Aggregated Multi-deep Deterministic Policy Gradient for Self-driving Policy

Junta Wu[1,2] and Huiyun Li[1,2](✉)

[1] Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518071, China
hy.li@siat.ac.cn

[2] Shenzhen College of Advanced Technology, University of Chinese Academy of Sciences, Shenzhen 518071, China

Abstract. Self-driving is a significant application of deep reinforcement learning. We present a deep reinforcement learning algorithm for control policies of self-driving vehicles. This method aggregates multiple sub-policies based on the deep deterministic policy gradient algorithm and centralized experience replays. The aggregated policy converges to the optimal policy by aggregating those sub-policies. It helps reduce the training time largely since each sub-policy is trained with less time. Experimental results on the open racing car simulator platform demonstrates that the proposed algorithm is able to successfully learn control policies, with a good generalization performance. This method outperforms the deep deterministic policy gradient algorithm with 56.7% less training time.

Keywords: Self-driving · Deep reinforcement learning · Deep deterministic policy gradient

1 Introduction

Self-driving vehicles capable of sensing environment and navigating intelligently attract enormous interest in both research and industry area [1]. The two main foundation stones behind the self-driving vehicles are perception and decision. The former enables the vehicles to percept the world, whilst the latter yields the control policies. Generally, control policies giving intelligent action commands (such as steering, acceleration and brake command) are essential for self-driving vehicles after they get the environment information from various sensors. However, how to learn optimal control policies is a challenge for self-driving vehicles.

Recently, with the accessibility of large amount of training data and computation resources, interest of applying machine learning techniques to self-driving vehicles has grown explosively. Particularly, the success of deep neural network (DNN) technique [2, 3], raises a new upsurge of study on self-driving vehicles. The DNN has been applied onto an end-to-end learning system for self-driving vehicles, which maps raw pixels from camera directly to steering commands [4].

Reinforcement learning [5] whose essence is learning through interaction [6] is an active branch of machine learning and had some success for solving challenging

https://doi.org/10.1007/978-3-030-05081-8_13

problems in control theory. As DNN was developed, reinforcement learning with DNN, i.e., deep reinforcement learning algorithms was proposed in [7]. The so-called deep Q-network (DQN) [7] is the first deep reinforcement learning algorithm to successfully learn an optimal control policy for an autonomous braking system [8]. There are also other deep reinforcement learning algorithms inspired by DQN are used in the field of self-driving vehicles. For example, deep Q-learning with filtered experiences (DQFE) algorithm [9], which extended experience replay technique, successfully learned a steering policy on the open racing car simulator (TORCS) [10, 11].

However, DQN is only applicable for the tasks with discrete and low-dimensional action spaces. One has to discretize action spaces to apply DQN onto self-driving vehicles, where action space is continuous. But this may result in the unstable control of vehicles. In order to solve this problem, researchers proposed deep deterministic policy gradient (DDPG) algorithm to directly learn continuous control policy [12]. Unfortunately, DDPG usually demands long training time to find an optimal policy.

Aggregation method has been proved to be successful in reinforcement learning [13–15]. In this paper, we propose a deep reinforcement learning algorithm with aggregation method to reduce the training time, which is able to learn multiple sub-policies at the same time by utilizing centralized experience replay technique and output the final control policy by aggregating all the sub-policies.

The remainder of this paper is organized as follows. Section 2 provides background knowledge. Section 3 describes details of the proposed method. Experimental results of the proposed method are presented in Sect. 4 and analysis of aggregation is provided in Sect. 5. Finally, the paper is concluded in Sect. 6.

2 Background

2.1 Reinforcement Learning

In a standard reinforcement learning setup, an agent interacts with environment ε at discrete time steps. In each time step t, the agent observes the state s_t and takes the action a_t according to its policy π which maps a state to a deterministic action or a probability distribution over the actions. Then it receives an immediate reward $r(s_t, a_t)$. The accumulated reward the agent can get from a specific time step t is defined as:

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

where $\gamma \in [0, 1]$ is a discount factor. The goal in a reinforcement learning task is to learn an optimal policy π^* by maximizing the expected accumulated reward from the start state, which is $\mathrm{E}[R_t]$.

2.2 Q-Learning and DQN

Q-Learning. Q-Learning [5] is an important branch of reinforcement learning. Q refers to the action-value function [5] $Q^{\pi}(s_t, a_t) = \mathrm{E}_{\pi}[R_t|s_t, a_t]$, which describes the expected accumulated reward after taking action a_t in state s_t by following the policy π. Generally, the action value function can be expressed as recursive form known as Bellman equation [7]:

$$Q^{\pi}(s_t, a_t) = \mathrm{E}_{s_{t+1}-\varepsilon, a_{t+1}-\pi}[r(s_t, a_t) + \gamma \mathrm{E}_{\pi}[Q^{\pi}(s_{t+1}, a_{t+1})]]. \tag{2}$$

Q-Learning makes use of temporal-difference learning method and greedy policy to iteratively update Q function [5]:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r(s_t, a_t) + \gamma \max_{a_t} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]. \tag{3}$$

DQN. DQN [7] combines deep neural network technique with Q-learning. The Q function is approximated with a deep neural network and converges to the optimal Q function Q^* by minimizing the following loss function:

$$L(\theta) = \mathrm{E}_{s_t, a_t - \rho(\bullet)}\left[(Q^*(s_t, a_t|\omega) - y_t)^2\right], \tag{4}$$

where $\rho(\bullet)$ is the behavior distribution, $y_t = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}|\omega^{'})$ is the target and the parameter $\omega^{'}$ is duplicated from the previous iteration.

Besides deep neural network technique, another key to the success of DQN is experience replay technique, which breaks the strong correlations of consecutive samples and therefore reduces the variance of updates [7].

2.3 Policy Gradient Algorithms and Deterministic Policy Gradient Algorithm

Policy Gradient Algorithms. Policy gradient algorithms derived from the policy gradient theorem [5] are used for learning stochastic policies $\pi_{\theta} : S \mapsto \mathrm{P}(\mathrm{A})$, where S, A represent state space and action space respectively and P(A) is the set of probability measures on A. They are strongly suited for reinforcement learning tasks with continuous action spaces. The basic idea behind the policy gradient algorithms is to maximize the performance $J(\theta) = \mathrm{E}_{s \sim \rho^{\pi}, a \sim \pi_{\theta}}[R]$, by updating the parameter along the policy gradient, i.e. the gradient of the policy's performance:

$$\nabla_{\theta} J(\theta) = \mathrm{E}_{s \sim \rho^{\pi}, a \sim \pi_{\theta}}[\nabla_{\theta} \log \pi_{\theta}(a|s) Q^{\pi}(s, a)], \tag{5}$$

where ρ^{π} is the state distribution and Q^{π} is the true action value function. Different policy gradient algorithms differ in how they estimate Q^{π}. Alternatively, approximating Q^{π} with the temporal-difference learning method [5] leads to the actor-critic (AC) algorithm [5].

Deterministic Policy Gradient Algorithm. Derived from deterministic policy gradient theorem [16], the deterministic policy gradient algorithm is used to learn deterministic policies [16] $\mu_{\theta} : S \mapsto A$ instead of stochastic policies. The deterministic policy gradient algorithm maximizes the performance function in the direction of deterministic policy gradient [16], which is:

$$\nabla_{\theta} J(\theta) = \mathrm{E}_{s \sim \rho^{\mu}, a \sim \mu_{\theta}} \left[\nabla_{\theta} \mu_{\theta}(s) \nabla_{a} Q^{\mu}(s, a) \big|_{a=\mu_{\theta}(s)} \right]. \tag{6}$$

Deterministic policy gradient algorithm also utilizes actor-critic (AC) framework where action value function Q^{μ} is used as a critic and the deterministic policy μ_{θ} is considered as an actor.

2.4 DDPG Algorithm

DDPG is a variant of deterministic policy gradient algorithm [12], which adopts deep neural network to approximate deterministic policy μ and action value function Q^{μ}. The gradient of the objective can be formulated as:

$$\nabla_{\theta} J(\theta) = \mathrm{E}_{s \sim \rho^{\mu}, a \sim \mu} \left[\nabla_{\theta} \mu(s|\theta) \nabla_{a} Q^{\mu}(s, a|\omega) \big|_{a=\mu_{\theta}(s)} \right], \tag{7}$$

where θ is parameter of actor network corresponding to the policy and ω is the parameter of critic network corresponding the action value function. The diagram of DDPG is shown in Fig. 1:

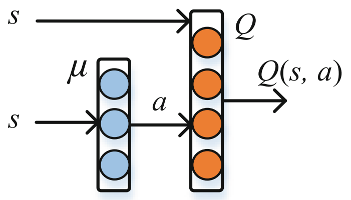


Fig. 1. Diagram of deep deterministic policy gradient

The actor network updated according to the chain rule and back-propagation of neural network eventually converges to the optimal policy which maximizes the overall performance. DDPG is an off-policy algorithm, utilizing the experience replay technique introduced in DQN to break the correlation of the samples and to keep samples independent identically distributed. In addition, the learning method of Q function is similar to that in DQN as well.

2.5 TORCS

TORCS is the competition software for the simulated car racing championship, which takes the client-server structure [11]. Figure 2 presents the client-server structure of TORCS. In TORCS, a controller of a car perceives the racing environment through a number of sensors to get the current state. Then it can perform the typical driving actions, such as accelerating, braking and steering the wheel, by following a specific control policy. The controller connects to the race server through UDP connection. In every time step, the server sends the current sensor information to the controller and waits for 10 ms to receive an action from the controller. If no action is received, the last performed action is taken.

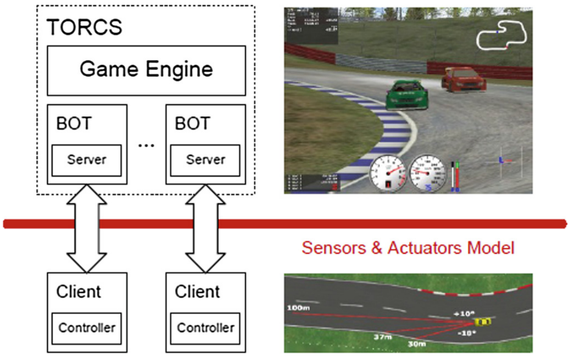


Fig. 2. Server-client structure of TORCS

Since TORCS simulates real driving environment well and its sensor information of the simulated car can be easily accessed, it is a suitable platform for the research of self-driving vehicles. Additionally, the interactive mechanism between the server (driving environment) and client (controller of a car) makes TORCS an excellent benchmark for deep reinforcement learning algorithm.

3 Proposed Method

3.1 Structure of Multi-DDPG

We have explained earlier that DDPG is more suitable for continuous control tasks than DQN. However, it takes long time to learn the optimal policy. In order to reduce the training time, we proposed an aggregated method with multi-DDPG structure. More specifically, we use multi-DDPG structure to learn several sub-policies simultaneously, which takes less training time. The structure of multi-DDPG is shown in Fig. 3. Then we achieve the aggregated policy by averaging the outputs of all the sub-policies, as shown in Fig. 4.

During training time, each sub-policy is learned with DDPG which utilizes AC framework. In each time step, the agent receives the environment state and acts according to one sub-policy. Then the performance of the action is evaluated by the corresponding Q network. By following the updating rule of DDPG, each sub-policy is improved gradually. Experiences encountered by all sub-policies are to be stored in the centralized experience replay buffer. We will go into details of centralized experience replay buffer in Sect. 3.2.

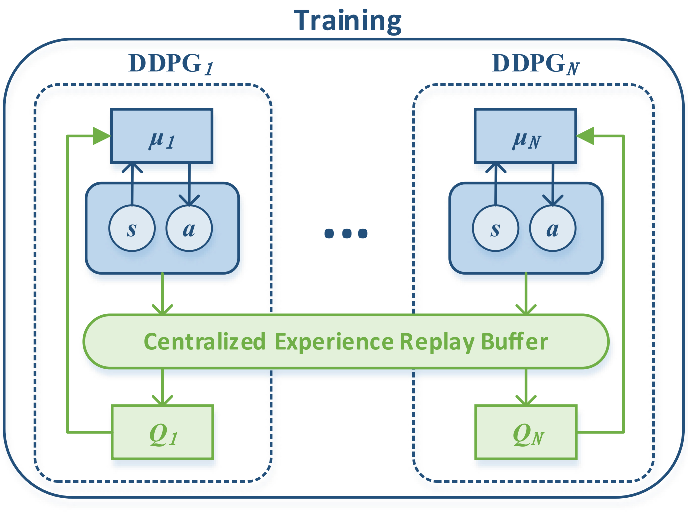


Fig. 3. Structure of multi-DDPG

We illustrate the aggregation of sub-policies in Fig. 4 where $\mu_1, \cdots, \mu_N$ are sub-polices, $a_1, \cdots, a_N$ are the corresponding outputs of the sub-policies and a is the output of the aggregated policy. After training, we get several different sub-policies. Since the sub-policies take less time to train than the optimal policy, each sub-policy performs worse than the optimal policy. However, we are able to improve the performance by aggregating these sub-policies to achieve the optimal policy. Considering that sub-policies are of equal importance and their outputs are real valued, the aggregation method we used in this paper is to average the outputs of the sub-policies. We name the above method as aggregated multi-DDPG (AMDDPG) since it bases on the aggregation of several sub-policies learned from multi-DDPG structure.

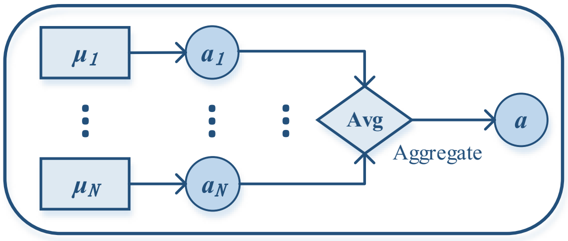


Fig. 4. Aggregation of sub-policies

3.2 Centralized Experience Replay Buffer

As introduced in Sect. 2.4, DDPG is an off-policy algorithm, which means it is capable of learning one policy through the experiences from other policies. Additionally, DDPG utilizes the experience replay technique to break the correlation of the samples and to keep samples independent identically distributed. Under such condition, it seems a natural idea to use a centralized experience replay buffer for AMDDPG. Specifically, during training time, each DDPG keeps the same experience replay buffer.

In a specific time step, the agent interacts with the environment, receiving environment state, acting according to one sub-policy and getting an immediate reward. We then store all state, action and reward information in the centralized experience replay buffer as the experiences the agent experienced. Basically, the experiences in centralized experience replay buffer are from all the sub-policies.

One intuitive insight to the centralized experience replay buffer is that the agent is able to utilize the experiences from multiple policies, which makes the agent broaden its view and have more knowledge of the environment. In this way, the agent can learn multiple sub-policies more efficiently.

3.3 Training Scheme and Algorithm

Multiple sub-policies are learned simultaneously with AMDDPG, but in a specific time step, the environment allows the agent use only one sub-policy to interact with it. Therefore, the training scheme we proposed in this paper is to alternately train multiple sub-policies. More specifically, we train one sub-policy in each episode. The full algorithm of AMDDPG is presented in Table 1.

Table 1. Aggregated multi-DDPG algorithm

```
Algorithm 1 Aggregated Multi-DDPG
  Randomly initialize $N$ critic networks $Q_i(s,a|\theta_i^Q)$ and actor networks $\mu_i(s|\theta_i^\mu)$
  Initialize centralized experience replay buffer $R$
  for episode = 1, M do
    Receive initial environment state $s_1$
    for t = 1, T do
      Select $Q_j(s,a|\theta_j^Q)$ and $\mu_j(s|\theta_j^\mu)$ according to the training scheme
      Execute action $a_t = \mu_j\left(s_t|\theta_j^\mu\right)$ and observe reward $r_t$ and new state $s_{t+1}$
      Store experience $\left(s_t, a_t, r_t, s_{t+1}\right)$ in $R$
      Update $\theta_j^Q$ and $\theta_j^\mu$ according to DDPG algorithm based on experiences
    end for
end for
```

4 Experimental Results

4.1 Setup

Reward Function. One key to learn a good policy for reinforcement learning task is to design an appropriate reward function. In this section, we discuss how to design the reward function in TORCS for AMDDPG.

In TORCS, one agent, i.e., a simulated car, can percept the environment through various sensors. Some of the sensor information are perfect for designing the reward function to guide the agent. We use the sensor information in Table 2 to construct the reward function.

Table 2. Sensor information for constructing the reward function

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

The reward function is presented in Eq. (8). In short, the performances we expect should be reflected through the terms in the reward function.

In each interaction between the car and TORCS environment, we expect the reward, r, to be as large as possible.

$$r = \left(v \times \left(\frac{1 - |v - \beta|}{\alpha} \right)^{\mathbf{I}[d_1 \leq 10]} \right) \times \cos\varphi \times (1 - |\sin\varphi|) \times (1 - |d_2|) \times \left(\frac{|d_1|}{50} \right)^{\mathbf{I}[d_1 \leq 50]}. \tag{8}$$

The term v indicates that, in order to maximize the reward, the car has to run along the track fast. In Eq. (8), the $\cos\varphi$ and $(1 - |\sin\varphi|)$ terms expects φ to be zero, which means the car runs along the track. The $(1 - |d_2|)$ term keeps the car in the center of the track. $\mathbf{I}[\![\bullet]\!]$ is an indicator function, which outputs 1 when the condition is satisfied, otherwise, 0. Therefore, the first term and last term of Eq. (8) can be reformulated as Eqs. (9) and (10).

$$v \times \left(\frac{1 - |v - \beta|}{\alpha} \right)^{\mathbf{I}[d_1 \leq 10]} = \begin{cases} v \times \left(\frac{1 - |v - \beta|}{\alpha} \right), & d_1 \leq 10 \\ v, & d_1 > 10 \end{cases}. \tag{9}$$

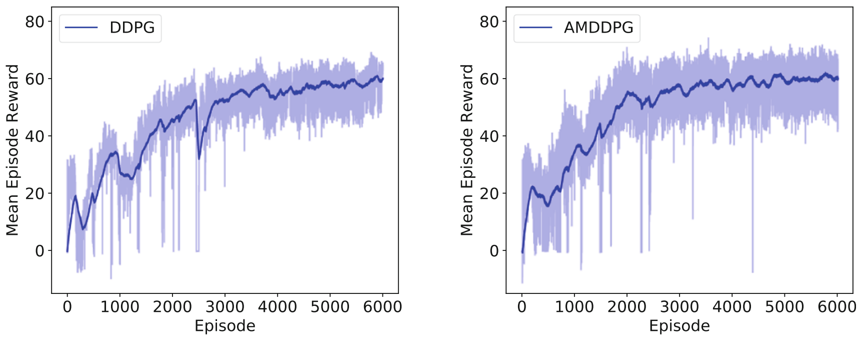
$$\left(\frac{|d_1|}{50}\right)^{\mathbf{I}[d_1 \leq 50]} = \begin{cases} \frac{|d_1|}{50}, & d_1 \leq 50 \\ 1, & d_1 > 50 \end{cases}. \tag{10}$$

The first term in Eq. (8) lets the car slow down when a turn is encountered and makes the car run as fast as possible at the straight. Besides, it is important to note that α and β are hyper parameters. They need to be fine-tuned to ensure the speed at turns is appropriate. The last term of Eq. (8) indicates that the car should observe the turn in advance and control steering angles.

Training. We trained a simulated car in TORCS to learn several self-driving policies with AMDDPG and get the optimal policy by aggregating those policies. During training process, the car received the sensor information as the environment state and executed the action given by one sub-policy. The action consists of the steering, accelerating and braking commands, whose detailed information is presented in Table 3. We have argued in Sect. 3 that AMDDPG demands less training than DDPG. Figure 6 illustrates the training time comparison between AMDDPG and DDPG.

Table 3. Commands description within one action

Commands	Range	Description
Steering	[−1, +1]	−1 and +1 mean respectively full right and full left
Acceleration	[0, 1]	Virtual gas pedal (0 means no gas, 1 means full gas)
Brake	[0, 1]	Virtual brake pedal (0 means no brake, 1 means full brake)



(a) Learning curve of DDPG (b) Learning curve of AMDDPG

Fig. 5. Learning curve of DDPG and AMDDPG

We have trained the car 6000 episodes on Aalborg track of TORCS. The learning curve of DDPG is demonstrated in Fig. 5(a) and that of AMDDPG is presented in Fig. 5(b). As Fig. 5 shows, after 6000 episodes, the policy of DDPG and the policy of AMDDPG both converge to and oscillate around the optimal one. Figure 6 demonstrates that AMDDPG takes less time to train since each sub-policy is trained with less iterations. The training time of AMDDPG for 6000 episodes is 22.84 h while that of DDPG is 52.77 h, showing that the AMDDPG is able to reduce the training time by 56.7%. However, the difference during the first 1500 episodes is not notable enough. It is because the car paid more attention on exploring the environment at the beginning and exploration episodes do not last long. The AMDDPG and DDPG take nearly equal episodes and time to explore the new environment. Alternatively, the first 1500 episodes can be considered as the initialization.

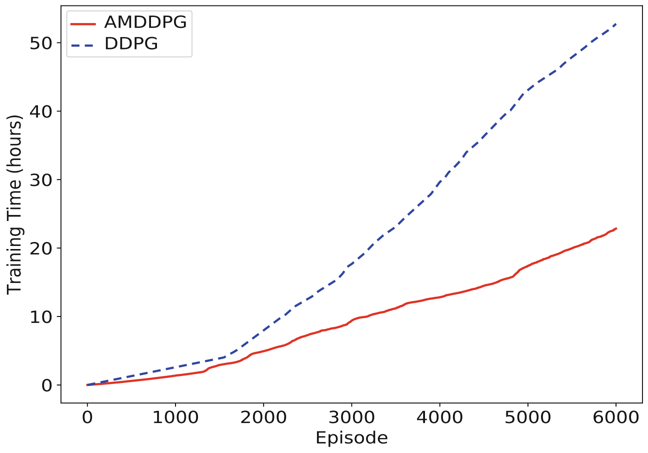


Fig. 6. Comparison of training time between AMDDPG and DDPG

4.2 Performance Test

Effectiveness of Aggregation. We show in this subsection the comparison of performance between the aggregated policy and sub-policies so as to illustrate the effectiveness of aggregation. For the sake of simplicity, we only show the performance of three sub-policies. We let the car run one lap on Aalborg track to test the policies. As Table 4 shows, the cars with sub-policies are not able to finish any lap of Aalborg track while the car with the aggregated policy can pass the track. This fully testify that aggregation does improve the performance of sub-policies.

Table 4. Comparison of performance between sub-policies and aggregated policy

Policy	Steps	Total reward (points)	Finish one lap or not
Sub-policy0	246	16690.60	No
Sub-policy1	246	15413.12	No
Sub-policy2	102	−1252.46	No
Aggregated policy	457	31603.37	Yes

Figure 7 presents the comparison of reward between the aggregated policy and sub-policies. Since the cars with sub-policies run out from the track before finish one lap, the last half of their reward curves is flat.

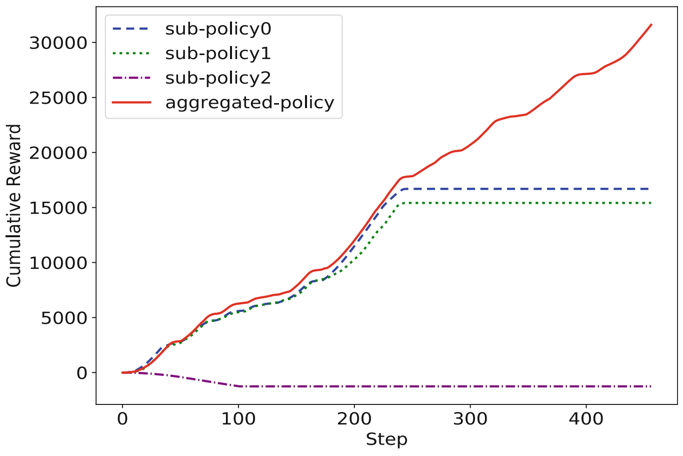


Fig. 7. Comparison of performance between the aggregated policy and sub-policies

Generalization Performance. We have learned self-driving policies on Aalborg track with AMDDPG successfully and the car performed well on the training track. However, in machine learning, one model might overfit the training dataset and perform worse on test dataset. Therefore, we test the aggregated policy learned by AMDDPG not only on the training track, Aalborg, but also on the test tracks, CG1 and CG2. Figure 8 illustrates the maps of the tracks for training and test.

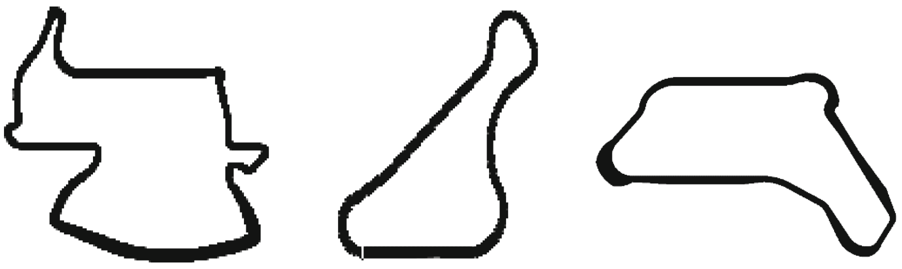


Fig. 8. Maps of tracks for training and test

As Table 5 shows. The car passed the training track and the test tracks, which demonstrates the aggregated policy learned by AMDDPG does not overfit the training track and have a good generalization performance on the test tracks.

Table 5. Generalization performance of aggregated policy

Track name	Total reward (points)	Finish one lap or not
Aalborg	30007.61	Yes
CG1	23755.62	Yes
CG2	35602.09	Yes

Effect from Number of Sub-policies. Theoretically, the aggregated policy converges to the optimal policy when the number of sub-policies is large enough. In practice, it is inefficient to aggregate a large number of sub-policies. Additionally, the larger the number of sub-policies, the more the memory overhead would be. Therefore, we choose some small numbers for sub-policies and compare the performance to decide the suitable number of sub-policies. We test the aggregated policies on Aalborg track and compare the total steps and total reward in one episode. In addition, we test the aggregated policies on the track CG1 and CG2 as well to compare generalization performance.

As Table 6 shows, there is not large difference in training time for different numbers but the performances differ largely. The aggregated policy with 3–10 sub-policies performs better. With 3–10 sub-policies, the total steps in one episode of Aalborg can reach the maximum steps and the car is able to gain much more reward than the car with more than 10 sub-policies. Besides, the car with 3–10 sub-policies can pass not only Aalborg track but also CG1 and CG2 track, which means the aggregated policy with 3–10 sub-policies has better generalization performance.

Table 6. Comparison of performance of aggregated policies with different numbers of sub-policies

Number of sub-policies	Training time (hours)	Total steps in Aalborg	Total reward in Aalborg	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

5 Analysis

In this section, we aim to explain why the aggregated policy performs better than sub-policies. Assume the environment state is s in a specific time step. $\mu(s)$ is the action given by the policy μ. Suppose N sub-policies are trained in AMDDPG, and each sub-policy is denoted as $\mu_i (i = 1, 2, \cdots N)$. In Addition, the optimal policy to which the aggregated policy converges is denoted as μ^*. For simplicity, the policy output at state s is represented as μ in the following formulas instead of $\mu(s)$. The aggregated policy $\bar{\mu}$ can be formulated as:

$$\bar{\mu} = \mathbf{Avg}[\mu_i] = \frac{1}{N}\sum_{i=1}^{N} \mu_i, \tag{11}$$

where $\mathbf{Avg}[\mu_i]$ is the average of sub-policies. Suppose the average bias between the sub-policies and the optimal policy is $\mathbf{Avg}[(\mu_i - \mu^*)^2]$, and the bias between the aggregated policy and the optimal policy is $(\bar{\mu} - \mu^*)^2$. We establish the relation between $\mathbf{Avg}[(\mu_i - \mu^*)^2]$ and $(\bar{\mu} - \mu^*)^2$ in Eq. (12):

$$\begin{aligned}
\mathbf{Avg}\left[(\mu_i - \mu^*)^2\right] &= \mathbf{Avg}\left[\mu_i^2 - 2\mu_i\mu^* + (\mu^*)^2\right] \\
&= \mathbf{Avg}\left[\mu_i^2\right] - 2\mu^*\mathbf{Avg}[\mu_i] + (\mu^*)^2 \\
&= \mathbf{Avg}\left[\mu_i^2\right] - 2\mu^*\bar{\mu} + (\mu^*)^2 \\
&= \mathbf{Avg}\left[\mu_i^2\right] - 2\bar{\mu}^2 + \bar{\mu}^2 + \left(\bar{\mu}^2 - 2\mu^*\bar{\mu} + (\mu^*)^2\right), \\
&= \mathbf{Avg}\left[\mu_i^2 - 2\mu_i\bar{\mu} + \bar{\mu}^2\right] + (\bar{\mu} - \mu^*)^2 \\
&= \mathbf{Avg}\left[(\mu_i - \bar{\mu})^2\right] + (\bar{\mu} - \mu^*)^2 \\
&= \mathbf{Var}(\mu_i) + (\bar{\mu} - \mu^*)^2 \\
&\geq (\bar{\mu} - \mu^*)^2
\end{aligned} \tag{12}$$

where $\mathbf{Var}(\mu_i)$ represents the variance of sub-policies. Equation (12) demonstrates the bias between the aggregated policy and the optimal policy is smaller than the average bias between the sub-policies and the optimal policy. Therefore the aggregated policy is closer to the optimal policy than sub-policies and performs better. As Eq. (12) shows, the variance of sub-policies is reduced when using the aggregated policy as the final policy and the aggregated policy is closer to the optimal policy.

6 Conclusion

This paper presented a new deep reinforcement learning algorithm, aggregated multi-deep deterministic policy gradient (AMDDPG), to find the optimal control policies for self-driving vehicles. We demonstrated AMDDPG improved the performance of sub-policies by aggregating them. The method reduced the training time by 56.7%. We also illustrated the aggregated policy generalizing well on other test tracks of TORCS. Finally, we investigated the effect from number of sub-policies.

However, a few limitations remain to our approach. One is unstable learning due to the high variance of DDPG algorithm. This leads the final policy to oscillate around the optimal policy. We leave this investigation to the future work.

Acknowledgment. This work was supported by Shenzhen Engineering laboratory on Autonomous Vehicles, NSFC 61672512, and the Shenzhen S&T Funding with Grant No. JCYJ201605 10154531467.

References

1. Urmson, C.: Self-driving cars and the urban challenge. IEEE Intell. Syst. **23**(2), 66–68 (2008)
2. Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. In: NIPS 2012 Proceedings of the 25th International Conference on Neural Information Processing Systems, pp. 1097–1105. Curran Associates Inc., New York (2012)
3. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**, 436–444 (2015)
4. Bojarski, M., Del Testa, D., Dworakowski, D., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
5. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
6. Arulkumaran, K., Deisenroth, M.P., Brundage, M., et al.: Deep reinforcement learning: a brief survey. IEEE Signal Process. Mag. **34**(6), 26–38 (2017)
7. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)
8. Chae, H., Kang, C.M., Kim, B.D., et al.: Autonomous braking system via deep reinforcement learning. arXiv preprint arXiv:1702.02302 (2017)
9. Xia, W., Li, H.Y.: Training method of automatic driving strategy based on deep reinforcement learning. J. Integr. Technol. **6**(3), 29–40 (2017)
10. Wymann, B., Espié, E., Guionneau, C., et al.: TORCS: the open racing car simulator (2015)
11. Loiacono, D., Cardamone, L., Lanzi, P.L.: Simulated car racing championship: competition software manual. Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Italy (2013)
12. Lillicrap, T.P., Hunt, J.J., Pritzel, A., et al.: Continuous control with deep reinforcement learning. In: ICLR (2016)
13. Jiang, J.: A framework for aggregation of multiple reinforcement learning algorithms. Dissertation, University of Waterloo, Waterloo, Ontario, Canada (2007)
14. Jiang, J., Kamel, M.S.: Aggregation of reinforcement learning algorithms. In: The 2006 IEEE International Joint Conference on Neural Networks, pp. 68–72. IEEE, Vancouver (2006)
15. Lowe, R., Wu, Y., Tamar, A., et al.: Multi-agent actor-critic for mixed cooperative-competitive environments. In: Advances in Neural Information Processing Systems (2017)
16. Silver, D., Lever, G., Heess, N., et al.: Deterministic policy gradient algorithms. In: ICML 2014 Proceedings of the 31st International Conference on International Conference on Machine Learning, pp. I-387–I-395. ICML, Beijing (2014)