J. Nonlinear Funct. Anal. 2025 (2025) 6 https://doi.org/10.23952/jnfa.2025.6

MATHRES Journal of Nonlinear Functional Analysis
Available online at http://jnfa.mathres.org

SOFT ACTOR-CRITIC ALGORITHM WITH ADAPTIVE NORMALIZATION

XIAONAN GAO[1], ZIYI WU[1], XIANCHAO ZHU[1,*], LEI CAI[2]

[1]School of Artificial Intelligence and Big Data, Henan University of Technology, Zhengzhou 450001, China
[2]School of Artificial Intelligence, Henan Institute of Science and Technology, Xinxiang 453003, China

Abstract. In recent years, breakthroughs were made in the field of deep reinforcement learning, but, their applications in the real world were seriously affected due to the instability of algorithms and the difficulty in ensuring convergence. As a typical algorithm in reinforcement learning, although the SAC algorithm enhances the robustness and agent's exploration ability by introducing the concept of maximum entropy, it still has the disadvantage of instability in the training process. In order to solve the problems, this paper proposes an Adaptive Normalization-based SAC (AN-SAC) algorithm. By introducing the adaptive normalized reward mechanism into the SAC algorithm, our method can dynamically adjust the normalized parameters of the reward during the training process so that the reward value has zero mean and unit variance. Thus it better adapts to the reward distribution and improves the performance and stability of the algorithm. Experimental results demonstrate that the performance and stability of the AN-SAC algorithm are significantly improved compared with the SAC algorithm.

Keywords. Adaptive normalization; Deep reinforcement learning; Reward mechanism; Soft actor-critic algorithm.
2020 MSC. 68T20.

1. INTRODUCTION

It is known that there is a significant difference between the learning mechanisms of reinforcement learning and supervised learning. In supervised learning, the model relies on human-annotated dataset labels for learning, with the aim of identifying mappings between inputs and outputs. In contrast, reinforcement learning algorithms learn through direct interaction with the environment, with the goal of maximizing the rewards received from the environment to explore and optimize strategies. With the advent of Deep Q-Learning (DQN) algorithms [1], it is possible to combine reinforcement learning with deep neural networks, which enables reinforcement

*Corresponding author.
E-mail address: xczhuiffs@163.com (X. Zhu).
Received December 24, 2024; Accepted March 1, 2025.

learning algorithms to deal with more complex and high-dimensional problems. This innovation has greatly improved the performance of reinforcement learning algorithms, making them widely used in many fields such as Go [2, 3], video games [4, 5], navigation planning [6, 7], investment and trading [8, 9, 10], and recommendation systems [11, 12].

Recently, remarkable work shave been made in the field of deep reinforcement learning. OpenAI researchers developed a reinforcement learning method that facilitates agents to continuously identify and master new skills through implicit course learning [13]. Mendonca et al. [14] proposed a supervised learning-based meta-reinforcement learning algorithm that can effectively assist exploration, especially in the environment of sparse rewards. Efroni et al. [15] proposed a reinforcement learning algorithm based on a finite time-domain look-ahead strategy, which achieved good results by using the return value of the optimal tree path to back up the descendant values of nodes. Ciosek et al. [16] proposed an algorithm called OAC (Optimistic Actor Critic), which uses two confidence intervals to estimate the value function, with a higher confidence interval to guide exploration and a lower confidence interval to prevent overfitting. Haarnoja et al. [17] proposed an algorithm called SAC (Soft Actor Critic), which enhances the exploration power of the Actor-Critic algorithm by introducing maximum entropy. Although the SAC algorithm achieves a good balance between exploration and utilization, there is still instability in its training process, which may lead to the inability of agents to adapt to changes in the environment during the exploration process, which affects the performance and stability of the algorithm.

In view of the shortcomings of the unstable SAC training process, this paper proposes an Adaptive Normalization-based SAC (AN-SAC) algorithm that combines the advantages of the SAC algorithm and the advantages of the adaptive normalization reward mechanism. Not only it can the principle of maximum entropy be used to enable the agent to pursue reward maximization and explore new space at the same time when executing the strategy, but also dynamically adjust the normalized parameters of the reward according to the agent's performance in the environment, so as to improve the performance and stability of the algorithm.

The core contributions of this paper are mainly threefold: (1) it is proposed to integrate an adaptive normalized reward method into the SAC algorithm to effectively improve the superiority and stability of the algorithm; (2) the beta distribution of the $(-1,1)$ range is used to initialize the neural network, and the training of the neural network can be accelerated in many problems. (3) Experiments are carried out in a variety of environments, and the results of the AN-SAC algorithm are compared with the classical SAC algorithm, and the superiority of the algorithm is verified.

2. PRELIMINARIES

Reinforcement learning is a machine learning paradigm that learns the optimal strategy through the interaction of agents with the environment. Through a series of interactions with the environment, the agent obtains feedback from the environment (Reward) and updates its own strategy

based on this feedback. The goal of reinforcement learning is to maximize the reward expectation in order to obtain an optimal policy. In reinforcement learning, the Markov Decision Process (MDP) framework is widely used to describe the process of agent interaction with the environment. The MDP consists of four elements: a state set, an action set, a state transition probability, and a reward-reward function. The agent obtains the state from the environment at each time step, selects an action according to the strategy, and the environment returns a reward and updates the state after the action is executed. The reward expectation at time t can be expressed as:

$$R_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l},$$

where γ is the discount factor, and the expectation of cumulative return in the current state is:

$$V_\pi(s) = E_\pi[\sum_{l=0}^{\infty} \gamma^l r_{t+l} | s_t = s] = E[R_t | s_t = s].$$

The action value is expressed as $Q_\pi(s,a) = E[R_t | s_t = s, a]$, and the optimal strategy is to select the action with the largest value in each state, which is denoted as

$$Q^*(s,a) = \max_\pi Q(s,a).$$

With the Bellman equation, the value function and the action value function can be iteratively calculated expressed as

$$Q^*(s,a) = E[r_t + \gamma max Q^*(s_{t+1}, a_{t+1})].$$

With the introduction of deep neural networks, reinforcement learning algorithms can handle more complex control and decision-making problems. However, the complexity of deep neural networks also brings some challenges, such as low training efficiency, poor stability, and sensitivity to hyperparameters.

The SAC is a reinforcement learning method that follows the principle of maximizing entropy. This algorithm adds entropy considerations to the objective function, which greatly enhances the algorithm's exploration ability and adaptability to the environment. By seeking the best balance between reward and entropy (i.e., the randomness of the strategy), the SAC can avoid prematurely falling into suboptimal deterministic strategies and local optimal solutions. Higher entropy means more environment exploration, which helps to avoid the convergence of the strategy to the local optimum, thus speeding up the learning process. The SAC formula for the optimal strategy is defined as:

$$\pi^* = \arg\max_\pi E_{s_t, a_t \sim \pi(\cdot|s_t)}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) + \alpha H(\pi(\cdot|s_t))],$$

where strategy π is used to update the largest total reward found; α is the entropy regularization coefficient, which is used to control the importance of entropy; $H(\pi(\cdot|s_t)) = E[-\log \pi(\cdot|s_t)]$ represents entropy value, the higher the entropy, the greater the agent's exploration of the environment, enabling the agent to find a more efficient strategy, which helps to speed up subsequent policy learning.

The Q value of SAC can be calculated by using the Bellman variance based on entropy improvement, expressed as

$$Q(s_t,a_t) = E_{s_{t+1}\sim D}[r(s_t,a_t) + \gamma V^{\pi}(s_{t+1})],$$

where s_{t+1} is obtained from the empirical replay buffer D, and the state value function is defined as

$$\begin{aligned} V(s_t) &= E_{a_t\sim\pi}[Q(s_t,a_t) - \alpha \log \pi(\cdot|s_t)] \\ &= E_{a_t\sim\pi}[Q(s_t,a_t) + H(\pi(\cdot|s_t))], \end{aligned}$$

where s_t represents the expected reward in a certain state. In addition, the policy network $\pi_\phi(s_t,a_t)$, the soft state value network $V_\psi(s_t)$, the target state value network $V_{\tilde{\psi}}(s_t)$, and two soft Q networks $Q_{\theta_{1,2}}(s_t,a_t)$ in the SAC are parameterized by ϕ,ψ, $\tilde{\psi}$, and θ, respectively. In order to achieve the optimal strategy for each network, the stochastic gradient descent method is used to optimize their objective function

$$J_V(\psi) = E_{s_t\sim D}[\frac{1}{2}(V_\psi(s_t) - E_{a_t\sim\pi_\phi}[\min_{i=1,2} Q_{\theta_i}(s_t,a_t) - \alpha \log \pi_\phi(a_t|s_t)])^2].$$

In addition, a form similar to a double-Q network is adopted, where the minimum value of soft Q is taken as two Q functions parameterized by θ_1 and θ_2, which helps to avoid overestimating inappropriate Q values and improves the training speed. The soft Q function is updated by minimizing the Bellman error:

$$J_V(\psi) = E_{(s_t,a_t)\sim D}[\frac{1}{2}(\min_{i=1,2} Q_{\theta_i}(s_t,a_t) - (r(s_t,a_t) + V_{\tilde{\psi}}(s_{t+1})))^2].$$

The policy network is updated by minimizing the Kurbach-Leibler (KL) divergence:

$$J_\pi(\phi) = E_{s_t\sim D, a_t\sim\pi}[\log \pi_\phi(s_t,a_t) - \min_{i=1,2} Q_{\theta_i}(s_t,a_t)].$$

3. AN-SAC

The AN-SAC algorithm proposed in this paper introduces the adaptive normalized reward mechanism into the SAC algorithm to improve the superiority and stability of the algorithm. The innovation is as follows: First, it adopts a new network initialization method to initialize network parameters through Beta distribution, which can distribute parameters within a reasonable range and help the network converge faster. Second, in the training process, the AN-SAC algorithm performs adaptive normalization of the cumulative reward, which is a method used to process the reward value in reinforcement learning, in order to convert the reward value into a data distribution with zero mean and unit variance. This conversion is beneficial for stabilizing the learning process, especially in the case of large reward distributions.

In the AN-SAC of this paper, the optimal strategy and the calculation of the Q value are expressed as:

$$\pi^* = arg\max_{\pi} E_{s_t,a_t\sim\pi(\cdot|s_t)}[\sum_{t=0}^{\infty} \gamma^t r_{new}(s_t,a_t) + \alpha H(\pi(\cdot|s_t))],$$

and

$$Q(s_t, a_t) = E_{s_{t+1} \sim D}[r_{new}(s_t, a_t) + \gamma V^{\pi}(s_{t+1})],$$

where $r_{new}(s_t, a_t)$ represents the normalized cumulative reward, which is

$$r_{new}(s_t, a_t) = \frac{qualities - mean\_reward}{std\_reward},$$

where qualities is the tensor of the cumulative reward, *mean_reward* is the mean of the cumulative reward, and *std_reward* is the standard deviation of the cumulative reward, which is calculated by the following formulas:

$$mean\_reward = \frac{1}{N} \sum_{i=1}^{N} qualities[i],$$

and

$$std\_reward = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (qualities[i] - mean\_reward)^2},$$

where N is the number of accumulated rewards.

AN-SAC not only follows the principle of maximum entropy of traditional SAC, but also uses the Beta distribution to initialize parameters and adaptively normalize the cumulative rewards. This improves the performance and stability of the algorithm.

Our method uses a new neural network initialization method, which takes a Beta distribution in the range of $(-1, 1)$ to initialize neural network parameters. The Beta distribution is a continuous probability distribution with a flexible shape and good properties, which is suitable for the initialization of neural networks. The Beta distribution can be expressed as:

$$f(x; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha - 1}(1 - x)^{\beta - 1}, x \in [0, 1],$$

where α and β are the shape parameters of the Beta distribution, and x is a random variable. By adjusting the values of α and β, we can obtain the Beta distributions of different shapes.

AN-SAC also adaptively normalizes cumulative rewards. This strategy makes the difference between the predicted value and the target value of the value network smaller by normalizing the cumulative reward, thereby improving the stability of the value network and the overall performance of the algorithm. The normalized cumulative rewards can be used to update the value network. In the AN-SAC algorithm, the update process of the value network is as follows: the cumulative reward $r(s_t, a_t)$ is first propagated forward through the neural network to obtain the mean *mean_reward* and standard deviation *std_reward* of the cumulative reward. Then, according to this information, the cumulative reward *qualities* is normalized to obtain the normalized cumulative reward $r_{new}(s_t, a_t)$. Then, the target network is used to calculate the state value and state-action value at the next moment as the update target of the value network. Then, the normalized cumulative reward is input into the value network to obtain the predicted value, and the difference between the predicted value and the target value is calculated by the loss function, and finally the value network is updated according to the gradient calculated by the loss function.

The AN-SAC approach is as follows (as shown in Algorithm 1):

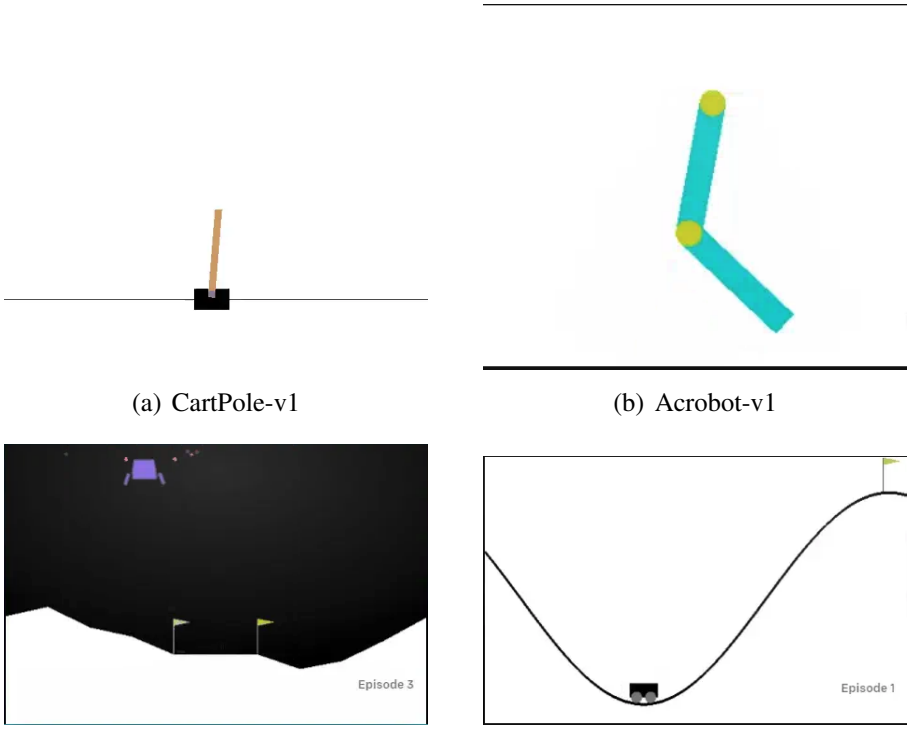
Algorithm 1 AN-SAC

Initialize the strategy network Π with the weights of the Beta distribution, and initialize the two value networks Q_{θ_1} and Q_{θ_2} with random network parameters θ_1 and θ_2.
Initialize the target value network $Q_{\tilde{\psi}_1}$,$Q_{\tilde{\psi}_2}$, copy the weights of Q_{θ_1}, Q_{θ_2} to $Q_{\tilde{\psi}_1}$,$Q_{\tilde{\psi}_2}$
Initialize the experience replay buffer D.
for sequence $e = 1, ..., E$ **do**
  Reset the environment to get the initial state s_1.
  for time step $t = 1, ..., T$ **do**
    Select action $a_t = \pi_\theta$ based on the current strategy
    Perform action a_t, get the reward r_t, and the environment state changes to s_{t+1}
    Store (s_t, a_t, r_t, s_{t+1}) into the experience replay buffer D
    for epoch $k = 1, ..., K$ **do**
      Sample N tuples from D
      Normalize the cumulative reward to get $r_{new}(s_t, a_t)$
      Use the target network to calculate the state value and state-action value at the next moment as the update target of the value network
      Input the normalized cumulative reward into the value network to obtain the predicted value
      Update the critic network based on the calculation of the loss functio
      Sample action $\tilde{a}_t$ with the reparameterization technique, and then update the actor network with the loss function
      Update the entropy regularization coefficient α
      Update the target network
    end for
  end for
end for

4. Experiments

OpenAI Gym [18] is an open-source reinforcement learning toolkit designed to facilitate the comparison of research and algorithms. It offers a variety of benchmarking environments, such as the Atari 2600 game, and provides a common interface for third-party tasks. This provides a convenient platform and challenging tasks for developers and researchers in the field of artificial intelligence. As shown in Figure 1, the Gym CarPole-v1, Acrobot-v1, LunarLander-v2, and MountainCar-v0 were selected as the environments in this paper, and the AN-SAC algorithm is experimentally compared with the SAC algorithm in different task environments.

In this experiment, the operating system is Win11, the language development environment is Python 3.8, and the neural network framework is built with Pytorch 2.2. In order to compare the improvement effect of the AN-SAC algorithm and make its hyperparameters completely consistent with those of the SAC algorithm, Table 1 sets the specific values of the hyperparameters, uses ReLU for all activation functions, and uses Adam as the optimizer to adjust the model parameters.



(a) CartPole-v1 (b) Acrobot-v1

(c) LunarLander-v2 (d) MountainCar-v0

FIGURE 1. Gym environment

Table 1 Hyperparameters settings.

Hyperparameters	value	parameter description
num_epochs	100	number of training rounds
capacity	500	experience buffer capacity
actor_lr	0.001	policy network learning rate
critic_lr	0.001	value network learning rate
alpha_lr	0.001	trainable parameter learning rate
tau	0.01	soft-update parameters
gamma	0.9	discount factor

In this paper, the performance and stability of the algorithm are evaluated by comparing the cumulative rewards of the algorithm in different environments, and the learning curve is shown in Figure 2. Compared with the traditional SAC algorithm, AN-SAC uses the normalized cumulative reward to train the value network, which enables the algorithm to better adapt to the changes in reward distribution and improve the performance of the algorithm. As an advanced version of the SAC algorithm, AN-SAC has different degrees of improvement compared with SAC in all the tasks for comparison. As can be seen from Figure 2(a), in the task CartPole-v1, the reward value of AN-SAC is higher than that of SAC most of the time, especially between the 69th and 77th epochs, the reward value of AN-SAC rises sharply to the highest point, while SAC is relatively low. This means that AN-SAC has better adaptability and strategy adjustment

during certain periods. In Figures 2(b) and (c), the reward value of AN-SAC is also higher than that of SAC most of the time, although there is no peak like 2(a) in these two tasks, the reward curve of AN-SAC is relatively stable and fluctuates less, while the reward curve of SAC fluctuates greatly, which means that AN-SAC is better in terms of stability. In general, the performance of AN-SAC in these tasks is higher than that of SAC most of the time, and the curve is relatively stable and fluctuates less than that of the SAC algorithm. Therefore, on the whole, the stability and performance of AN-SAC are better than those of the SAC algorithm.

(a) CartPole-v1 (b) LunarLander-v2

(c) Acrobot-v1 (d) MountainCar-v0

FIGURE 2. Training curves of AN-SAC and SAC.

Table 2 Final performance of AN-SAC and SAC.

Task	CartPole-v1		Acrobot-v1		LunarLander-v2		MountainCar-v0	
	AN-SAC	SAC	AN-SAC	SAC	AN-SAC	SAC	AN-SAC	SAC
Mean	**33.65**	18.69	**-222.41**	-364	**-134.65**	-219.78	**-8486**	-65634
median	**32**	14.5	**-222.5**	-326	**-152.0**	172.42	**-19306**	-55721
std	16.07	**9.47**	**92.56**	133.95	**41.27**	126.73	**7107**	61316

Table 2 demonstrates the mean, median, and standard deviation of the cumulative rewards obtained by the AN-SAC algorithm and the SAC algorithm on the corresponding tasks at the end of the above task training, and the algorithm data with good performance in each task is bolded. It is clearly that the AN-SAC algorithm has obvious advantages in several other tasks except for the large standard deviation of the task CartPole-v1. This demonstrates that the

performance of AN-SAC is better than that of the SAC algorithm in these tasks, and its stability is also better in Acrobot-v1, LunarLander-v2, and MountainCar-v0.

5. CONCLUSION

In this paper, an AN-SAC algorithm was proposed, which integrates an adaptive normalized reward mechanism into the update step of the value network, which can improve the stability and performance of the value network, thereby improving the performance of the whole algorithm. The Beta distribution in the range of $(-1, 1)$ was used to initialize the neural network, which helps the neural network to converge faster and accelerate the training of the neural network. In this paper, the effectiveness of the algorithm was verified in four environments, and the experimental results were provided to compare with the traditional SAC algorithm. AN-SAC demonstrates good performance in processing tasks and tends to be more stable in the training process.

Funding

This work was supported by the Research Foundation for Advanced Talents 2022BS073 of Henan University of Technology, the Key Scientific Research Projects of Higher Education Institutions in Henan Province under Grant No. 24A520014 and No. 24B520006, the Henan Provincial focus on research and development Project (No. 231111220700 and No. 241111110 200), the National Natural Science Foundation of China (No. 61473114), the Science and Technology Project of Science and Technology Department of Henan Province, China (No. 242102210016, No. 242102220121, and No. 212102210149), the Open Fund of Key Laboratory of Grain Information Processing and Control (Henan University of Technology), Ministry of Education (No. KFJJ2023015). In addition, this work was also supported by the College Students' Innovative Entrepreneurial Training Plan Program PX-38244826.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, et al., Human-level control through deep reinforcement learning, Nature, 518 (2015) 529-533.

[2] D. Silver, J. Schrittwieser, K. Simonyan, et al., Mastering the game of go without human knowledge, Nature, 550 (2017) 354-359.

[3] D. Silver, A. Huang, C.J. Maddison, et al., Mastering the game of Go with deep neural networks and tree searc, Nature, 529 (2016) 484-489.

[4] K. Arulkumaran, A. Cully, J. Togelius, Alphastar: An evolutionary computation perspective, Proceedings of the Genetic and Evolutionary Computation Conference Companion, 314-315, 2019.

[5] X. Zhu, R. Zhang, W. Zhu, MDMD options discovery for accelerating exploration in sparse-reward domains, Knowledge-Based Systems, 241 (2022) 108151.

[6] L. Gao, T. Ma, K. Liu, et al., Application of improved Q-Learning algorithm in path planning, J. Jilin Univ. 36 (2018) 439-443.

[7] K. Zhu, T. Zhang, Deep reinforcement learning based mobile robot navigation: A review, Tsinghua Sci. Tech. 26 (2021) 674-691.

[8] Y. Deng, F. Bao, Y. Kong, et al., Deep direct reinforcement learning for financial signal representation and trading, IEEE Trans. Neural Networks Learn. Sys. 28 (2016), 653-664.

[9] M.A.H. Dempster, V. Leemans, An automated FX trading system using adaptive reinforcement learning, Expert Sys. Appl. 30 (2006) 543-552.

[10] J.E. Moody, M. Saffell, Y. Liao, et al., Reinforcement learning for trading systems and portfolios, KDD'98: Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, 279-283, 1998.

[11] E. Ie, V. Jain, J. Wang, et al,. Reinforcement learning for slate-based recommender systems: A tractable decomposition and practical methodology. arxiv preprint arxiv:1905.12767, 2019.

[12] I. Munemasa, Y. Tomomatsu, K. Hayashi, et al., Deep reinforcement learning for recommender systems, 2018 International Conference on Information and Communications Technology (ICOIACT), 2018, DOI: 10.1109/ICOIACT.2018.8350761

[13] B. Baker, I. Kanitscheider, T. Markov, et al., Emergent tool use from multi-agent autocurricula, arxiv preprint arxiv:1909.07528, 2019.

[14] R. Mendonca, A. Gupta, R. Kralev, et al., Guided meta-policy search, Advances in Neural Information Processing Systems 32 (NeurIPS 2019), Vancouver, 2019.

[15] Y. Efroni, G. Dalal, B. Scherrer, et al., How to combine tree-search methods in reinforcement learning, Proceedings of the AAAI Conference on Artificial Intelligence, Article No. 429, 3494-3501, 2019.

[16] K. Ciosek, Q. Vuong, R. Loftin, et al., Better exploration with optimistic actor critic, Advances in Neural Information Processing Systems 32, Article No. 160, pp. 1787-1798, 2019.

[17] T. Haarnoja, A. Zhou, P. Abbeel, et al., Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, International Conference on Machine Learning, PMLR, 80 (2018) 1861-1870.

[18] G. Brockman, V. Cheung, L. Pettersson, et al., Openai gym, arxiv preprint arxiv:1606.01540, 2016.