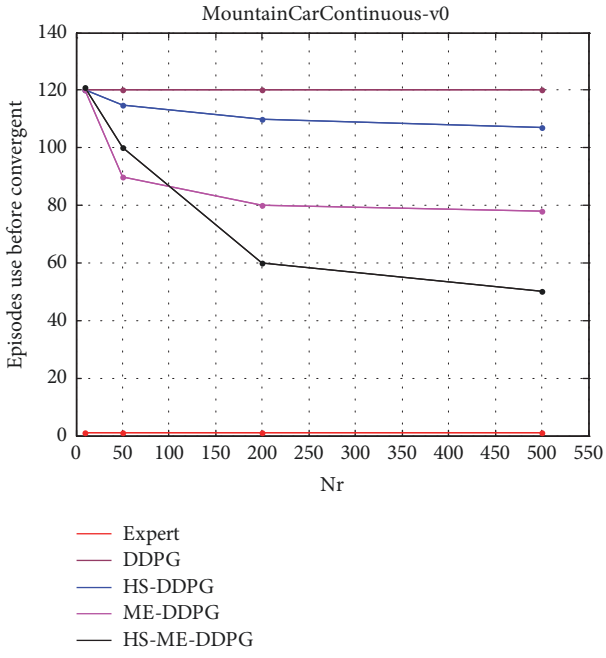

Figure 4: The relationship between episode use before convergent and the number of Demonstrations N_T.

optimization, and maintains important performance guarantees. Our experiments show that DNNs lend themselves naturally to approximate the structure of reward function as they combine representational power with computational efficiency compared to state-of-the-art methods. Our experiments also show that the IRL method is applicable to continuous state space and action space.

In future work, we plan to experiment with more difficult tasks and explore other stochastic optimization techniques

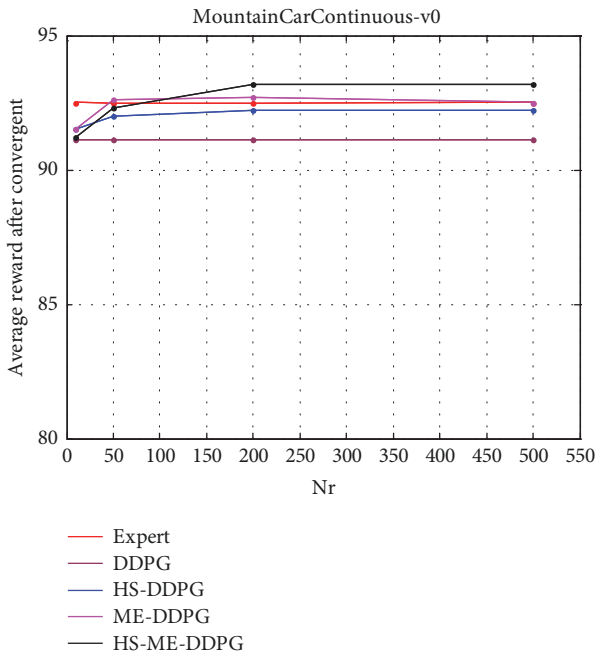

Figure 5: The relationship between average reward after convergent and the number of Demonstrations N_T.

to make IRL algorithms more effective. Especially in the face of large-scale complex tasks, we need to use distributed asynchronous stochastic gradient descent. Under this circumstance, delay and convergence are the difficulties faced by distributed asynchronous stochastic gradient descent. Being judiciously chosen is to use quasilinear step-size sequence reaffirming the application of DASGD to large-scale optimization problems. This is also a useful attempt to solve large-scale problems in deep inverse reinforcement learning. Also, perhaps more apparent, we will explore the benefits of different network types in deep Inverse Reinforcement Learning.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by National Natural Science Foundation of China (61806221).

References

[1] S. Mozer and M. C. Hasselmo, "Reinforcement learning: an introduction," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 16, no. 1, pp. 285-286, 2005.

[2] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Playing atari with deep reinforcement learning," in *Proceedings of the Workshops at the 26th Neural Information Processing Systems*, pp. 201–220, Lake Tahoe, Calif, USA, 2013.

[3] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI '07)*, pp. 2586–2591, Urbana, IL, USA, January 2007.

[5] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the 21st International Conference on Machine Learning (ICML '04)*, pp. 1–8, ACM, July 2004.

[6] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of the 23rd International Conference on Machine Learning (ICML '06)*, pp. 729–736, IEEE, Pennsylvania, PA, USA, June 2006.

[7] B. Ziebart, A. Maas, A. Bagnell, and A. Dey, "Maximum entropy inverse reinforcement learning," in *Proceedings of the 23rd AAAI Conference on Articial Intelligence*, pp. 1433–1438, 2008.

[8] S. Levine, Z. Popovic, and V. Koltun, "Nonlinear inverse reinforcement learning with gaussian processes," *Advances in Neural Information Processing Systems. Granada*, pp. 19–27, 2011.

[9] J. Choi and K. E. Kim, "Bayesian nonparametric feature construction for inverse reinforcement learning," in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pp. 1287–1293, AAAI Press, Catalonia, Spain, 2013.

[10] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning[EB/OL]," 2015, https://arxiv.org/abs/1507.04888.

[11] X. Chen and A. E. Kamel, "Neural inverse reinforcement learning in autonomous navigation," *Robotics & Autonomous Systems*, vol. 84, pp. 1–14, 2016.

[12] Y. Bengio and Y. LeCun, "Scaling learning algorithms towards AI," *Large-Scale Kernel Machines*, vol. 34, no. 5, pp. 1–41, 2007.

[13] A. Y. Ng and J. S. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the 17th International Conference on Machine Learning*, pp. 663–670, Morgan Kaufmann Publishers Inc., 2000.

[14] N. Aghasadeghi and T. Bretl, "Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals," in *Proceedings of the International Conference on Intelligent Robots and Systems: Celebrating 50 Years of Robotics (IROS '11)*, pp. 1561–1566, September 2011.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel et al., "Continuous control with deep reinforcement learning[EB/OL]," 2015, https://arxiv.org/abs/1509.02971.

[16] S. Sharifzadeh, I. Chiotellis, R. Triebel et al., "Learning to drive using inverse reinforcement learning and deep Q-networks," NIPS workshop on Deep Learning for Action and Interaction, IEEE, 2016.

[17] Z. Shao and M. J. Er, "A review of inverse reinforcement learning theory and recent advances," in *Proceedings of the Congress on Evolutionary Computation (CEC '12)*, pp. 1–8, IEEE, June 2012.

[18] S. Ghadimi and G. Lan, "Stochastic first- and zeroth-order methods for nonconvex stochastic programming," *SIAM Journal on Optimization*, vol. 23, no. 4, pp. 2341–2368, 2013.

[19] M. J. Mataric, "Reward functions for accelerated learning," in *Proceedings of the 11th International Conference on Machine Learning*, pp. 181–189, Morgan Kaufmann Publishers, New York, NY, USA, 1994.

[20] I. Zamora, Lopez N. G., Vilches V. M. et al., "Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo[EB/OL]," 2016, https://arxiv.org/abs/1608.05742.