Driving Control with Deep and Reinforcement Learning in The Open Racing Car Simulator

Yuanheng Zhu[1,2(✉)] and Dongbin Zhao[1,2(✉)]

[1] Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
{yuanheng.zhu,dongbin.zhao}@ia.ac.cn
[2] School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract. Vision-based control is a hot topic in the field of computational intelligence. Especially the development of deep learning (DL) and reinforcement learning (RL) provides effective tools to this field. DL is capable of extracting useful information from images, and RL can learn an optimal controller through interactions with environment. With the aid of these techniques, we consider to design a vision-based robot to play The Open Racing Car Simulator. The system uses DL to train a convolutional neural network to perceive driving data from images of first-person view. These perceived data, together with the car's speed, are input into a RL-learned controller to get driving commands. In the end, the system shows promising performance.

Keywords: TORCS · Vision-based control · Reinforcement learning
Deep learning

1 Introduction

The Open Racing Car Simulator (TORCS)[1] is an open source 3D car racing simulator. It provides realistic experience with powerful physics engines and sophisticated 3D graphics. Players can not only drive cars in TORCS, but also design their own robots with intelligent techniques [8,17]. Based on computational intelligence (CI) techniques, numerous robots have been successfully developed by researchers to play TORCS [2,12].

Most robots use real measurements from the TORCS engine as input state, such as distance, angle, track shape, to name a few. These data are reliable and low-dimensional, but must be provided by the TORCS engine. In contrast, when humans play TORCS or drive real cars, they can perform well based on only drivers' view. In recent years, deep learning (DL) makes it feasible and easy to process high-dimensional images [4,6,7]. Essential features can be extracted with deep neural networks (DNNs). Inspired by that, in [3], authors try to predict

[1] http://torcs.sourceforge.net/.

L. Cheng et al. (Eds.): ICONIP 2018, LNCS 11303, pp. 326–334, 2018.
https://doi.org/10.1007/978-3-030-04182-3_29

driving data from the first-person view in TORCS. They collect images and data and put them into a convolutional neural network (CNN) to train network weights.

However, DL lacks the ability of interacting with external environment. To achieve vision-based control in complex systems like TORCS, researchers have been working on combining DL with reinforcement learning (RL). RL considers how to choose a series of actions to maximize the accumulated rewards from environment [5,9,14–16]. Researchers in [10,11,18] combine these two methods and propose deep reinforcement learning (DRL) to play Atari games. To achieve satisfying results, these DRL algorithms have to run plenty of trials through interactions with environment, and most early trials end up with failures. For vision-based autonomous driving, it is more reliable to separate action-decision and image-perception processes apart. A driving controller should be learned with only a small number of trials, and a perception module can be trained by data that are collected from skilled drivers in a safe condition.

In this paper, we aim to integrate the latest RL and DL methodologies together to design a vision-based self-driving robot in TORCS. First we use low-dimensional, ground-truth driving data provided by the TORCS engine to learn a driving controller with only a small number of trials. Then we train a CNN to perceive driving data from images of first-person view. After integrating the strategy and perception parts together, our robot only takes the first-person view and its own speed as input, and can drive successfully. Since the framework only involves collecting data from human drivers in a safe condition and learning controllers with a small number of trials, it is easy and reliable to extend the work to practical applications.

2 Problem Description in TORCS

The control variables in TORCS (Fig. 1) include $\mathbf{u}_t = [\delta_t, \tau_t]^T$. δ_t is the steering angle percentage, ranged by $[-1, 1]$. τ_t is the throttle or brake percentage, ranged by $[-1, 1]$. t specifies the time index. As for the gear control, we use an automatic transmission algorithm to shift the gear automatically.

For ease of analysis, the car dynamics is treated as a discrete-time system with a fixed step dT. The evolving variables, which we term as *inherent variables*, include $\mathbf{x}_t = [d_t, a_t, v_t]^T$, where d_t is the deviation distance (m), a_t is the deviation angle (rad), and v_t is the current speed (km/h). Its evolution is determined by command $\mathbf{u}_t$ and *dynamical variables* $\mathbf{y}_t = [d_t, a_t, v_t, \kappa_t]^T$ where κ_t is the road curvature (m^{-1}). The transition function is defined as $\mathbf{x}_{t+1} = f(\mathbf{y}_t, \mathbf{u}_t)$ and it is unknown to robot designers. Due to disturbance and sensor noise, observations of $\mathbf{x}_{t+1}$ are perturbed by noise, which here we assume as Gaussian noise $\varepsilon \sim \mathcal{N}(0, \Sigma_\varepsilon)$, where $\Sigma_\varepsilon = \mathrm{diag}(\sigma_{\varepsilon_d}, \sigma_{\varepsilon_a}, \sigma_{\varepsilon_v})$.

The following *cost* function evaluates the driving performance at each step

$$c_t = c(\mathbf{z}_t) = 1 - \exp\left(-\frac{1}{2b^2}\left[\omega_d d_t^2 + \omega_a a_t^2 + \omega_v (v_t - v_t')^2\right]\right) \tag{1}$$

Fig. 1. Screenshot of TORCS.

where v_t' is an auxiliary variable that represents the desired speed (km/h) at the current position. b, $\omega_d, \omega_a, \omega_v$ are the cost coefficients. The *cost variables* $\mathbf{z}_t$ are composed of $\mathbf{z}_t = [d_t, a_t, v_t, \kappa_t, v_t']^T$. The long-term goal is to minimize the *return*, which is the sum of costs during a period of time $\min J = \min \mathbb{E}\left[\sum_{t=1}^{T} c_t\right]$.

The desired speed v_t' is widely used in the design of TORCS robots [1,13]. It is calculated based on track curvature and road friction coefficient. Due to page limit, we omit the calculation details and suggest readers to refer to [1,13].

The controller is constructed in the form of $\mathbf{u}_t = \pi(\mathbf{z}_t|p)$, where p represent controller parameters, and the target is to minimize J. Note that the controller needs all the variable information, including not only physical variables like d_t, a_t, κ_t, but also auxiliary variable v_t'. In the open-source TORCS, these data are available from the TORCS engine. In the next, we plan to learn the controller parameters by RL based on real variable information, and then perceive these variables from images by DL so that the robot can drive based on first-person view.

3 Learn Driving Controller by Modified PILCO

3.1 Gaussian Process Model

PILCO [5], short for Probabilistic Inference for Learning COntrol, is a model-based RL algorithm. In PILCO, the system dynamics is considered as a Gaussian Process (GP). Suppose we have collected a group of driving data $\{\mathbf{y}_t, \mathbf{u}_t\}$. We use $\tilde{\mathbf{y}}_t = [\mathbf{y}_t^T, \mathbf{u}_t^T]^T$ as training inputs, and the difference $\Delta\mathbf{x}_{t+1} = \mathbf{x}_{t+1} - \mathbf{x}_t + \varepsilon$ as training targets, where ε is Gaussian noise. The mean and variance of $\mathbf{x}_{t+1}$ now become $\mu_{t+1} = \mathbf{x}_t + \mathbb{E}_f[\Delta\mathbf{x}_{t+1}]$, $\Sigma_{t+1} = \mathrm{var}_f[\Delta\mathbf{x}_{t+1}]$.

Suppose there exist n training inputs, $\tilde{\mathbf{Y}} = [\tilde{\mathbf{y}}_1, \ldots, \tilde{\mathbf{y}}_n]$, and n training targets, $\Delta\mathbf{X} = [\Delta\mathbf{x}_1, \ldots, \Delta\mathbf{x}_n]$. Consider the scalar target $\Delta\mathbf{x}_i \in \mathbb{R}$ and deterministic test input $\tilde{\mathbf{y}}_*$. The predictive probability of test target $\Delta\mathbf{x}_*$ is Gaussian with mean and variance as

$$\mu_* = \mathbb{E}_f[\Delta \mathbf{x}_*] = \mathbf{k}_*^T(\mathbf{K} + \sigma_\epsilon^2 \mathbf{I})^{-1}\Delta \mathbf{X} = \mathbf{k}_*^T \beta \tag{2}$$

$$\sigma_*^2 = \mathrm{var}_f[\Delta \mathbf{x}_*] = k_{**} - \mathbf{k}_*^T(\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1}\mathbf{k}_* \tag{3}$$

where $\mathbf{k}_* = k(\tilde{\mathbf{Y}}, \tilde{\mathbf{y}}_*)$, $k_{**} = k(\tilde{\mathbf{y}}_*, \tilde{\mathbf{y}}_*)$, $\beta = (\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1}\Delta \mathbf{X}$, and $\mathbf{K}$ is the Gram matrix with entries $K_{ij} = k(\tilde{\mathbf{y}}_i, \tilde{\mathbf{y}}_j)$. Here the kernel function k selects the squared exponential (SE) kernel

$$k(\tilde{\mathbf{y}}_1, \tilde{\mathbf{y}}_2) = \alpha^2 \exp\left(-\frac{1}{2}(\tilde{\mathbf{y}}_1 - \tilde{\mathbf{y}}_2)^T \Lambda^{-1}(\tilde{\mathbf{y}}_1 - \tilde{\mathbf{y}}_2)\right) \tag{4}$$

where α and Λ are function parameters. These parameters can be learned by evidence maximization.

When test input is distributed, the target distribution is complicated but we can still approximate it as a GP. Still consider the scalar target, i.e. $\Delta \mathbf{x}_i \in \mathbb{R}$, and suppose the test input satisfies $\tilde{\mathbf{y}}_* \sim \mathcal{N}(\mu, \Sigma)$. The target distribution is approximated by Gaussian $\Delta \mathbf{x}_* \sim \mathcal{N}(\mu_*, \sigma_*^2)$ where

$$\mu_* = \beta^T \mathbf{q}, \sigma_*^2 = \alpha^2 - \mathrm{tr}\left((\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1}\tilde{\mathbf{Q}}\right) + \beta^T \tilde{\mathbf{Q}}\beta - \mu_*^2$$

$$\mathbf{q} = [q_1, \ldots, q_n]^T, q_i = \alpha^2 |\Sigma \Lambda^{-1} + \mathbf{I}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\tilde{\mathbf{y}}_i - \mu)^T(\Sigma + \Lambda)^{-1}(\tilde{\mathbf{y}}_i - \mu)\right)$$

and $\tilde{\mathbf{Q}}$ is a $n \times n$ matrix with entries

$$\tilde{Q}_{ij} = \frac{k(\tilde{\mathbf{y}}_i, \mu)k(\tilde{\mathbf{y}}_j, \mu)}{|2\Sigma \Lambda^{-1} + \mathbf{I}|^{\frac{1}{2}}} \exp\left((\tilde{\rho}_{ij} - \mu)^T(\Sigma + \frac{1}{2}\Lambda)^{-1}\Sigma \Lambda^{-1}(\tilde{\rho}_{ij} - \mu)\right)$$

and $\tilde{\rho}_{ij} = \frac{1}{2}(\tilde{\mathbf{y}}_i + \tilde{\mathbf{y}}_j)$. The above results of scalar input can be easily extended to multivariate case, so we omit it here.

One drawback of GP is its computational complexity. If the data set is large, the training and predicting processes will be slow and unsuitable for real applications. We discretize the input space into non-overlapping equal-sized cells, and each cell can store at most one data [19]. In this way, the stored data are naturally separated and a sparse training set is obtained.

3.2 Return Evaluation

The driving controller π is specified to a linear controller with saturation

$$\mathbf{u}_t = \pi(\mathbf{z}_t) = \mathbf{u}_{\max}\mathrm{sat}(\mathbf{w}\mathbf{z}_t + \mathbf{b}) \tag{5}$$

where the saturation function is defined by $\mathrm{sat}(a) = \frac{1}{8}(9\sin(a) + \sin(3a))$. $\mathbf{u}_{\max}$ indicates the maximum command values. For simplicity, we denote $\mathbf{p} = \{\mathbf{w}, \mathbf{b}\}$.

With the controller structure in (5), it is feasible to compute mean and variance of control variables $\mathbf{u}_t$ with a Gaussian distributed input $\mathbf{z}_t$. Similarly, the probability of control variables is approximated by Gaussian with the calculated

mean and variance. If we split the distribution of dynamical variables $\mathbf{y}_t$ from $\mathbf{z}_t$ and combine with action $\mathbf{u}_t$, the Gaussian distribution of input $\tilde{\mathbf{y}}_t = [\mathbf{y}_t^T, \mathbf{u}_t^T]^T$ is known. With the trained GP model, the probability of next-step $\mathbf{x}_{t+1}$ is predicted. Combined with the cost function given in (1), the expected cost of $\mathbf{z}_{t+1}$ is analyzed by $\mathbb{E}[c_{t+1}] = \int c(\mathbf{z}_{t+1})p(\mathbf{z}_{t+1})d\mathbf{z}_{t+1}$ if we further specify the desired velocity v'_{t+1}.

The above process can be repeated for the next $(T-1)$ steps. Given road curvatures $\kappa_t, \ldots, \kappa_{t+T}$ and desired velocities $v'_t, \ldots, v'_{t+T}$, the distributions of $\mathbf{x}_{t+1}, \ldots, \mathbf{x}_{t+T}$ and the corresponding costs are calculated. The estimated return of a starting state $\mathbf{x}_t$ under the current controller is analyzed and is related to the controller parameters $\mathbf{p}$ in the form $J = \sum_{k=t+1}^{t+T} \mathbb{E}[c(\mathbf{z}_k)] \propto \mathbf{p}$.

3.3 Policy Gradient Search

With the analytic solution of J, we calculate the gradient of J towards the controller parameters $\mathbf{p}$. Then policy gradient search is followed to adjust $\mathbf{p}$ to minimize J. However, computing J needs not only a starting state $\mathbf{x}_t$, but also external variables $\kappa_{t+1}, \ldots, \kappa_{t+T}$ and $v'_{t+1}, \ldots, v'_{t+T}$. We define multiple scenarios with different starting states $\mathbf{x}$ and different κ, v' for policy gradient search in order to gain comprehensive performance. Once the gradient is calculated, $\mathbf{p}$ can be trained by many optimization methods to minimize J.

4 RL Experiment Results

Now we apply the modified PILCO algorithm in TORCS. The track we used for learning is *CG Track 3* and it is marked by lane lines to mimic real-world roads with one lane as illustrated in Fig. 1. The slowdown deceleration a_b selects $2\,\mathrm{m/s^2}$. The discrete-time step selects 0.1 s. The Gaussian noise ε of the observed $\mathbf{x}_t$ satisfies $\varepsilon \sim \mathcal{N}(0, \Sigma_\varepsilon)$, where $\Sigma_\varepsilon = \mathrm{diag}([0.01, 0.01, 1.5]^2)$. The bounds of control actions are $\mathbf{u}_{\max} = [1, 1]^T$. The width b in cost function selects 0.4. The importance weights are set to $\omega_d = 1, \omega_a = 1, \omega_v = 400$. When computing the gradient $\mathrm{d}J/\mathrm{d}\mathbf{p}$ with GP model, the future steps T choose 30. To store GP training data, each dimension is divided by 20 between its lower and upper bounds according to experimental experience. 7 scenarios are defined for calculating the gradient, including straight cases and turning cases with different velocities, deviation distances, and curvatures.

After 6 trials, the controller is able to complete the track and the learning stops. Trajectories of states and actions using the final learned controller are plotted in Fig. 2. For comparison, the desired velocity is also plotted along with the real velocity. Small deviations only occur at the moment when the road changes from one segment to another. And the deviations are regulated in a short time.

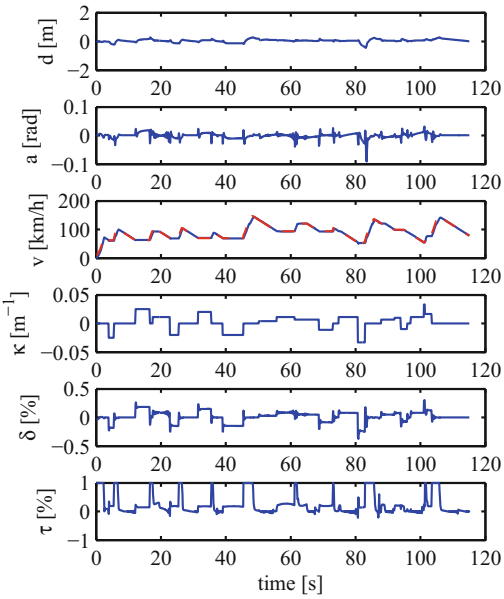

Fig. 2. Trajectories of states and actions using the final learned controller by modified PILCO. The blue solid line in figure 'v' represents the car's velocity, while the red dash-dot line represents the desired velocity. (Color figure online)

5 Perceive Driving Data from Images by DL

In the above section, we learn a driving controller with the full access to state variables provided by TORCS engine. As mentioned above, these data can be perceived from the driver's view, except the car velocity that is known to the car. Inspired by the work of [3], in this section we use a CNN to predict the driving data from images.

First we let a human player drive the car, and store images and driving data every 0.1 s. The images are directly captured from the first-person view with the size of $3 \times 210 \times 280$ (RGB). The driving data include deviation distance, deviation angle, road curvature, and desired velocity. To increase the diversity of data set and improve the generalization of network, the car is driven on different tracks with different backgrounds and lanes. At last we collect a total of 53139 images and driving data as the train set and 10699 images and driving data as the test set.

The network uses the same architecture given in [6], except the output layer is adjusted to suit our needs. To speed up the learning process, we use the results of [6] to initialize the network weights. The network is trained using stochastic gradient descent with a batch size of 64, a momentum of 0.9, and a weight decay of 0.0005. The learning rate is initialized to 0.01 and is dropped by a factor of 0.9 every 8000 steps.

Network is trained for a maximum of 100000 iterations. The curves of train loss and test loss are depicted in Fig. 3. The loss curves drop dramatically once the network starts training. That is because [6] has trained the network on a large data set of real-world images, and we use their results to initialize our network. The shallow layers have already had a high level of feature extraction. With more iterations, the train loss vibrates occasionally but the test loss keeps dropping slightly. The prediction performance of the trained CNN is illustrated in the next section where we combine the network with the driving controller.

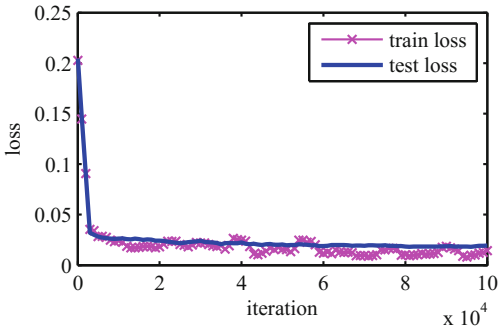

Fig. 3. Loss curves of CNN along iterations.

6 Combination of RL and DL in TORCS

Now we combine the visual perception trained by DL with the controller learned by RL, and apply them in TORCS to drive on the track of *CG track 3*[2]. CNN outputs are plotted in Fig. 4. For comparison, the ground-truth values are presented in the same figures. The predictions generally match the true values. But it is noted that the curves are not as smooth as those produced by the controller with the full access to driving data in Fig. 2. Some noticeable vibrations occur in d and a. This phenomenon is caused by CNN errors. There are small differences between the predicted values and the true values. Prediction errors disturb the controller to output right commands, and sometimes even make the car move to the opposite directions. Fortunately the prediction errors are small, so the car will not leave the track in spite of occasionally inaccurate commands.

[2] Video results are available in https://www.youtube.com/watch?v=hUpuE7qL5NQ.

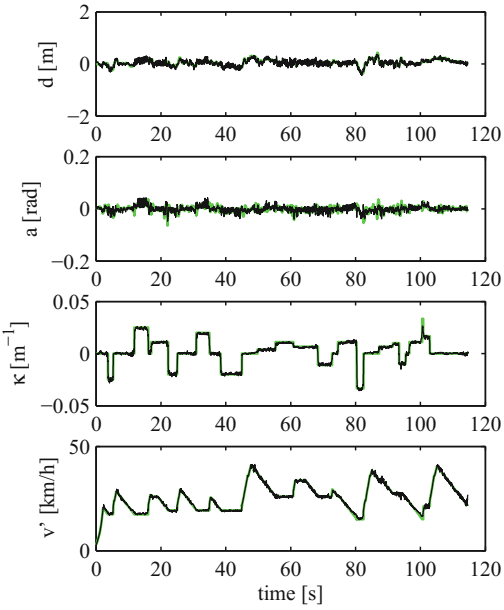

Fig. 4. Trajectories of predicted and true values in TORCS by vision-based robot. The black thin lines indicate the predicted values by CNN, while the green thick lines indicate the true values. (Color figure online)

7 Conclusion

In this paper, we first use a modified PILCO algorithm to learn a driving controller with full access to the TORCS engine. The algorithm learns a satisfactory controller with just several trials. Then we train a CNN to perceive driving data from images of first-person view in a supervised learning manner. After combining the two parts together, we get a vision-based robot for TORCS. It takes images and car's velocity as input, and drives the car well on the road.

Acknowledgments. This work is partly supported by the Beijing Science and Technology Plan under Grants Z181100008818075, National Natural Science Foundation of China (NSFC) under Grants No. 61603382, No. 61573353, No. 61533017, and the National Key Research and Development Program of China under Grant No. 2016YFB0101003.

References

1. Cardamone, L., Loiacono, D., Lanzi, P.L.: Learning drivers for TORCS through imitation using supervised methods. In: 2009 IEEE Symposium on Computational Intelligence and Games, pp. 148–155 (2009)
2. Cardamone, L., Loiacono, D., Lanzi, P.L.: On-line neuroevolution applied to the open racing car simulator. In: 2009 IEEE Congress on Evolutionary Computation, pp. 2622–2629 (2009)

3. Chen, C., Seff, A., Kornhauser, A., Xiao, J.: DeepDriving: learning affordance for direct perception in autonomous driving. In: 2015 IEEE International Conference on Computer Vision (ICCV), pp. 2722–2730 (2015)
4. Chen, Y., Zhao, D., Lv, L., Zhang, Q.: Multi-task learning for dangerous object detection in autonomous driving. Inf. Sci. **432**, 559–571 (2018)
5. Deisenroth, M.P.: Efficient Reinforcement Learning Using Gaussian Processes. KIT Scientific Publishing, Karlsruhe (2010)
6. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.) Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc., Red Hook (2012)
7. Lecun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
8. Loiacono, D., et al.: The 2009 simulated car racing championship. IEEE Trans. Comput. Intell. AI Games **2**(2), 131–147 (2010)
9. Loiacono, D., Prete, A., Lanzi, P.L., Cardamone, L.: Learning to overtake in TORCS using simple reinforcement learning. In: IEEE Congress on Evolutionary Computation, pp. 1–8 (2010)
10. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
11. Mnih, V., et al.: Playing Atari with deep reinforcement learning. CoRR abs/1312.5602 (2013)
12. Muǹoz, J., Gutierrez, G., Sanchis, A.: Controller for TORCS created by imitation. In: 2009 IEEE Symposium on Computational Intelligence and Games, pp. 271–278 (2009)
13. Muǹoz, J., Gutierrez, G., Sanchis, A.: A human-like TORCS controller for the simulated car racing championship. In: Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, pp. 473–480 (2010)
14. Shao, K., Zhu, Y., Zhao, D.: Cooperative reinforcement learning for multiple units combat in StarCraft. In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–6 (2017)
15. Shao, K., Zhu, Y., Zhao, D.: StarCraft micromanagement with reinforcement learning and curriculum transfer learning. IEEE Trans. Emerg. Top. Comput. Intell. 1–12 (2018)
16. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
17. Wymann, B., Dimitrakakisy, C., Sumnery, A., Guionneauz, C.: TORCS: the open racing car simulator (2015)
18. Zhao, D., Wang, H., Shao, K., Zhu, Y.: Deep reinforcement learning with experience replay based on SARSA. In: 2016 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–6 (2016)
19. Zhao, D., Zhu, Y.: MEC - a near-optimal online reinforcement learning algorithm for continuous deterministic systems. IEEE Trans. Neural Netw. Learn. Syst. **26**(2), 346–356 (2015)