Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 4275623, 12 pages
https://doi.org/10.1155/2020/4275623

Research Article

Deep Ensemble Reinforcement Learning with Multiple Deep Deterministic Policy Gradient Algorithm

Junta Wu and Huiyun Li

Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518071, China

Correspondence should be addressed to Huiyun Li; hy.li@siat.ac.cn

Received 28 October 2019; Revised 15 December 2019; Accepted 30 December 2019; Published 22 January 2020

Academic Editor: Piotr Jędrzejowicz

Deep deterministic policy gradient algorithm operating over continuous space of actions has attracted great attention for reinforcement learning. However, the exploration strategy through dynamic programming within the Bayesian belief state space is rather inefficient even for simple systems. Another problem is the sequential and iterative training data with autonomous vehicles subject to the law of causality, which is against the i.i.d. (independent identically distributed) data assumption of the training samples. This usually results in failure of the standard bootstrap when learning an optimal policy. In this paper, we propose a framework of m-out-of-n bootstrapped and aggregated multiple deep deterministic policy gradient to accelerate the training process and increase the performance. Experiment results on the 2D robot arm game show that the reward gained by the aggregated policy is 10%–50% better than those gained by subpolicies. Experiment results on the open racing car simulator (TORCS) demonstrate that the new algorithm can learn successful control policies with less training time by 56.7%. Analysis on convergence is also given from the perspective of probability and statistics. These results verify that the proposed method outperforms the existing algorithms in both efficiency and performance.

1. Introduction

Reinforcement learning is an active branch of machine learning, where an agent tries to maximize the accumulated reward when interacting with a complex and uncertain environment [1, 2]. Reinforcement learning combining deep neural network (DNN) technique [3, 4] had gained some success in solving challenging problems. One of the most noticeable results was achieved through the deep Q-network (DQN), which exploited deep neural networks to achieve maximum accumulated reward [5]. DQN has performed well over 50 different Atari games and inspired many deep reinforcement learning (DRL) algorithms [6–8].

However, DQN only deals with the tasks with small, discrete state and action spaces while many reinforcement learning tasks have large, continuous, real-valued state and action spaces. Although such tasks could be solved with DQN by discretizing the continuous spaces, the instability of the control system may be increased. For overcoming this difficulty, deterministic policy gradient (DPG) algorithm [9] with the DNN technique was proposed, producing deep deterministic policy gradient (DDPG) algorithm [10]. Unfortunately, DDPG suffers from inefficient exploration and unstable training [11]. Many existed works attempted to solve the problems. Gu et al. proposed the Q-prop method, a Taylor expansion of the off-policy critic as a control variant to stabilize DDPG [12]. Q-Prop combines the on-policy Monte Carlo and the off-policy DPG; it achieves the advantages of sample efficiency and stability. Mnih et al. proposed A3C to stabilize the training process of DDPG, by training the parallel agents with asynchronously accumulated updates [13]. Interactive learning with the environment in multiple threads is performed at the same time, and each thread summarizes the learning results and stores them in a common place. In this way, A3C avoids the problem of too strong correlation of empirical playback and achieves an asynchronous concurrent learning model. This method consumes considerable computation resources. When the implementation complexity is not a strong limit, we can use any of these policy gradient-related methods to generate subpolicies to further improve our method, where the centralized experience replay buffer stores and shares experiences from all subpolicies, enabling more knowledge gained from the environment.

Additionally, researchers attempted to overcome the disadvantage of unstable training of DDPG and speed up the convergence of DDPG with bootstrap technique recently [14]. Osband et al. developed bootstrapped DQN as the critic of DDPG [15]. Yang et al. employed a multiactor architecture for multitask purpose [16]. DBDDPG [11] and MADDPG [17] both used multiactor-critic structure to improve the exploration efficiency and increase the training stability. Shi et al. introduced deep soft policy gradient (DSPG) [18], an off-policy and stable model-free deep RL algorithm by combining policy and value-based methods under maximum entropy RL framework. The authors discover that the standard bootstrap is likely to fail when learning an optimal policy, since in most reinforcement learning tasks, the sequential and iterative training data subject to the law of causality, which is against the i.i.d. (independent identically distributed) assumption of the training samples. Hence, a novel bootstrap technique is needed for achieving the optimal policy.

In consideration of the above shortcomings of the previous work, this paper introduces a simple DRL algorithm with m-out-of-n bootstrap technique [19, 20] and aggregated multiple DDPG structures. The control policy will be gained by averaging all learned subpolicies. Additionally, the proposed algorithm uses the centralized experience replay buffer to improve the exploration efficiency. Since m-out-of-n bootstrap with random initialization produces reasonable uncertainty estimates at low computational cost, this helps in the convergence of the training. The proposed bootstrapped and aggregated DDPG can substantially reduce the learning time.

The remainder of this paper is organized as follows. Section 2 presents a brief background. Section 3 introduces the proposed method in detail and analyses the convergence of the algorithm. The experimental results of the proposed method are presented in Section 4. The paper is concluded in Section 5.

2. Background

2.1. Reinforcement Learning. In a classical scenario of reinforcement learning, an agent aims at learning an optimal policy according to the reward function by interacting with the environment E in discrete time steps, where policy is a map from the state space to action space [1]. At each time step, the environment state s_t is observed by the agent, and then it executes the action a_t by following the policy π. Afterwards, a reward $r(s_t, a_t)$ is received immediately. The following equation defines the accumulated reward that the agent receives from step t:

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i), \tag{1}$$

where $\gamma \in [0, 1]$ is a discount factor. As the agent maximizes the expected accumulated reward $E[R_t]$ from the initial state, the optimal policy π^* will be gained finally.

2.2. Deterministic Policy Gradient Algorithm. Policy gradient (PG) algorithms optimize a policy directly by maximizing the performance function with the policy gradient. Deterministic policy gradient algorithm which is originated from deterministic policy gradient theorem [9] is one of the policy gradient methods. It learns deterministic policies $\mu(\cdot \mid \theta)$: $S \longmapsto A$ with the actor-critic framework, while the critic estimates the action-value function and the actor represents the deterministic policy function. The updates for the action-value function and the policy function are given below:

$$\begin{aligned} w &= \arg\min_{w} E_{s\sim\rho^{\mu}(\ ),a\sim\mu(\ |\theta)}\Big[(r(s,a) + \gamma Q(s', \mu(s' \mid \theta) \mid w) \\ &\quad - Q(s,a \mid w))^2 \Big], \\ \theta &= \arg\max_{\theta} E_{s\sim\rho^{\mu}(\ )}[Q(s, \mu(s \mid \theta) \mid w)], \end{aligned} \tag{2}$$

where $\rho^{\mu}(\cdot)$ denotes the discounted state distribution [9]. Since full optimization is expensive, stochastic gradient optimization is usually used instead. The following equation shows the deterministic policy gradient [9] which is used to update the parameter of the deterministic policy:

$$\nabla_{\theta} J = \mathrm{E}_{s\sim\rho^{\mu}(\ ),a\sim\mu(\ |\theta)}\Big[\nabla_{\theta}\mu(s \mid \theta)\nabla_{a} Q(s, a \mid w)\big|_{a=\mu(s|\theta)}\Big]. \tag{3}$$

2.3. DDPG Algorithm. DDPG applies the DNN technique onto the deterministic policy gradient algorithm [10], which approximates deterministic policy function μ and action-value function Q with neural network, as shown in Figure 1.

There are two sets of weights in DDPG. w, θ are weights for main networks while w', θ' are weights for target networks which are introduced in [5] for generating the Q-learning targets. We use $Q(\cdot, \cdot \mid w)$ and $\mu(\cdot \mid \theta)$ to denote the main networks while $Q'(\cdot, \cdot \mid w)$ and $\mu'(\cdot \mid \theta)$ represent the target networks. As equations (4) and (5) shows, weights of the main networks are updated according to the stochastic gradient, while weights of target networks are updated with "soft" updating rule [10], as shown in equation (6):

$$w \longleftarrow w + \alpha_w \Big[(r(s,a) + \gamma Q')(s', \mu'(s' \mid \theta) \mid w') - Q(s, a \mid w))^2 \Big], \tag{4}$$

$$\theta \longleftarrow \theta + \alpha_{\theta} \nabla_{\theta}\mu(s \mid \theta)\nabla_{a} Q(s, a \mid w)\big|_{a=\mu(s|\theta)}, \tag{5}$$

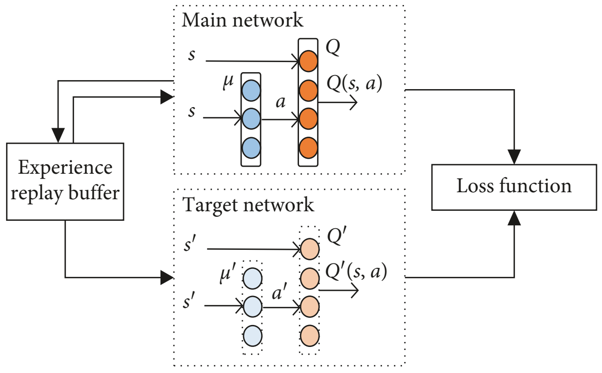


Figure 1: Diagram of deep deterministic policy gradient.

$$\begin{aligned} w' &= \tau w + (1-\tau)w', \\ \theta' &= \tau\theta + (1-\tau)\theta'. \end{aligned} \tag{6}$$

DDPG utilizes the experience replay technique [10] to break training samples' temporal correlation, keeping them subject to the i.i.d. (independent identically distributed) assumption. Furthermore, the "soft" updating rule is used to increase the stability of the training process. DDPG updates the main actor network with the policy gradient, while the main critic network is updated with the idea of combining the supervised learning and Q-learning which is used in DQN. After training, the main actor network converges to the optimal policy.

3. Methods

3.1. Structure of Multi-DDPG. Compared with DQN, DDPG is more appropriate for reinforcement learning tasks with continuous action spaces. However, it takes long time for DDPG to converge to the optimal policy. We propose multi-DDPG structure and bootstrap technique to train several subpolicies in parallel so as to cut down the training time.

We randomly initialize N main critic networks $Q_i(s, a \mid w_i)$ and main actor networks $\mu_i(s \mid \theta_i)$ with weights w_i and θ_i $(i = 1, 2, \ldots, N)$, and then, we initialize N target networks Q'_i and μ'_i with weights $w'_i \longleftarrow w_i, \theta'_i \longleftarrow \theta_i$ $(i = 1, 2, \ldots, N)$ and initialize the centralized experience replay buffer R.

The structure of multi-DDPG with the centralized experience replay buffer is shown in Figure 2. We name the proposed method which utilizes the multi-DDPG structure and bootstrap technique as bootstrapped aggregated multi-DDPG (BAMDDPG). Figure 3 demonstrates that BAMDDPG averages all action outputs of trained subpolicies to achieve the final aggregated policy. For clarity, the terms *agent*, *main actor network*, and *subpolicy* refer to the same thing and are interchangeable in this paper. Algorithm 1 presents the entire algorithm of BAMDDPG.

In Algorithm 1, "**#Env**" means the number of environment modules while "#selected DDPG" represents the number of selected DDPG components. During the training process, each DDPG component which exploits the actor-critic framework is responsible for training the corresponding subpolicy. Figure 2 demonstrates the training process of a DDPG component, containing the interaction procedure and the update procedure.

In the interaction procedure, the main actor network which represents an agent interacts with the environment. It receives the current environment state s_t and outputs an action a_t. The environment gives the immediate reward r_t and the next state s_{t+1} after executing the action. Then the transition tuple $(s_t, a_t, r_t, s_{t+1})_t$ is stored into the central experience replay buffer. To efficiently explore the environment, noise sampled from an Ornstein–Uhlenbeck process $\mathcal{N}$ is added to the action.

In the update procedure, a random minibatch of transitions used for updating weights is sampled from the central experience replay buffer. The main critic network is updated by minimizing the loss function which is based on the Q-learning method [1], while the target networks are updated by having them slowly track the main networks. Weights of the main actor network are updated with the policy gradient along which the overall performance increases. By following such an update rule, each subpolicy of BAMDDPG gradually improves. The centralized experience replay buffer stores experiences from all subpolicies.

Figure 3 illustrates the aggregation details of subpolicies. We denote subpolicies approximated by main actor networks with $\mu_1(\cdot), \mu_2(\cdot), \ldots, \mu_N(\cdot)$ and the outputs of these subpolicies with $a_1, a_2, \ldots, a_N$. In addition, the aggregated policy's output is denoted as a.

In practice, we train multiple subpolicies by setting a maximum number of episodes. Since episodes in BAMDDPG terminate earlier than that of the original DDPG algorithm with less steps, the training time of subpolicies is less than the optimal policy. It can be predicted that the performance of less-trained subpolicies will be worse than the optimal policy to some degree, but we can aggregate the trained subpolicies to increase the performance and get the optimal policy. Furthermore, we use the average method as aggregation strategy in consideration of the equal status and real-valued outputs of all subpolicies. Specifically, the outputs of all subpolicies are averaged to produce the final output.

As Figure 2 demonstrates, the interaction procedure of a DDPG requires an environment component to interact with the agent. Therefore, multi-DDPG structure requires multiple environment modules. However, for some reinforcement learning tasks, the environment module does not support being copied for multiple DDPGs. In such case, the environment component interacts with only one subpolicy in each time step. BAMDDPG supports reinforcement learning tasks with both one environment module and multiple environment modules by choosing one subpolicy or multiple subpolicies to interact with the environments in each time step. All subpolicies are then updated simultaneously with sampled minibatch from the centralized experience replay buffer. In the end, all trained subpolicies are averaged to form the final policy. Algorithm 1 presents the BAMDDPG algorithm.

Additionally, from the perspective of intuition, the centralized experience replay technique exploited in

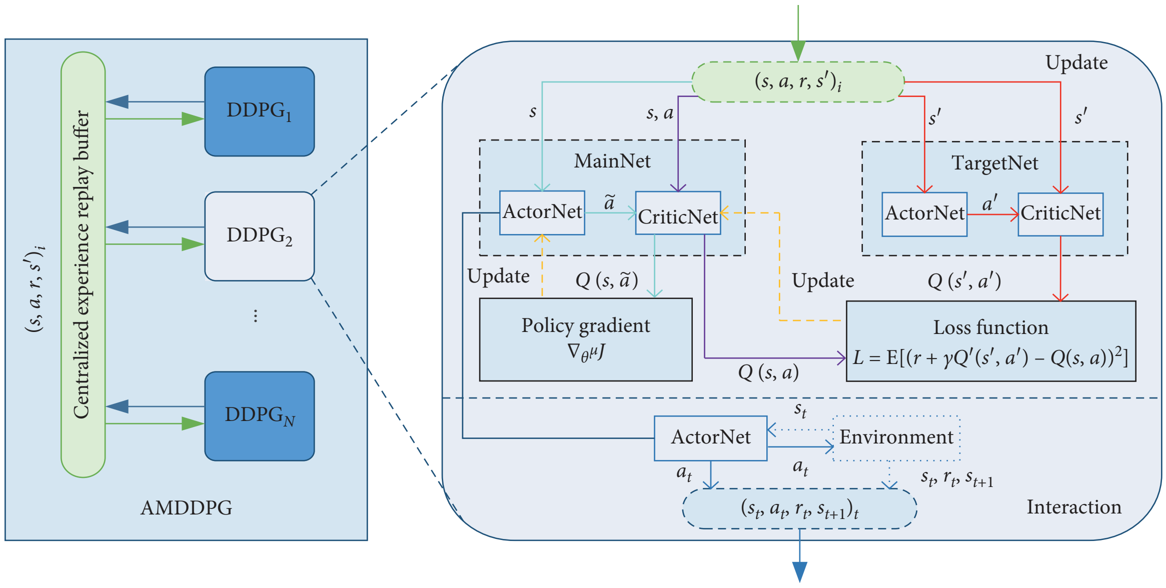


Figure 2: Structure of BAMDDPG.

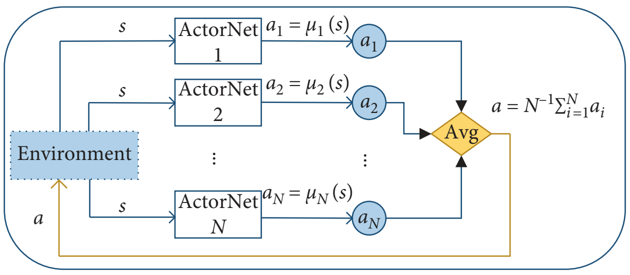


Figure 3: Aggregation of subpolicies.

```
Randomly initialize $N$ main critic networks $Q_i(s, a \mid w_i)$ and main actor networks $\mu_i(s \mid \theta_i)$ with weights $w_i$ and $\theta_i$ $(i = 1, 2, \ldots, N)$
Initialize $N$ target networks $Q_i'$ and $\mu_i'$ with weights $w_i' \longleftarrow w_i, \theta_i' \longleftarrow \theta_i (i = 1, 2, \ldots, N)$
Initialize centralized experience replay buffer $R$
**for** episode = 1, M **do**
  Initialize an Ornstein–Uhlenbeck process $\mathcal{N}$ for action exploration
  **if** #Env == 1 **do**
    Alternately select $Q_i$ and $\mu_i$ among multiple DDPGs to interact with the environment
  **else do**
    Select all $Q_i$ and $\mu_i$, each DDPG is bound with one environment
  **end if**
  **for** $t = 1, T$ **do**
    **for** #selected DDPG **do**
      Receive state $s_t$ from its bound environment
      Execute action $a_t = \mu_i(s_t \mid \theta_i) + \mathcal{N}_t$ and observe reward $r_t$ and new state $s_{t+1}$
      Store experience $(s_t, a_t, r_t, s_{t+1})$ in $R$
    **end for**
    **for** $i = 1, N$ do
      Update $w_i$, $\theta_i$, $w_i'$, and $\theta_i'$ according to equations (4)–(6)
    **end for**
  **end for**
**end for**
Get final policy by aggregating subpolicies: $\mu(s) = N^{-1}\sum_{i=1}^{N}\mu_i(s \mid \theta_i)$
```

Algorithm 1: Bootstrapped and aggregated multi-DDPG (BAMDDPG).

BAMDDPG enables each agent to use experiences encountered by other agents. This makes the training of subpolicies of BAMDDPG more efficient since each agent owns a wider vision and more environment information.

3.2. Analysis on Convergence with Bootstrap and Aggregation. For ease of description, we suppose BAMDDPG trains N subpolicies simultaneously and denote these subpolicies with $\mu_i\,(i = 1, 2, \cdots N)$. The aggregated policy is denoted as $\overline{\mu}$ which can be formulated as

$$\overline{\mu} = \text{Avg}[\mu_i] = N^{-1}\sum_{i=1}^{N}\mu_i, \tag{7}$$

where $\overline{\mu}$ represents the aggregation of subpolicies. Let the optimal policy denoted as μ^*. Then the following formula holds [20]

$$\text{Avg}\left[\left(\mu_i - \mu^*\right)^2\right] \geq \left(\overline{\mu} - \mu^*\right)^2, \tag{8}$$

where $\text{Avg}[(\mu_i - \mu^*)^2]$ means the average bias of subpolicies and the optimal policy while $(\overline{\mu} - \mu^*)^2$ represents bias of the aggregated policy and the optimal policy.

Equation (8) demonstrates that the aggregated policy has better performance than subpolicies and approximates the optimal policy more closely than any subpolicy. Under this conclusion, the aggregated policy approximates the optimal policy quickly as subpolicies are trained to a certain extent [21].

Further, we analyze the convergence from the perspective of probability and statistics [22]. Assume all policies are from the policy space U. The subpolicies $\mu_1, \mu_2, \ldots, \mu_N$ are sampled according to a distribution function F in U. Let $\widehat{F}$ denote the empirical cumulative distribution function as

$$\widehat{F}(x) = N^{-1}\sum_{i=1}^{N} I\left(u_i \leq x\right), \tag{9}$$

where N is the number of the sampled subpolicies. $I[\cdot]$ is an indicator function which outputs 1 when the condition is satisfied, otherwise 0. The operator "$\leq$" in "$u_i \leq x$" means x is a better policy than u_i in U, which indicates the agent acting by following the policy x is able to gain more reward than those only adopting u_i. According to the rule of Dvoretzky–Kiefer–Wolfowitz inequality [23], we get

$$P\left(\sup_{x\in U}|\widehat{F}(x) - F(x)| > \delta\right) \leq 2e^{-2n\delta^2}, \tag{10}$$

where $P(\cdot)$ represents probability, $\sup(\cdot)$ means upper bound, and δ is an arbitrary small positive integer.

Equation (10) shows that $\widehat{F}$ converges uniformly to the true distribution function exponentially fast in probability. Suppose we are interested in the mean $\mu = \text{T}(F)$, then the unbiasedness of the empirical measure extends to the unbiasedness of linear functions of the empirical measure. Actually, empirical cumulative distribution can be seen as a discrete distribution with equal probability for each component, which means we can get a policy from the empirical cumulative distribution by averaging multiple policies. Therefore, the aggregating policy $\overline{\mu}$ subjects to empirical cumulative distribution and it subjects to true distribution. Since $\overline{\mu}$ is a better policy than $\mu_i\,(i = 0, \ldots, n)$ in U, $\overline{\mu}$ converges to the optimal policy of U.

3.3. The m-out-of-n Bootstrap. Bootstrap [14] is a significant resample technique in statistics, which generally works by random sampling with the replacement process. In this paper, we try to train multiple DDPG components with bootstrap. It is analyzed that such requirement can be simply attained by initializing the network weights of different DDPG components with different methods [15]. Therefore, we adopt this technique as a prior and multiple DDPG components are trained in parallel on different subdataset from experience replay buffer.

However, standard bootstrap fails as the training data subject to a long-tail distribution, rather than the usual normal distribution, as the i.i.d. assumption implies. A valid technique is m-out-of-n bootstrap method [19], where the number of bootstrap samples is much smaller than that of the training dataset. More specifically, we draw subsamples without replacement and use these subsamples as new training datasets. Multiple DDPG components are then trained with this newly produced training dataset.

4. Results and Discussion

4.1. 2D Robot Arm. In order to illustrate the effectiveness of aggregation, we use BAMDDPG to learn a control policy for a 2D robot arm task.

4.1.1. Benchmark and Reward Function. As Figure 4 demonstrates, a 2D robot arm contains a two-link arm with one joint which is attempting to get to the blue block. The first link rotates around the root point while the second link rotates around the joint point. The action of an agent consists of two real-valued numbers denoting angular increment. We construct the reward according to the distance between the finger point of the arm (endpoint) and the blue block. The farther away the finger point being from the blue block, the lesser the reward is. Additionally, the reward adds one when the distance $\sqrt{dx^2 + dy^2}$ is less than the threshold δ. When the finger point stops within the blue block for a while (more than 50 iterations), the reward adds ten. The following equation presents the reward:

$$r = -\frac{\sqrt{dx^2 + dy^2}}{200} + I\left[\sqrt{dx^2 + dy^2} < \delta\right] + 10 * I[\eta > 50], \tag{11}$$

where $I[\cdot]$ is an indicator function which outputs 1 when the condition is satisfied, otherwise 0.

4.1.2. Performance of Aggregated Policy. During the training process of BAMDDPG, each agent interacts with its corresponding environment, producing multiple learning curves. Figure 5 demonstrates learning curves of 3 subpolicies with shared experience on 2D robot arm benchmark. The curve depicts the moving average of episode reward while the shaded area depicts the moving

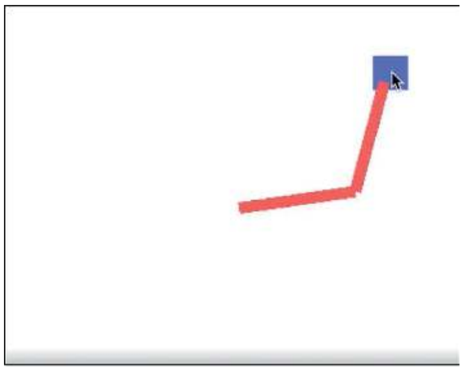


Figure 4: 2D robot arm benchmark.

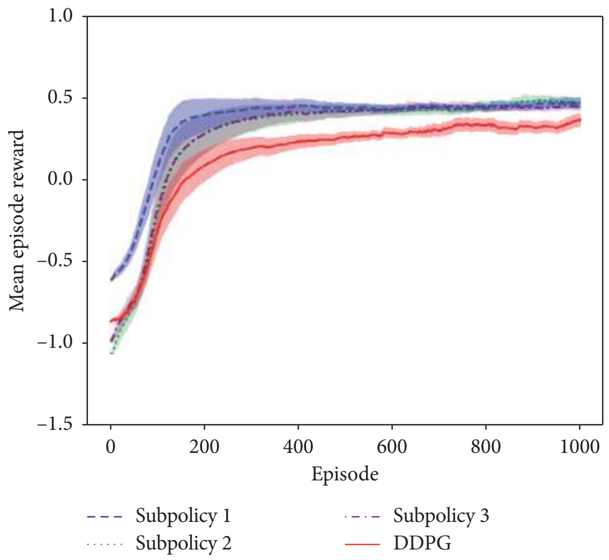


Figure 5: Comparison between sub-DDPGs of BAMDDPG and DDPG.

average ± partial standard deviation. As Figure 5 shows, the training process of BAMDDPG's subpolicies is better than that of DDPG. The centralized experience replay buffer stores and shares experiences from all subpolicies, enabling more knowledge gained from the environment. Therefore, BAMDDPG's subpolicies can gain more reward during the training process. After about 1000 episodes, the subpolicies of BAMDDPG and the policy of the original DDPG both converge.

The key of BAMDDPG is the aggregation of subpolicies. In this section, we show the comparison of performance between the aggregated policy and subpolicies so as to illustrate the effectiveness of aggregation. Suppose the action given by the i^{th} subpolicy is $a_i = [a_{i1}, a_{i2}]$, then the immediate reward of the i^{th} subpolicy is given by

$$IR_i = -\frac{f(a_i)}{200} + I[f(a_i) < \delta] + 10 * I[\eta > 50], \tag{12}$$

where $f(a_i)$ denotes the distance between the finger point of the arm and the blue block after executing action a_i while it is an implicit function. The immediate reward of the aggregated policy can be expressed in the same way:

$$IR = -\frac{f\left(\sum_{i=1}^{n} a_i\right)}{200} + I\left[f\left(\sum_{i=1}^{n} a_i\right) < \delta\right] + 10 * I[\eta > 50], \tag{13}$$

where $\sum_{i=1}^{n} a_i$ represents the action taken by the aggregated policy.

Table 1 shows the performance comparison of subpolicies and aggregated policy of BAMDDPG. The result demonstrates that reward gained by the aggregated policy is 10%~50% better than those gained by subpolicies.

4.2. TORCS

4.2.1. Benchmark and Reward Function.

4.2.1. Benchmark and Reward Function. The Open Racing Car Simulator (TORCS) is a car driving simulation software with high portability, which takes the client-server architecture [24, 25]. It realistically simulates real cars by modeling the physical dynamic models of the car engines, brakes, gearboxes, clutches, etc. It is a commonly used DRL benchmark and is appropriate for test of self-driving techniques. Using TORCS, a developer is able to easily access a simulated car's sensor information. Therefore, the controller of the simulated car is able to get the current environment state and follow a policy to send controlling instructions, including control of steering, brake, and throttle. Figure 6 presents TORCS's client-server architecture. The controller connects to the race server through the user datagram protocol (UDP). At each time step, the information of the current driving environment state is perceived by the simulated car and is sent to the controller. The server then waits for an instruction from the controller for 10 ms. The simulated car executes the corresponding actions according to the current instruction, or last instruction if no new instruction is sent.

Designing a suitable reward function is a key for using TORCS as the platform to test BAMDDPG, which helps to learn a good policy to control the simulated car. We describe the details of designing the reward function in this section. As the driving environment state of TORCS can be perceived by various sensors of the simulated car, we can create the reward function using these sensor data which is shown in Table 2.

Equation (14) presents our constructed reward function, which restricts the behavior of the simulated car in TORCS. Each time the simulated car interacts with the driving environment of TORCS, we expect to gain as large reward as possible through the following equation:

$$r = \left(v \times \left(\frac{1 - |v - \beta|}{\alpha} \right)^{\mathbf{I}[\![d_1 \leq 10]\!]} \right) \times \cos\phi \times (1 - |\sin\phi|) \times (1 - |d_2|) \times \left(\frac{|d_1|}{50} \right)^{\mathbf{I}[\![d_1 \leq 50]\!]}, \tag{14}$$

Table 1: Performance comparison of subpolicies and the aggregated policy.

Policy	Episodes	Total reward	Average reward
Subpolicy1	20	720.69	36.03
Subpolicy2	20	538.28	26.91
Subpolicy3	20	463.98	23.20
Aggregated policy	20	829.17	41.46

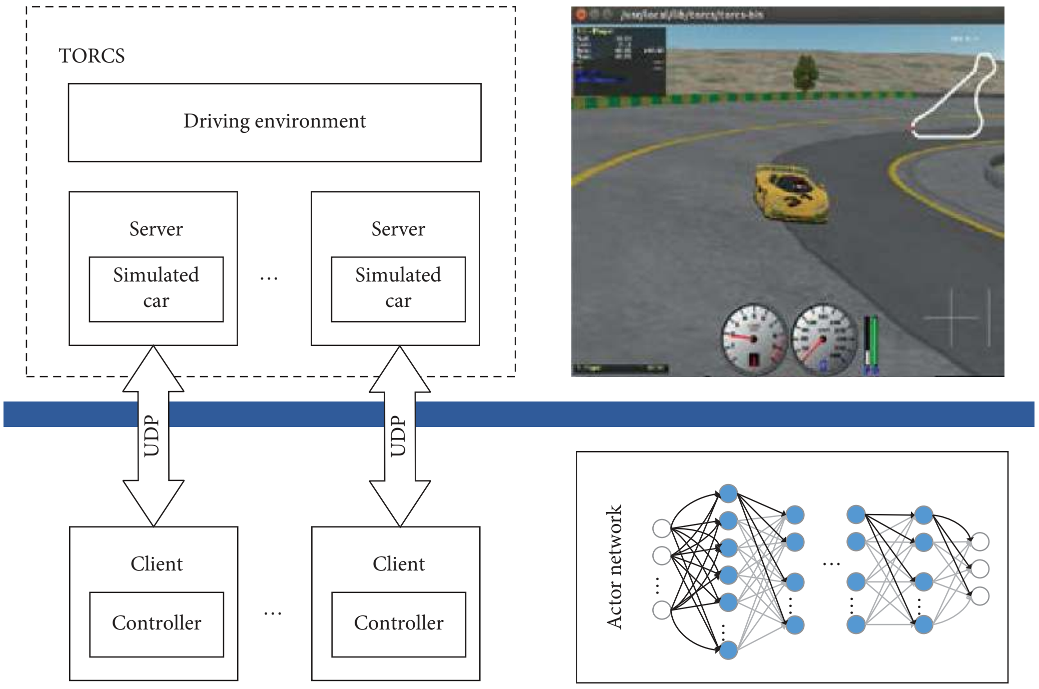


Figure 6: Diagram of the client-server architecture of TORCS.

Table 2: Information of sensor data for creating the reward function.

Name	Range (unit)	Description
ϕ	$[-\pi, +\pi]$ (rad)	Angle between track direction and car's forward direction
v	$[-\infty, +\infty]$ (km/h)	Speed of the car along the direction of the car's longitudinal axis
d_1	$[0, 200]$ (m)	Distance between the track edge ahead and the car
d_2	$[-\infty, +\infty]$	Distance between the track axis and the car. $\lvert d_2\rvert = 1$ means the car is on the right or left edge of the track, $d_2 = 0$ means the car is right on the track axis, and $\lvert d_2\rvert > 1$ means the car is outside of the track

where the term v represents the car is expected to run as fast as possible so as to maximize the reward. The terms $\cos\phi$ and $(1 - \lvert\sin\phi\rvert)$ mean ϕ is expected as zero so that the car can run along the track all the time. The term $(1 - \lvert d_2\rvert)$ represents the car is on the track axis. $I[\![\cdot]\!]$ represents an indicator function whose value is 1 or 0 depending on whether the condition is met or not. The following equation reformulates the first term of equation (14):

$$v \times \left(\frac{1 - \lvert v - \beta\rvert}{\alpha}\right)^{I[\![d_1 \le 10]\!]} = \begin{cases} v \times \left(\dfrac{1 - \lvert v - \beta\rvert}{\alpha}\right), & d_1 \le 10, \\ v, & d_1 > 10. \end{cases} \quad (15)$$

Equation (15) takes into account the speed constraints of the car whether the car encounters a turn or not. The car slows down when a turn is encountered and drives as fast as possible along a straight route. Here, $d_1 = 10$ is set to be the threshold of encountering a turn. The car is at a turn when $d_1 < 10$ and the corresponding reward is a quadratic function with respect to the speed of the car. Note that α and β are hyper parameters needing to fine-tune. Figure 7 illustrates the graph of the quadratic function when $\alpha = 120, \beta = 180$. The quadratic function reaches the maximum value when $v = 90.5$, which means the expected speed of the car at a turn is 90.5 km/h and the car will decelerate automatically when it encounters a turn.

Equation (16) reformulates the last term in equation (14). It restricts the distance between the track edge ahead and the

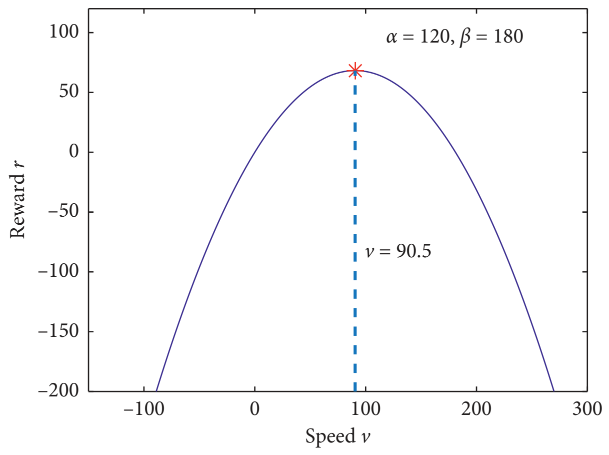


Figure 7: Graph of speed constrained function.

car. This term means that the turn should be observed by the car in advance and the steering angles should be adjusted according to the turn:

$$\left(\frac{|d_1|}{50}\right)^{I[\![d_1\le 50]\!]} = \begin{cases} \dfrac{|d_1|}{50}, & d_1 \le 50, \\ 1, & d_1 > 50. \end{cases} \quad (16)$$

4.2.2. Learning Curve and Training Time. We successfully achieve the optimal self-driving policy with BAMDDPG by aggregating multiple subpolicies in TORCS. During one episode of the training process, one subpolicy is selected. The corresponding agent perceives the driving environment state through various sensors and executes the action by following the selected subpolicy. Table 3 presents the detailed description of the action commands, including steering, brake, and throttle.

After the interaction, all subpolicies were updated using the minibatch from the centralized experience replay buffer. We have argued that less training time is demanded by BAMDDPG than DDPG. Figure 8(a) illustrates the comparison of learning curve between BAMDDPG and DDPG while Figure 8(b) demonstrates the comparison of training time.

In our experiments on TORCS, the simulated car was trained 6000 episodes with the Aalborg track using BAMDDPG and DDPG, respectively. Figure 8(a) illustrates the learning curve comparison of DDPG and BAMDDPG. The curve depicts the moving average of episode reward while the shaded area depicts the mean ± the standard deviation. Figure 8(a) demonstrates that BAMDDPG and DDPG both converge and oscillate around a specific mean episode reward after being trained 6000 episodes. Figure 8(b) demonstrates that BAMDDPG takes less time to train since the aggregated policy quickly approximates the optimal policy as subpolicies are trained to a certain extent. It takes 22.84 hours for BAMDDPG to be trained 6000 episodes, but 52.77 hours for DDPG, which demonstrates BAMDDPG can cut down the training time by 56.7%. Figure 8(b) also shows that training time spent by BAMDDPG and DDPG is not so different in the first 1500 episodes. The reason is that the attention is mostly paid on environment exploring by the simulated car at first and these initial episodes finish quickly. Exploring time spent by BAMDDPG and DDPG is nearly the same. From the perspective of network training, the first 1500 episodes can be considered as the initialization of the corresponding networks.

4.2.3. Effectiveness of Aggregation. The ability of the BAMDDPG algorithm to reduce training time is based on policy aggregation. Section 3.3 illustrated the conclusion that the performance of the aggregated policy is better than that of subpolicies through theoretical analysis. In addition, Section 4.1 has shown the effectiveness of aggregation on 2D robot arm benchmark. In this section, we are to further illustrate the effectiveness of aggregation on TORCS.

In order to avoid the influence of too many subpolicies on the conciseness and contrast of expression, only three subpolicies are trained by the BAMDDPG algorithm in this experiment. The trained subpolicies and the aggregated policy control the same simulated car on the same track, Aalborg track, within one lap. Then, we observe the total reward and whether the car can finish one lap on the track or not. Table 4 illustrates the simulated car controlled by the aggregated policy finishes the Aalborg track and gained much larger total reward than subpolicies, but the cars controlled by subpolicies all left the track and are not able to complete the track, which indicates that aggregation technique does increase the performance of subpolicies.

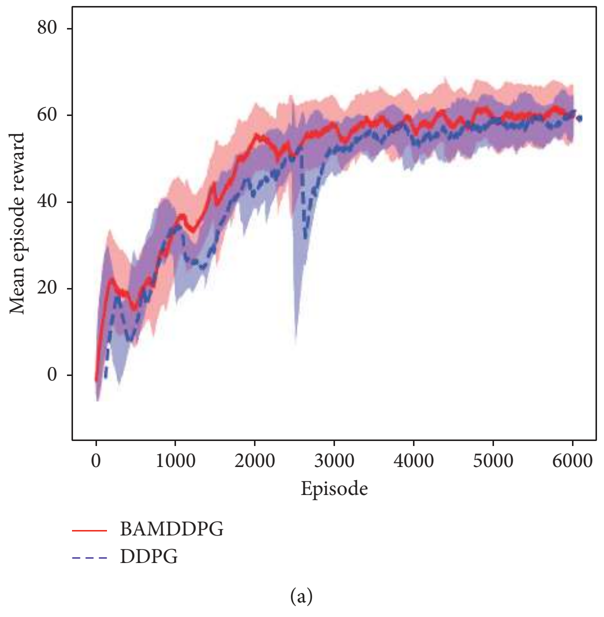
Figure 9 further illustrates the difference in total reward between subpolicies and the aggregated policy. As shown by the real line, the total reward of the aggregated policy is in a steady upward trend as the number of steps increases. However, the total reward of subpolicy 2 and subpolicy 3 increases steadily in the initial stage and then stops rising because the car pulled out of the track at some point. The performance of subpolicy 1 is the worst, and its total reward is always the lowest and ultimately remains unchanged due to the car leaving the track.

4.2.4. Effect from Number of Subpolicies. The final policy gained by BAMDDPG is based on the aggregation of subpolicies, but the algorithm does not give specific number of subpolicies. In theory, when there is large enough number of subpolicies, the aggregated policy successfully approximates the optimal policy. However, aggregating a large number of subpolicies is inefficient in consideration of computing and storage resource consumption in practice.

Under the consideration of balancing efficiency and performance, this section explores the appropriate number range of subpolicies through experiment. We choose the numbers of subpolicies within 30 and get the appropriate number of subpolicies by comparing the performance of the aggregated policies with different number of subpolicies. These aggregated policies are tested on the Aalborg track, and we then compare their training time, total reward within

Table 3: Description of action commands.

Commands	Range	Description
Steering	[−1, +1]	−1 is full right while +1 is full left
Brake	[0, 1]	Brake pedal (1 is full brake while 0 is no brake)
Throttle	[0, 1]	Gas pedal (1 is full gas while 0 is no gas)



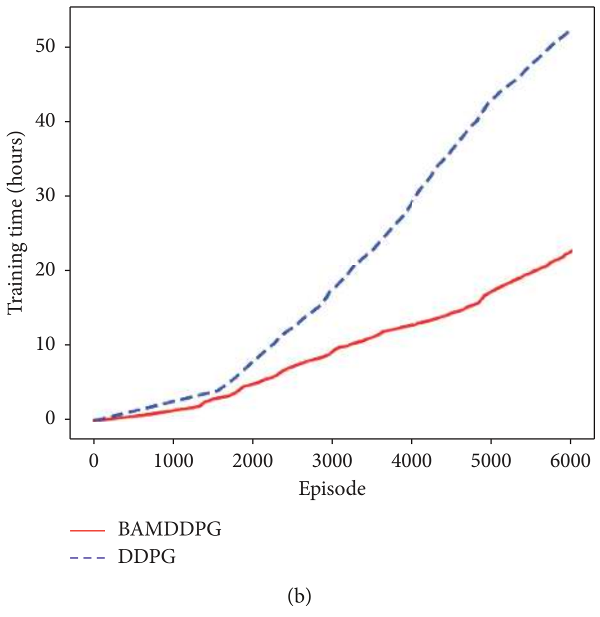


Figure 8: (a) Learning curve and (b) training time comparison of BAMDDPG and DDPG.

Table 4: Performance comparison of the aggregated policy and subpolicies.

Policy	Steps	Total reward (points)	Complete one lap
Subpolicy1	246	16690.60	No
Subpolicy2	246	15413.12	No
Subpolicy3	102	−1252.46	No
Aggregated policy	457	31603.37	Yes

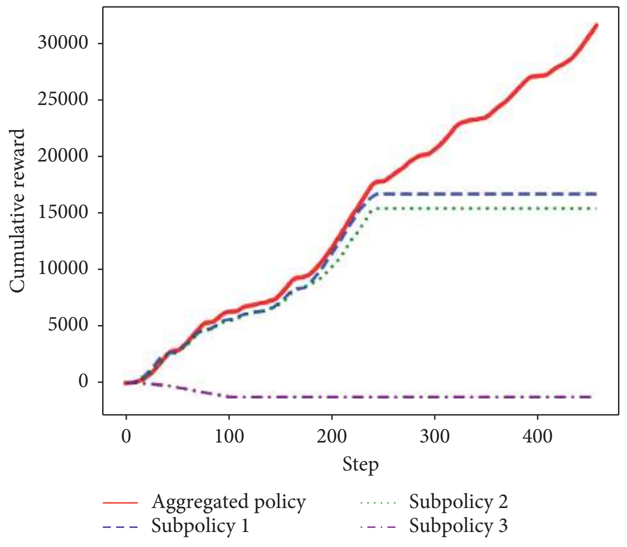


Figure 9: Performance comparison of the aggregated policy and subpolicies.

5000 steps. Furthermore, we compare the generalization performance of the aggregated policies by testing them on the CG1 track and CG2 track. Experimental results are demonstrated in Figure 10 and Table 5.

Figure 10 illustrates the comparison of total reward gained by aggregated policies with different number of subpolicies on the Aalborg track. Since the episode of TORCS may not terminate, we set the maximum number of steps to be 5000 in one episode. The aggregated policies with 3–10 subpolicies are able to reach the maximum number of steps while others terminate early in one episode. Therefore, they gained much larger reward than those aggregated policies with over 10 subpolicies.

Table 5 demonstrates, for policies aggregating from different numbers of subpolicies within 30, no large difference appears in training time, but the performances of different policies vary from each other. The policies aggregating from 3 to 10 subpolicies can achieve the maximum interaction number of 5000 steps on the Aalborg track, complete the training Aalborg track with larger total reward

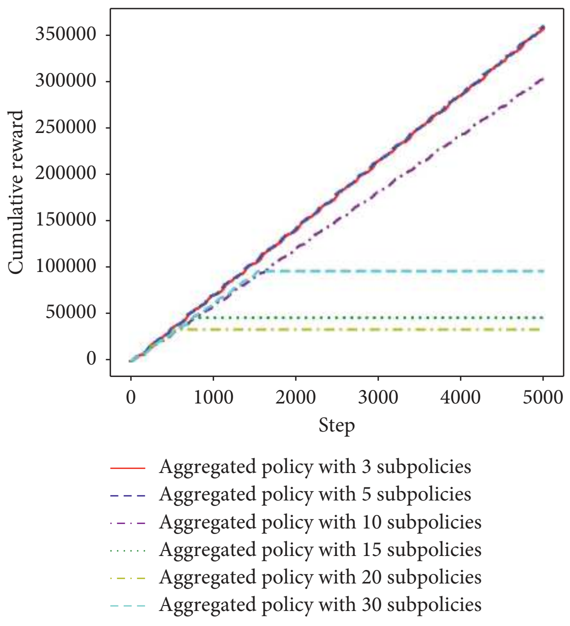


FIGURE 10: Reward comparison of aggregated policies with different numbers of subpolicies on Aalborg.

TABLE 5: Comparison of aggregated policies with different numbers of subpolicies.

Number of subpolicies	Training time (hours)	Total steps	Total reward	Pass Aalborg	Pass CG1	Pass CG2
3	22.84	5000	331086.10	Yes	Yes	Yes
5	24.40	5000	360804.43	Yes	Yes	Yes
10	24.16	5000	303678.65	Yes	Yes	Yes
15	22.09	771	47121.87	Yes	No	Yes
20	20.49	567	34343.05	Yes	No	Yes
30	21.74	1541	97146.37	Yes	No	Yes

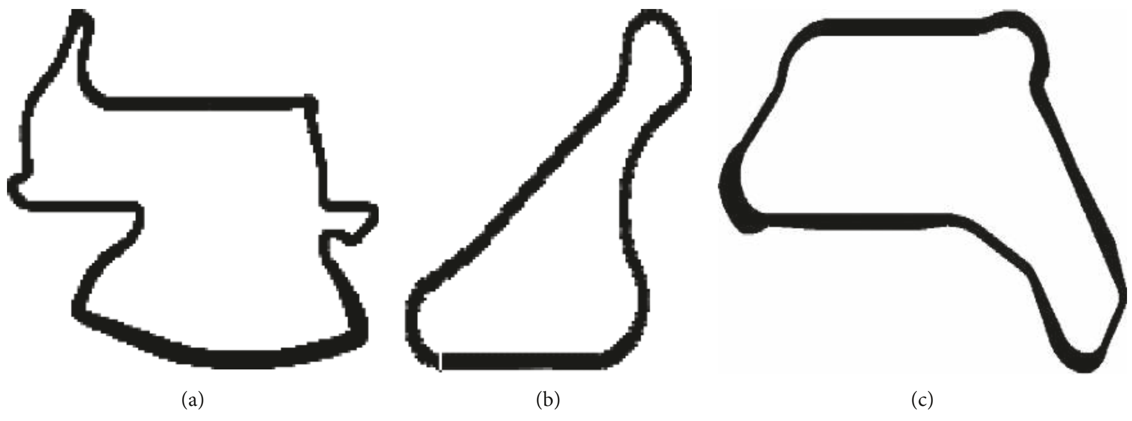


FIGURE 11: Maps of training and test tracks. (a) Aalborg; (b) CG1; (c) CG2.

than the aggregated policies with over 10 subpolicies, and pass the test track CG1 and CG2 safely.

Generally speaking, when the number of subpolicies is 3–10, the corresponding aggregated policies perform well and have better generalization performance than the aggregated policies with over 10 subpolicies, which means 3–10 is the appropriate number of subpolicies for BAMDDPG in practical application.

However, the aggregated policies with over 10 subpolicies cannot reach the maximum steps on the Aalborg track and are not able to finish the CG1 track. The reason why the aggregated policies with over 10 subpolicies performed worse mainly lies in the limit of the centralized experience replay buffer. During the training time, we fixed the size of the centralized experience replay buffer to 100, 000 transition tuples (s_t, a_t, r_t, s_{t+1}), by considering the

Table 6: Generalization performance of the aggregated policy.

Track name	Total reward (points)	Complete
Aalborg	30007.61	Yes
CG1	23755.62	Yes
CG2	35602.09	Yes

feasibility and efficiency of implementation. However, this buffer could not manage to share all experiences with more than 10 subpolicies. As a result, the aggregated policies with over 10 subpolicies gained less knowledge and performed not well. The experiment with a larger buffer size will display a better performance with aggregation of 10 subpolicies. But the memory setting has a nonmonotonic effect on the reinforcement learning (RL) performance [26]. The influence of the memory setting in RL arises from the trade-off between the correct weight update direction and the wrong direction.

4.2.5. Generalization Performance. Generalization performance is a research hotspot in the field of machine learning, and it is also a key evaluation index for the performance of algorithms. An overtrained model often performs well in the training set, while it performs poorly in the test set. In our experiments, self-driving policies are learned successfully on the Aalborg track using BAMDDPG. The car controlled by these policies has good performance on the training track. However, the generalization performance of the learned policies is not known. Hence, we test the performance of the aggregated policy learned with BAMDDPG on both the training and test tracks, including Aalborg, CG1, and CG2, whose maps are illustrated in Figure 11.

The total reward of the aggregated policy shown in Table 6 differs in different tracks since the length of different tracks is not the same. On a long track, the car travels for a longer time, and the total reward will be larger. In our experiment, route CG2 is the longest and CG1 is the shortest.

Table 6 illustrates that the car controlled by the aggregated policy passes the test tracks successfully. It demonstrates that the learned aggregated policy from BAMDDPG achieves a good generalization performance.

5. Conclusions

This paper proposed a deep reinforcement learning algorithm, by aggregating multiple deep deterministic policy gradient algorithm and an m-out-of-n bootstrap sampling method. This method is effective to the sequential and iterative training data, where the data exhibit long-tailed distribution, rather than the norm distribution implicated by the i.i.d. data assumption. The method can learn the optimal policies with much less training time for tasks with continuous space of actions and states.

Experiment results on the 2D robot arm game show that the reward gained by the aggregated policy is 10%~50% better than those gained by the nonaggregated subpolicies. Experiment results on TORCS demonstrate the proposed method can learn successful control policies with less training time by 56.7%, compared to the normal sampling method and nonaggregated subpolicies.

Data Availability

The program and data used to support the findings of this study are currently under embargo while the research findings are commercialized. Requests for data, 12 months after publication of this article, will be considered by the corresponding author. The simulation platform (The Open Racing Car Simulator, TORCS) used to support the findings of this study is open-sourced and is available at http://torcs.sourceforge.net/.

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 NSFC (61672512 and 51707191), CAS Key Laboratory of Human-Machine Intelligence-Synergy Systems, Shenzhen Institutes of Advanced Technology, and Shenzhen Engineering Laboratory for Autonomous Driving Technology.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT press, Cambridge, MA, USA, 1998.

[2] 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.

[3] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, pp. 1097–1105, Lake Tahoe, NV, USA, March 2012.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

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

[6] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-Learning," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, Phoenix, AZ USA, March 2016.

[7] T. Schaul, J. Quan, I. Antonoglou et al., "Prioritized experience replay," 2015, https://arxiv.org/abs/1511.05952.

[8] Z. Wang, T. Schaul, M. Hessel et al., "Dueling network architectures for deep reinforcement learning," in *Proceedings of the 33rd International Conference on Machine Learning*, vol. 4, pp. 2939–2947, New York, NY, USA, 2016.

[9] D. Silver, G. Lever, N. Heess et al., "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning*, pp. 387–395, Bejing, China, June 2014.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel et al., "Continuous control with deep reinforcement learning," *Computer Science*, vol. 8, no. 6, p. A187, 2015.

[11] Z. Zheng, C. Yuan, Z. Lin et al., "Self-adaptive double bootstrapped DDPG," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 3198–3204, Stockholm, Sweden, July 2018.

[12] S. Gu, T. Lillicrap, Z. Ghahramani et al., “Q-prop: sample-efficient policy gradient with an off-policy critic,” in *Proceedings of the International Conference on Learning Representations*, New Orleans, LA, USA, May 2017.

[13] V. Mnih, A. P. Badia, M. Mirza et al., “Asynchronous methods for deep reinforcement learning,” in *Proceedings of the International Conference on Machine Learning*, pp. 1928–1937, San Juan, PR, USA, May 2016.

[14] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*, CRC Press, Boca Raton, FL, USA, 1994.

[15] I. Osband, C. Blundell, A. Pritzel et al., “Deep exploration via bootstrapped DQN,” in *Proceedings of the Advances in Neural Information Processing Systems*, pp. 4026–4034, Barcelona, Spain, December 2016.

[16] Z. Yang, K. E. Merrick, H. A. Abbass et al., “Multi-task deep reinforcement learning for continuous action control,” in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pp. 3301–3307, Melbourne, Australia, August 2017.

[17] R. Lowe, Y. Wu, A. Tamar et al., “Multi-agent actor-critic for mixed cooperative-competitive environments,” *Advances in Neural Information Processing Systems*, 2017.

[18] W. Shi, S. Song, and C. Wu, “Soft policy gradient method for maximum entropy deep reinforcement learning,” 2019, https://arxiv.org/abs/1909.03198.

[19] R. Davidson and E. Flachaire, “Asymptotic and bootstrap inference for inequality and poverty measures,” *Journal of Econometrics*, vol. 141, no. 1, pp. 141–166, 2007.

[20] H. Ishwaran, L. F. James, and M. Zarepour, “An alternative to the out of bootstrap,” *Journal of Statistical Planning and Inference*, vol. 139, no. 3, pp. 788–801, 2009.

[21] J. Wu and H. Li, “Aggregated multi-deep deterministic policy gradient for self-driving policy,” in *Proceedings of 5th International Conference on Internet of Vehicles*, vol. 11253, pp. 179–192, Paris, France, November 2018.

[22] A. M. F. Mood, *Introduction to the Theory of Statistics*, McGraw-Hill Education, New York, NY, USA, 1950.

[23] A. Dvoretzky, J. Kiefer, and J. Wolfowitz, “Asymptotic minimax character of the sample distribution function and of the classical multinomial estimator,” *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 642–669, 1956.

[24] B. Wymann, E. Espié, C. Guionneau et al., *Torcs: The Open Racing Car Simulator*, SourceForge, 2015.

[25] D. Loiacono, L. Cardamone, and P. L. Lanzi, *Simulated Car Racing Championship: Competition Software Manual*, Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Milan, Italy, 2013.

[26] R. Liu and J. Zou, “The effects of memory replay in reinforcement learning,” in *Proceedings of the ICML 2017 Workshop on Principled Approaches to Deep Learning*, Sydney, Australia, 2017, https://arxiv.org/pdf/1710.06574.pdf.