Proceedings of the 2017 IEEE
International Conference on Robotics and Biomimetics
December 5-8, 2017, Macau SAR, China

Continuous Reinforcement Learning From Human Demonstrations With Integrated Experience Replay for Autonomous Driving

Sixiang Zuo[1], Zhiyang Wang[2], Xiaorui Zhu[1,*], and Yongsheng Ou[2,*]

***Abstract*— As a promising subfield of machine learning, Reinforcement Learning (RL) has drawn increasing attention among the academia as well as the public. However, the practical application of RL is still restricted by a variety of reasons. The two most significant challenges of RL are the large exploration domain and the difficulty to converge. Integrating RL with human expertise is technically an interesting way to accelerate the exploration and increase the stability. In this work, we propose a continuous reinforcement learning method which integrates Deep Deterministic Policy Gradient (DDPG) with human demonstrations. The proposed method uses a combined loss function for updating the actor and critic networks. In addition, the experience replay buffer is also drawn from different transition data samples to make the learning more stable. The proposed method is tested with a popular RL task, i.e. the autonomous driving, by simulations with TORCS environment. Experimental results not only show the effectiveness of our method in improving the learning stability, but also manifest the potential capability of our method in mastering human preferences.**

I. Introduction

Reinforcement Learning (RL) has been steered onto the track of a rapid development since the last several decades. The victories in the challenge matches between the deep reinforcement learning program AlphaGo [1] and the world Go champions, Ke Jie and Lee Se-do, made RL at the forefront of world attention. As a highly intelligent machine learning methodology, RL does not rely on specific supervision, but purely depends on learning policies from scratch by exploring the environment. Even though an increasing number of application areas have witnessed the crucial role of RL, including in robot manipulation [2], mobile robot navigation [3], human healthcare [4] etc., most RL algorithms are still in the primary stage and show very limited prospects in real-world practical tasks for various reasons.

The first challenge for RL is the large scale of the state and action space, which makes learning the value of each state very slow. Although many learning algorithms have been used as function approximators for estimating the value function, the high dimensionality and continuousness of the state space is also problematic. In [5], by learning a more compact and abstract space with Gaussian Process Latent Variable Model (GPLVM), the states are mapped into a latent space, which largely reduces the dimension of the state space and thus benefits learning. Furthermore, the collision of RL and deep learning gave birth to the great success of deep reinforcement learning (DRL), which considerably addresses the issue. By learning a deep convolutional neural network to approximate the Q-function, Mnih et al. successfully construct a DRL framework called Deep Q-Network (DQN) which plays Atari games in human level [6], [7]. DQN differs from the traditional Q-learning not only in the deep structure for estimating the Q-function, but also in two significant improvements, i.e. the Experience Replay and Target Network. In [8], an extension called Deep Deterministic Policy Gradient (DDPG) is proposed for continuous action spaces. DDPG preserves Experience Replay and Target Network from DQN but uses the actor-critic mechanism to simultaneously update a policy network and a value network thus generates continuous and deterministic actions.

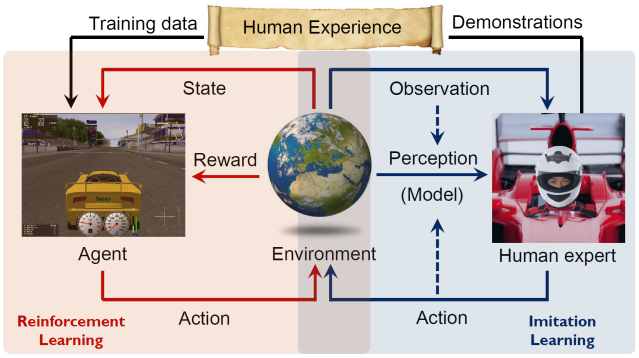


Fig. 1. The framework of Reinforcement Learning, Imitation Learning, and their integration. The demonstrations of human experts can be used to pretrain a model for RL, to shape the reward/policy of RL, and so on.

Another problem for RL is the difficulty of convergence. On the one hand, to converge to a considerably optimal policy, enormous explorations are always required. In most practical cases, to find an optimal policy with a large number of iterations by trial and error is unacceptable. On the other hand, the evaluation of the behavior of the agent during exploration is also critical. Since RL algorithms only update the policy according to the reward, the definition of the reward function is of great concern. Unfortunately, in real-world tasks, the reward is always implicit, and also hard to be defined manually. Introducing human expertise is intuitively an efficient way not only to speedup convergence, but also to provide a better guidance for evaluation. Imitation

[1]Sixiang Zuo and Xiaorui Zhu are with Harbin Institute of Technology. 150001 Harbin, P.R.China
[2]Zhiyang Wang and Yongsheng Ou are with Key Laboratory of Human-Machine Intelligence-Synergy Systems, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. 518055 Shenzhen, P.R.China
*Xiaorui Zhu is the corresponding author, and her mail address is (xiaoruizhu@hit.edu.cn).
*Yongsheng Ou is also the corresponding author, and his mail address is (ys.ou@siat.ac.cn).

978-1-5386-3741-8/17/$31.00 © 2017 IEEE

Learning or Learning from Demonstration (LfD) [9] provides a straightforward framework for the agent to master flexible policies with human guidance. A popular algorithm called DAGGER [10] provides a easy way for incorporating human experience. However, a human expert should always be available to provide feedback which is not convenient in practice. Finn et al. proposed the Inverse Reinforcement Learning (IRL), which automatically learns reward functions from human demonstrations for robotic control [11]. Since it does not provide a mechanism to obtain feedback from the interaction with the environment, the generalization ability of imitation learning is limited.

To combine RL and imitation learning, studies have been conducted on reward shaping [12], [13], policy shaping [14] and knowledge transfer [15] with human demonstrations(Fig. 1). Presented in [16], Deep Q-learning from Demonstrations (DQfD) vastly accelerates DQN by pretraining an initial behavior network, and also introducing a supervised loss and a L2 regularization loss when training the target network. The results of DQfD outperforms Double DQN and pure imitation learning in average rewards on 27 and 31 of 42 Atari games respectively. In [17] the reward is from a reward predictor trained with human feedback rather than from interactions with the environment. This method separates the goal learning from the behavior learning by training a reward predictor with non-expert human preferences, and the behavior of the agent is improved accordingly. However, this method performs poorly in some experiments, and has to be improved to satisfy large scale practical tasks in the real world.

Following the great achievements of DQfD, in this work, we continue considering the integration of reinforcement learning with human demonstrations. Our contributions are in three folds. First, since DQfD is derived from DQN, it only considers discrete action domain. We follow up the scenario of DQfD but differs in taking continuous actions domains into account by improving DDPG with human demonstrated data. Second, we make small modifications to the technique of Experience Replay of DQN and DDPG by constructing an integrated replay buffer, and adjust the usage of the buffer according to need. The small alteration evidently improves DDPG in stability. Third, we attempt to learn human preferences without contradicting to the overall target of the task. Most RL methods have only a solely overall goal, but does not consider the human behavior in realizing the goal, which can be reflected from the slightly differences between human personal preferences in completing the same task. We tackle this problem by recording demonstrations with human preferences and define specific optimization objectives to ensure the consistence with human preferences. We also apply our method to the problem of autonomous driving and examine our framework in the driving simulator TORCS[18].

The remainder of this paper is structured as follows. In Section II foundations of our work are briefly introduced. We review two import state-of-the-arts, i.e. DDPG and DQfD which motivate our work. In Section III we propose the framework which combines DDPG with human demonstrations and integrated experience replay. The validity of our proposed method is demonstrated in Section IV with a TORCS-based autonomous driving task. The proposed method not only masters the preferences in choosing the lane, but also outperforms human expert demonstration in average reward. Finally Section V concludes the paper and puts forward some future plans.

II. Preliminary

In this section, we mainly review some details of the state-of-the-art reinforcement learning algorithms in brief, i.e., Deep Q-Learning (DQN), Deep Deterministic Policy Gradient (DDPG) and Deep Q-Learning from Demonstration (DQFD), which have motivated our work.

A. Deep Q-Learning

Proposed by Mnih et al. in 2013 [6] and improved in 2015 [7], Deep Q-Learning (DQN) is an extension of the traditional Q-learning, which formalizes the interaction of the agent with the environment with a Markov Decision Process (MDP). In particular, the agent at the time step t transfers from the state s_t to a new state s_{t+1} by taking an action a_t, and receives a reward r_t which indicates the performance of the transition. The objective of Q-learning is to maximize the Q value function according to Bellman Equation with the following form:

$$Q\left(s_t, a_t\right) = r_t + \gamma \max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1}\right) \tag{1}$$

where γ is a discounting factor. Before DQN, when approximating the value function for Q-learning with a nonlinear structure, e.g. a neural network, the convergence of the exploration cannot be guaranteed. DQN introduces Experience Replay and Target Network to address the issue. First, when exploring with the environment, the sequences of transitions $\{s_t, a_t, r_t, s_{t+1}\}$ are saved in a replay buffer. Then a deep neural network is trained to approximate the Q-function with the transitions randomly drawn from the replay buffer. The technique largely breaks the correlations of consecutive transitions and makes the learning more stable. Second, when update the Q-network, the target of Q-values are computed as follows

$$y_t = r_t + \gamma \max_{a_{t+1}} \hat{Q}\left(s_{t+1}, a_{t+1}; \theta\right) \tag{2}$$

where θ is the set of parameters of the Q-network and $\hat{Q}$ denotes the target network. The target of learning is to learn the value of θ by minimize the error between the target Q values and the output of the current Q-network (TD-error):

$$J\left(Q\right) = \frac{1}{N} \sum_{t} \left(r_t + \gamma \max_{a_{t+1}} \hat{Q}\left(s_{t+1}, a_{t+1}|\theta\right) - Q\left(s_t, a_t|\theta\right)\right)^2 \tag{3}$$

Since Q-learning only considers discrete action domain, a greedy policy $\mu\left(s\right) = \arg\max_a Q\left(s, a\right)$ is applied for selecting an action. However, for continuous action domain, DQN is no longer valid. Another actor-critic approach based on the Deterministic Policy Gradient (DPG) algorithm [19] called Deep Deterministic Policy Gradient (DDPG) is proposed [8].

B. Deep Deterministic Policy Gradient

DDPG is based on the actor-critic mechanism, but also preserves the technique of Experience replay and Target Network from DQN. DDPG separately parameterizes a critic function $Q(s,a)$ and an actor function $\mu(s|\theta^{\mu})$. The critic function is defined similar to the value function in Q-learning, and updated by minimizing:

$$\begin{aligned} J(Q) &= \tfrac{1}{N}\sum_{t} L^2 \\ L &= r_t + \gamma \hat{Q}(s_{t+1}, \hat{\mu}(s_{t+1}|\theta^{\hat{\mu}})|\theta^{\hat{Q}}) - Q(s_t, a_t|\theta^{Q}) \end{aligned} \tag{4}$$

The actor function deterministically maps the current state to a specific action, and is updated by:

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

where t is the training step, and N is the number of training steps. Q and μ represent the critic and actor function respectively. $\hat{Q}$ and $\hat{\mu}$ represent the target critic function and the target actor function, respectively. θ^{Q} and θ^{μ} are the parameters of critic and actor network, respectively. Finally, the target networks are updated by copying the original networks with a delay factor τ:

$$\begin{aligned} \theta^{\hat{Q}} &= \tau\theta^{Q} + (1-\tau)\theta^{\hat{Q}} \\ \theta^{\hat{\mu}} &= \tau\theta^{\mu} + (1-\tau)\theta^{\hat{\mu}} \end{aligned} \tag{6}$$

C. Deep Q-Learning from Demonstration

To improve the performance of DQN with human expertise, T. Hester et al. [16] proposed an improved DQN algorithm with demonstration call Deep Q Learning from Demonstration (DQfD). The main goal of this algorithm is to let the agent learn as much as possible from demonstrated data before running on a real system. DQfD uses pre-training to let the agent imitate the demonstrator and get a good initial value function which satisfies the Bellman equation at the same time. The main contribution of DQfD is to accelerate the learning speed by pretraining with demonstration data. It applies three losses to update the network:

$$J_{DQ}(Q) = [R(s,a) + \gamma Q(s_{t+1}, a_{t+1}^{\max}; \theta')] - Q(s_t, a_t; \theta))^2 \tag{7}$$

$$J_E(Q) = \max_{a \in A}[Q(s,a) + l(s, a_E, a)] - Q(s, a_E) \tag{8}$$

$$J(Q) = J_{DQ} + \lambda_1 J_E(Q) + \lambda_2 J_{L2}(Q) \tag{9}$$

where J_DQ is the double DQN loss, J_E is the supervised large margin classification loss, and J_{L2} is the L2 regularization loss. λ_1 and λ_2 are the weights between the losses.

Experiments show good results about DQfD, and it receives more average rewards than DQN on 27 of 42 Atari games. However, just like DQN, DQfD can only be implemented in discrete action spaces. In the next section, we explain our proposed method by integrating DDPG with human demonstration in great detail.

III. DDPG With Demonstration

The DDPG algorithm has good results in continuous domains, including driving cars in TORCS. But it has some shortcomings e.g. the need for a rich supply of training data. Also, in some special cases like driving cars, we may want the agent to learn some driving styles from expert's demonstrations, but the DDPG algorithm can just optimize along a given reward function, which is hard to describe the expert's preferences. In addition, if supervised learning is introduced for imitating the expert's preferences, the large amount of labelled data needed by supervised learning is almost uncollectible in practice. So, we bring experts demonstrations together with reinforcement learning, try to let the agent learn an appropriate policy by interacting with the environment on itself, and learn from expert's demonstrations at the same time.

A. Combined Loss

We try to introduce demonstration data into DDPG. Unlike DQN, the actions here are continuous. So, in order to determine the similarity between actions, we approximately evaluate the two considered actions by their mean squared error. In the previous work of [16], they designed a large margin classification loss to make the greedy policy imitate the demonstrator as (8).

As DQN uses a greedy way to find the optimal policy, and the actions are finite discrete values, it is easy to calculate the supervised error for DQfD. However, in continuous action domain, the greedy policy is inapplicable, so we try to use the output of actor network directly for supervision.

In this paper, we want the agent to learn both from the demonstrator and the environment, which means that, if the agent is currently at a certain state of demonstration buffer, the action it takes should be consistent with the corresponding demonstrator's action. So, we evaluate the similarity between these two actions by the following formulation:

$$|\mu(s) - \mu_E(s)| < \varepsilon \tag{10}$$

where, μ and μ_E represent the actor network's output and the demonstrator's action respectively. s is the state which the agent is currently in. ε is a given threshold to control the similarity. When (10) holds, the supervised loss J_{sup} is defined to be zero, otherwise, we will calculate the supervised loss J_{sup} as follows:

$$\begin{aligned} J_{\text{sup}} &= \tfrac{1}{N}\sum_{(s_E, a_E)\in D^{demo}} (Q_{\text{sup}} - Q(s_E, a_E|\theta^{Q}))^2 \\ Q_{\text{sup}} &= H(a_E, \mu(s_E)) + Q(s_E, \mu(s_E|\theta^{\mu})|\theta^{Q}) \end{aligned} \tag{11}$$

where, $H(a_E, \mu(s_E))$ is zero when (10) holds, and a positive value otherwise.

We combine this supervised loss with the original TD-loss to update the parameters of the critic network, and control

the weighting between the losses by a given parameter λ.

$$J_Q = R(s_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{12}$$

$$J_{com} = J_Q + \lambda J_{\text{sup}} \tag{13}$$

where, J_Q is the TD-loss, and J_{com} is the combined loss.

B. Integrated Replay Buffer

In DQN, the authors use a replay buffer to store the transitions generated by interacting with the environment, and randomly sample batches for training. This approach helps to reduce the relevance between the sequential transitions, and guarantees the i.i.d. of the training samples.

In DQfD, two replay buffers are utilized to store the self-generated data and demonstrator's data respectively. Training data is sampled from these two buffers by a certain proportion.

As DQfD does, we also use different buffers to store the demonstration data (D^{demo}) and the self-generated data (D^{replay}). Further more, we add another buffer to store the self-generated training data with good performances (D^{good}). For the beginning episodes which usually does not contain enough good data, we instead use another buffer (D^{best}) to collect the best transitions in every training episode, and substitute for the good performance buffer temporally. As this procedure can partly ensure the data diversity, we find that this may improve the performance of training, and the training tends to be a little more stable in this way. For the self-generated data, only the Q-learning loss is applied, while for demonstration data, both Q-learning loss and the supervised loss are applied. For both kind of data, we implement regularization to prevent the training from over-fitting.

How to evaluate whether a transition is good or bad is a challenge task. In our work, the performance evaluation of actions is defined by a reward function, which is also used in reinforcement learning procedure. But in other tasks, some transitions with special results might be considered to be good. In the beginning training episodes, the policies are usually not appropriate and the transitions usually have poor performance, so we instead include the best transition (the transition with the highest reward) in this episode in the good performance buffer. As the training proceeds, the agent will learn a better policy, which means that the number of good performance transitions will increase rapidly. When the number of D^{good} is larger than D^{best}, we use D^{good} to be the good performance buffer instead of D^{best}.

Algorithm 1 DDPG with demonstration

```
Randomly initialize critic network and actor network with weights θ^Q and θ^μ.
Initialize target network with the same weights as above.
Initialise replay buffer D^replay, D^demo, D^best and D^good.
Initialize good transition threshold r_th and the training threshold ξ.
for episode=1,M do
    Receive initial observation state s_1.
    for step=1,T do
        Select an action with Ornstein-Uhlenbeck exploration noise.
        Store transition (s_t, a_t, r_t, s_{t+1}) in D^replay.
        if r_t > r_th then
            Store transition (s_t, a_t, r_t, s_{t+1}) in D^good.
        else
            Compare with other transitions in this episode, and store the best transition in D^best.
        end if
        if num(D^good) > num(D^best) then
            Sample a random mini-batch from D^replay, D^good and D^demo(when the number of training episode is less than ξ) with a certain proportion
        else
            Sample a random mini-batch from D^replay, D^best and D^demo(when the number of training episode is less than ξ) with a certain proportion.
        end if
        Update the critic by minimizing the combined loss (13).
        Update the actor using the policy gradient (5).
        Update the target networks by (6).
    end for
end for
```

C. Network Training Process

The parameter updating procedure is exactly the same as DDPG when self-generated data is used. Our DDPG with demonstration algorithm is shown in Fig. 2. When demonstration data is used, the algorithm differs with DDPG in the updating of the critic network.

We use demonstration data (s_E,a_E) to be the critic network's inputs, while s_E is the input of the actor network. The actor network yields an action a, and we want this action to be similar enough with the demonstrator's action a_E. Meanwhile, we also want this action to satisfy the Bellman Equation and to yield higher rewards when interacting with the environment. So, we use a combined loss to update the critic network's parameters as Section III-A shows. To guarantee the convergence of training, we only implement supervised loss in the beginning ξ episodes, where ξ is a given threshold. This updating process is summarized in Algorithm 1.

IV. Experiments

We implement our algorithm on a simulation platform named TORCS, which is usually used for testing AI drivers. The driving process can be viewed as a Markov Decision Process (MDP), and we take a vector which contains 29 sensor inputs as the state, and 3 continuous values (steering, acceleration, and brake) as the actions. These sensor inputs

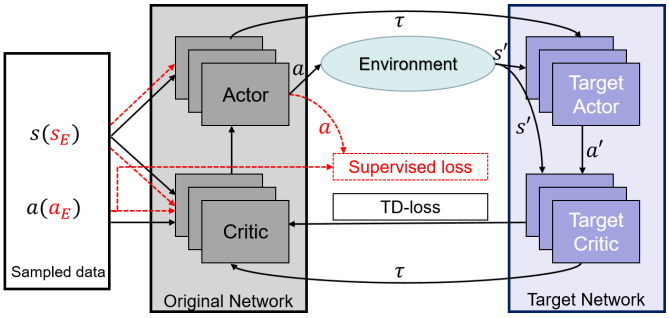
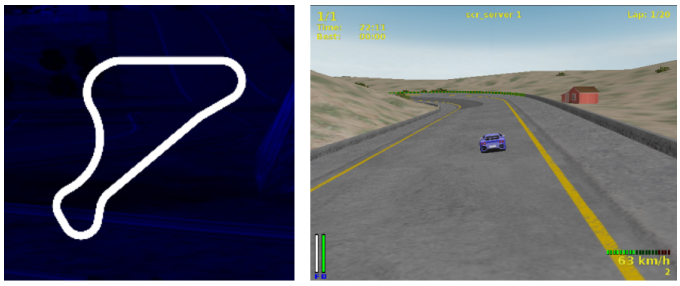


Fig. 2. The algorithm structure of DDPG with demonstration. The demonstration data (sampled from D^{demo}) is shown in red color, while the self-generated data (sampled from D^{replay} and D^{good}) is in black. s and a, s' and a' represent the current and next state-action pair respectively. τ is the weight for updating the target network.



(a) (b)

Fig. 3. The training track. (a). The global map. (b). Experimental scenes

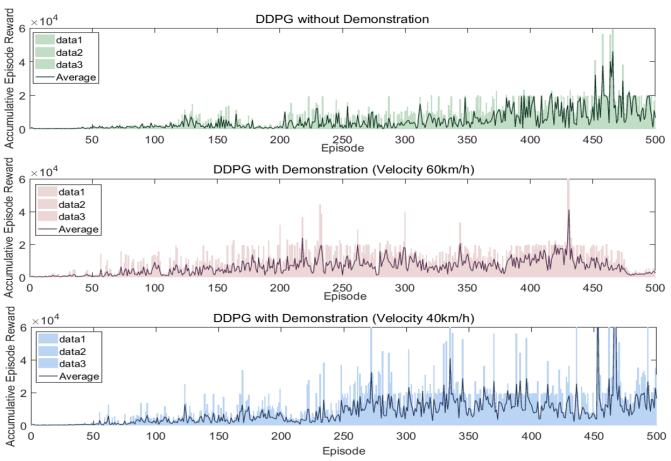


Fig. 4. The comparison results of the accumulative episode rewards. Both the 2 results of DDPG with demonstration can get higher accumulative rewards than the original DDPG.

contain the angle, range finders, velocities etc. which can be found in [18]. In this work, we define the reward function in a simple way:

$$R(V_x, \theta) = V_x \cdot \cos(\theta) \tag{14}$$

Where, V_x is the velocity along the car's orientation, and θ is the angle between the car and the track. When the car is out of the track, or stay at a very low velocity (less than $5km/h$), we just terminate this episode and give negative rewards.

The training track we select is a practicing track named CG Speedway number1 which is shown in Fig. 3.

A. Learning from demonstrations

We use a hand coded driver to generate the demonstration data. The hand coded driver is defined to prefer to drive on one single side (right or left), instead of driving along the central line, and the driving velocity is maintained at a low value. We record the states and actions of the demonstration driver, and calculate the rewards by the reward function above.

Fig. 4. shows the comparison result of the accumulative episode rewards with(without) the demonstration. The picture shows the result of the beginning 500 episodes which is about a quarter of the whole training process. We can find that both the 2 results of DDPG with demonstration can get higher accumulative rewards than the original DDPG, and the positions at which the curves start to rise are also a little earlier than the original DDPG. We also find that during the training process, the agent usually tries to drive on a certain single side as the demonstrator does, rather than driving along the central line.

In fact, with the help of reinforcement learning, the learned policy always yield more rewards than the demonstrator, which means that although the demonstration is suboptimal, it still helps the agent to learn a better policy. With the help of demonstrations, we can use a simple reward function rather than a carefully designed one which is usually hard to define. Note that the reward function we define here is not conflicted with the demonstrator's actions. If we define a reward function which has contradiction with the demonstrator's actions (e.g. limits the agent to drive along the central line), the training may not converge to a stable policy.

B. Integrated buffers

We conduct contrast experiments to verify the effectiveness of integrated buffers. Fig. 5. shows a more general case for DDPG with integrated buffers and the original DDPG. For each algorithm, we conduct 3 experiments, and visualize the average line of these 3 experiments. The result shows that DDPG with integrated buffers can get as high average rewards as the original DDPG, but with the help of integrated buffer, our algorithm seems to learn more stably, especially after 1200 steps. To show the result more clearly, Fig. 6. shows the comparison result of the variance(every 10 points as a group for variance). We can find that, DDPG with integrated buffers has lower variance in the most cases.

V. CONCLUSION

In this paper, we proposed a reinforcement learning method which could be implemented in continuous action spaces. This algorithm which we called DDPG with demonstration can learn some demonstrator's preferences and accelerate the training process at the same time. We collected the demonstrator's data, introduced supervised loss

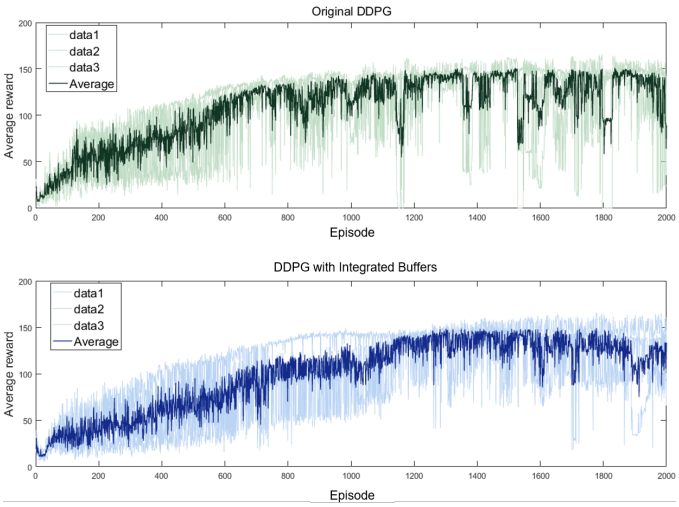


Fig. 5. The result of average line of 3 experiments. It shows that DDPG with integrated buffers can get as high average rewards as the original DDPG, but with the help of integrated buffer, our algorithm seems to learn more stably, especially after 1200 steps.

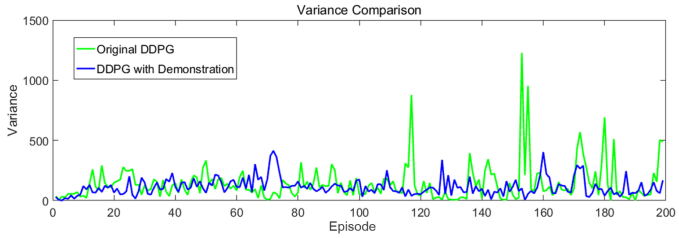


Fig. 6. The comparison results of the variance. DDPG with integrated buffers has the lower variance in the most cases.

into DDPG, and used a combined loss to train the network. In order to reduce the fluctuation during training, we sampled training data from an integrated experience replay buffer, which is consisted of 3 buffers: D^{replay}, D^{demo} and D^{good}. We implemented this algorithm in an auto-driving simulator TORCS, and found that with the help of demonstration data, the training tends to converge to an appropriate policy faster than the original DDPG. The integrated experience replay buffer can have some positive effect on reducing the training variance, and our proposed algorithm can learn more stably than the original DDPG.

ACKNOWLEDGMENT

This research was supported by National Natural Science Foundation of China under Grant No. 91648012.

REFERENCES

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484-489, 2016.

[2] S. Calinon, P. Kormushev, and D. G. Caldwell, "Compliant skills acquisition and multi-optima policy search with EM-based reinforcement learning," *Rob. Auton. Syst.*, vol. 61, no. 4, pp. 369-379, 2013.

[3] S. Ross, B. Chaib-draa, and J. Pineau, "Bayesian reinforcement learning in continuous POMDPs with application to robot navigation," in *IEEE International Conference on Robotics and Automation*, 2008, pp. 2845-2851.

[4] N. Prasad, L.-F. Cheng, C. Chivers, M. Draugelis, and B. E. Engelhardt, A Reinforcement Learning Approach to Weaning of Mechanical Ventilation in Intensive Care Units, *arXiv preprint: 1704.06300*, 2017.

[5] S. Bitzer, M. Howard, and S. Vijayakumar, Using Dimensionality Reduction to Exploit Constraints in Reinforcement Learning, in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 3219-3225.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," in *2013 Annual Conference on Neural Information Processing Systems (NIPS 2013) Deep Learning Workshop*, 2013, pp. 1-9.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. a Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529-533, 2015.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *2016 International Conference on Learning Representations*, 2016.

[9] A. G. Billard, S. Calinon, and R. Dillmann, "Learning from Humans," in *Springer Handbook of Robotics 2nd edition*, 2016, pp. 1995-2014.

[10] S. Ross, G. J. Gordon, and J. A. Bagnell, "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS) 2011*, 2011.

[11] C. Finn, S. Levine, and P. Abbeel, "Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization," in *International Conference on Machine Learning (ICML)*, 2016.

[12] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé, "Reinforcement learning from demonstration through shaping," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2015, pp. 3352-3358.

[13] G. Wang, Z. Fang, P. Li, and B. Li, "Shaping in reinforcement learning via knowledge transferred from human-demonstrations," in *Chinese Control Conference, CCC*, 2015, pp. 3033-3038.

[14] S. Griffith, K. Subramanian, and J. Scholz, "Policy Shaping: Integrating Human Feedback with Reinforcement Learning," *Adv. Neural Inf. Process. Syst.*, pp. 1-9, 2013.

[15] G. Wang, Z. Fang, P. Li, and B. Li, "Transferring knowledge from human-demonstration trajectories to reinforcement learning," *Trans. Inst. Meas. Control*, pp. 1-8, 2016.

[16] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys, "Learning from Demonstrations for Real World Reinforcement Learning," *arXiv preprint:1704.03732*, 2017.

[17] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," *arXiv preprint:1706.03741*, 2017.

[18] D. Loiacono, L. Cardamone, P. L. Lanzi, P. Milano, D. Elettronica, and I. Bioingegneria, Competition Software Manual, April, 2013.

[19] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," *Proc. 31st Int. Conf. Mach. Learn.*, pp. 387-395, 2014.