An Advanced Actor-Critic Algorithm for Training Video Game AI

ZhongYi Zha[1], XueSong Tang[2], and Bo Wang[1(✉)]

[1] Key Laboratory of Ministry of Education for Image Processing and Intelligent Control, Artificial Intelligence and Automation School, Huazhong University of Science and Technology, Wuhan 430074, China
wb8517@hust.edu.cn

[2] College of Information and Science, Donghua University, Shanghai, China

Abstract. This paper presents an improved deep reinforcement learning (DRL) algorithm, namely Advanced Actor Critic (AAC), which is based on Actor Critic (AC) algorithm, to the video game Artificial intelligence (AI) training. The advantage distribution estimator, Normal Constraint (NC) function and exploration based on confidence are introduced to improve the AC algorithm, in order to achieve accurate value estimate, select continuous spatial action correctly, and explorate effectively. The aim is to improve the performance of conventional AC algorithm in a complex environment. A case study of video game StarCraft II mini-game AI training is employed. The results verify that the improved algorithm effectively improves the performance in terms of the convergence rate, maximum reward, average reward in every 100 episode, and time to reach a specific reward, etc. The analysis on how these modifications improve the performance is also given through interpretation of the feature layers in the mini-games.

Keywords: Deep reinforcement learning · Video game AI · Advanced Actor Critic · Advantage distribution estimator · Normal constraint

1 Introduction

In 2016, Alpha Go [1], the computer program developed by Google's DeepMind team, defeated the 18-time world champion Lee Sedol in a five-game Go match and manifested the power of the artificial intelligence (AI). In 2017, Alpha Go Zero [2], which adopted the deep reinforcement learning algorithm (DRL) and defeated the previous generation of Go in a short time, dominated the Go match and revealed the power of the DRL. Nowadays, DRL has been widely applied to industries in different fields, e.g., manufacture, natural language processing, medical image processing, intelligent drive and video game AI design, etc. Among these areas, AI design of video games is of particular interest to researchers, as it provides a convenient test platform for investigating machine learning algorithm performance in complex environments.

H. Zhang et al. (Eds.): NCAA 2020, CCIS 1265, pp. 368–380, 2020.
https://doi.org/10.1007/978-981-15-7670-6_31

With the development of DRL, the end-to-end feature extraction method breaks the conventional manual feature extraction method, so that AI can complete the game task without intervention of humans. In 2013, Mnih et al. employed DRL to play video games. The DRL based AI even surpassed top human players in some Atari games [3]. However, these Atari games are generally too simple compared with practical applications in real life. Before long, Deep Mind team opened the StarCraft II learning environment (SC2LE) [4] as a new test platform for DRL studies. The SC2LE involves dynamic perception and estimation, incomplete information game and multi-agent cooperation problems, resulting in a more practical test environment, where decision making becomes much closer to the real-life situations.

Although the latest StarCraft II AI has been able to beat top human players in certain circumstances [5], the AI training required a vast number of episodes to achieve satisfying performance, due to the limits of the employed conventional Actor Critic (AC) algorithm ([6–8]). For example, the advantage estimator returns single value that results in limited accuracy, and the random action selection lacks effectiveness, etc. In this paper, an improved design of Actor Critic (AC) algorithm, namely Advanced Actor Critic (AAC), is proposed for better convergence of AI training. A distributional advantage estimator is employed to estimate the distribution of advantage in a certain state. In addition, a normal constraint, which means the loss function of spatial selection based on the normal distribution, as well as an exploration method based on confidence are introduced. As to the neural networks framework, the fully-convolution with long short-term memory networks (CNN LSTM) [9] is selected according to the Deep Mind approach [4]. Additionally, the sparse reward [10] is addressed by reward shaping. Finally, a combined model based on Rainbow [11] (a combination of Double Q Learning [12], dueling DQN [13], Priori-tized Experience Replay [14], distributional reward [15], noisy net [16]) is adopted, where the proposed algorithm is embedded. Such an approach is applied to the SC2LE mini-games to verify that AI can learn faster, play better and have stronger robustness with the ACC based training program. The main contribution of this paper is to improve the conventional AC algorithm, which leads to better performance in complex environments, and to propose a regional updating action mode instead of independent point updating for intelligent control in high-dimensional space.

2 Background

The SC2LE is integrated by Deep Mind [4]. In DRL, state, action and reward are the most important elements. In StarCraft II mini-games, these elements are represented by map features and mini-map features.

2.1 Environment

Reward. The reward and penalty in each mini-game are shown in detail in Table 1.

Table 1. Reward and Penalty in mini-games.

Mini-map name	Reward	Penalty
Move to beacon	+1 when the marine reaches a beacon	None
Collect mineral shards	+1 when the marine collects the mineral shards	None
Find and defeat Zerglings	+5 when the marine kills the Zergling	−1 when the marine is killed
Defeat Zerglings and Banelings	+5 when the marine kills the Zergling or Baneling	−1 when the marine is killed

State. In StarCraft II games, the environment has been processed by the interface, outputting several feature layers about the current state (see Fig. 1).

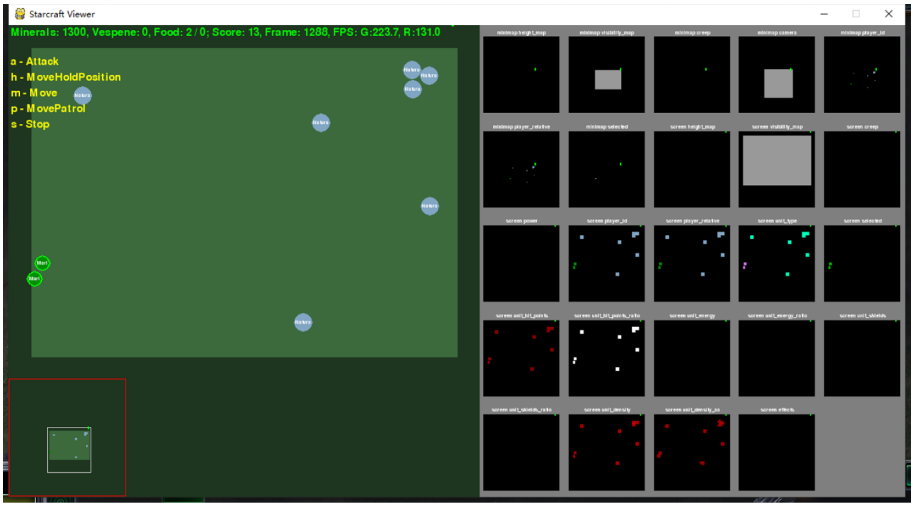

Fig. 1. Feature layers of the current state, which includes all simplified information of the mini-games.

During training, the size is the resolution of these feature images. Generally, the height and width are set to N, i.e., the spatial pixel observed and operated is $N \times N$. Deep Mind pointed out that $N \geq 64$ [4] is required when performing any micro operation in the game. However, for mini-games, $N = 32$ is proved sufficient.

Among these $N \times N$ pixels, each feature layer represents specific feature, such as player ID, unit ID, war fog, unit attribute, etc. Some belong to scale, while others belong to catalog.

In addition, the difference between the mini-map and the screen is that the screen is only a part of the mini-map camera, but the analysis of the unit is

clearer. After processing, the mini-map is transformed into $F_{minimap} \times N \times N$, and the screen is transformed into $F_{screen} \times N \times N$. $F_{minimap}$ and F_{screen} represent the quantity of features that mini-map or screen has.

Action. The instruction set of StarCraft II is very large. Through the interface, each action is divided into two parts (Spatial Action and Non-Spatial Action) and output to the environment. The whole action is: $A = (A_{non-spatial}, [A_{add}, A_{spatial}])$. Among them, Spatial Action represents that AI clicks on a position in the screen, and its value range belongs to the pixel size of the environment, that is, $A_{spatial} = (x, y) \in N \times N$. Non-Spatial Action represents operation instructions, such as select, move, attack, build, view, etc. Its value range is the game instruction set of the whole StarCraft II pairs, and at the same time, it is subject to the actions available in the current environment, that is, $A_{non-spatial} \in A_{available} \cap A_{all}$. $A_{available}$ involves actions that can be performed in a certain state and A_{all} contains all action instructions in StarCraft II. Especially, non-spatial actions are necessary while spatial actions are optional. For example, if an unit is selected by AI, only the non-spatial action is output, which means the spatial action becomes needless. A_{add} is the type of $A_{non-spatial}$ operation. When $A_{non-spatial}$ and $A_{spatial}$ are determined, A_{add} is generated automatically.

2.2 Actor Critic and Fully-Convolution with LSTM

Actor Critic. The policy adopted by the agent is the distribution of all state and action trajectories interacting with the environment. How this distribution takes the trajectory depends on the parameter θ. One of the trajectories is $\tau = (s_1, a_1, s_2, a_2, ..., a_t, s_t)$.The probability of getting the trajectory is $p_\theta(\tau) = \prod_{t=1}^{T} p_\theta(a_t \mid s_t)$. The relationship between the generation of environment and the parameter θ of trajectory can be ignored.

In order to get the best performance of an agent, the optimal goal is: $\theta = \arg\max_\theta E_{\tau \sim p_\theta}(R_\theta(\tau))$. $R_\theta(\tau)$ is the cumulative reward of τ, and the gradient is:

$$\nabla R_\theta(\tau) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} (\sum_{s=t}^{T_n} A_t \nabla log(p_\theta(a_t^n \mid s_t^n))) \tag{1}$$

In this paper, spatial policy net and non-spatial policy net are independent (π_θ represents the non-spatial policy net while π'_θ represents spatial policy net, and ρ indicates whether spatial action is valid or not), therefore the loss function in each episode is:

$$Loss_{\pi_\theta} = \sum_{t=1}^{n} A_t [log(\pi_\theta(a_t \mid s_t)) + \rho log(\pi'_\theta(a_t \mid s_t))] \tag{2}$$

Combine Actor Critic with CNN LSTM. The following figure shows the framework of interaction between the intelligent algorithm and StarCraft II Environment(see Fig. 2).

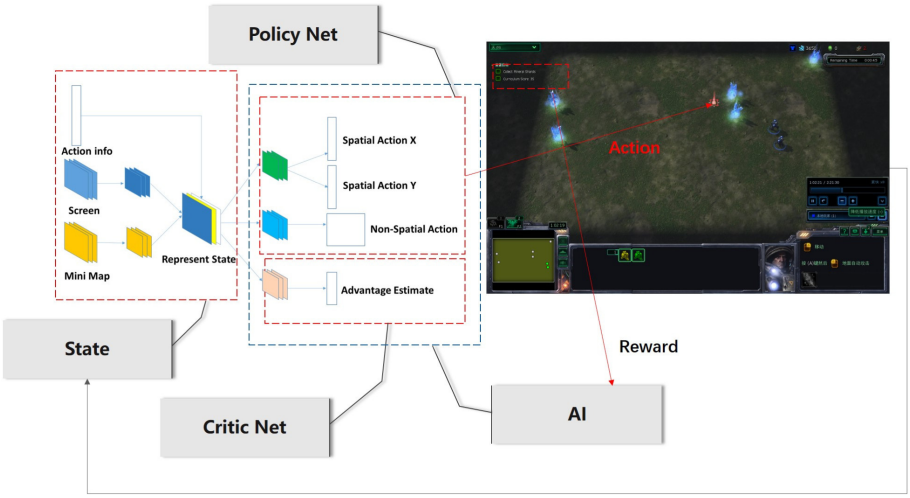

Fig. 2. Interaction between DRL based AI and StarCraft II environment. The inputs of the network are (1) available action (2) screen features (3) mini-map features. Outputs are (1) X and Y of the spatial action (2) non-spatial action (3) estimate of advantage. These outputs are all distributions.

3 Reprocessing of Environment Feature

Several feature layers have been extracted from the environment of the StarCraft II games. However, some of these layers are scale, while some are catalog, which means the distribution of values is very uneven. In addition, some feature layers are almost useless in mini-games. In order to train neural network better and faster, it is necessary and significant to reprocess these feature layers.

These layers can be simply divided into catalog value layers and scalar value layers. The catalog value layers include player ID, friend, visibility, creep, etc. Generally, these values are small. The scalar value layers include unit type, unit health, terrain level, and armor value. The prior knowledge of a StarCraft II player can be exploited to choose more helpful map feature layers and discard useless ones. Among these layers, the feature layers of mini-map are chosen as: visibility, camera, player ID, player relative, selected and the screen feature layers are: visibility, player ID, player relative, selected, HP, energy.

After the layer selection, the layers are numerically processed. In the catalog value layers, the values are standardized between [0, 1] to facilitate the training efficiency of neural networks. In the scalar value layers, the threshold is set to half of its maximum value. A sigmoid function is adopted to change the value classification into a state value. For example, if a unit has more than 50% of its maximum hit point (HP), it is considered to be healthy, otherwise be weak. After this reprocessing, the ranges of all layer values are compressed to be [0, 1].

4 Advanced Actor Critic (AAC)

The AC algorithm is improved from several perspectives: the updated data can break the correlation of time; the strategy network is more robust; the evaluation network is more accurate. The improvements are explained as follows.

4.1 Distributional Advantage

In AC algorithm, critic network is responsible for outputting the advantage estimate based on a certain policy. The accuracy of the estimate determines the update efficiency of policy net. As a result, a distributional network is designed to get more information. In this network, the maximum and minimum values are determined by the background conditions, and the probability of the advantage is handled by the last layer of convolution network plus softmax function. When updating, the sample $\widehat{R}$ obtained from Monte Carlo difference (MD) or temporal difference (TD) is transformed into a pulse function $D(\widehat{R})$ (avoiding None by using Clamp function) and then KL divergence is used to evaluate the loss of advantage estimates (see Fig. 3).

In this case, the advantage A_t and the value loss function $Loss_{V_\theta}$ can be formulated as follows:

$$A_t = E_{p(V_t)\sim V_\theta}(V_\theta(s_t)) \tag{3}$$

$$Loss_{V_\theta} = D_{KL}[D(\widehat{R}) \parallel V_\theta(s_t)] \tag{4}$$

4.2 Exploration Based on Confidence

One of the important problems in DRL is exploration. In mini-games, the exploration ability of agent in the early stage significantly influences its performance. Without exploration, the agent keeps taking the same action if the reward is always positive. Thus, it might be trapped by the local optimal solution and ignore other actions that may result in huge potential reward. Therefore, a good agent needs to have proper exploration ability. In contrast to the conventional AC algorithm exploration method, the maximum probability of the non-spatial policy, namely the confidence, is adopted to explore the environment. ω represents the lowest level of confidence in the conduct of random exploration and AI still explores based on $\epsilon - greedy$ to prevent falling into the local optimal solution. The less confidence the AI has, the more exploration is encouraged, and vice versa.

$$A_{non-spatial} = \begin{cases} \arg\max_\theta \pi_\theta(a_t \mid s_t), else \\ random\, choice, max(\pi_\theta(a_t \mid s_t)) < \omega \ or \ \epsilon < 0.1 \end{cases} \tag{5}$$

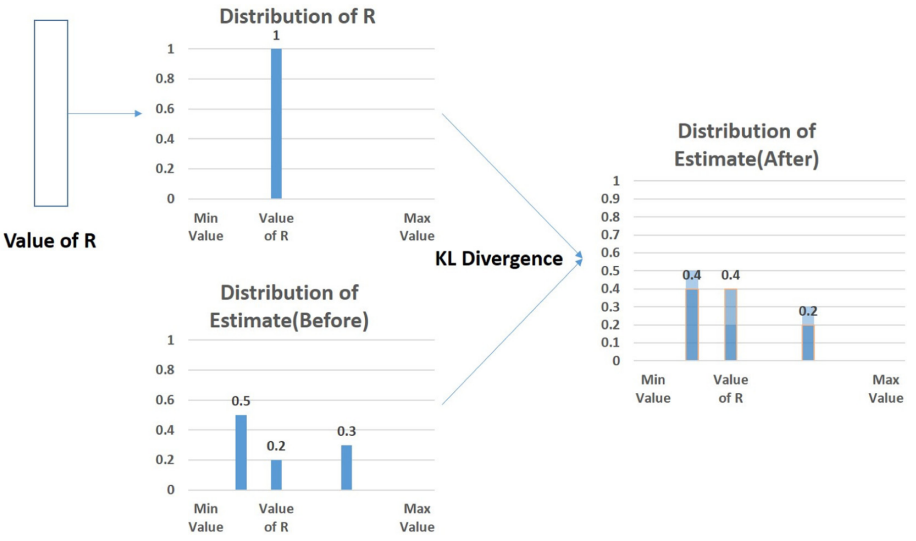

Fig. 3. The process of converting sample $\widehat{R}$ into pulse function and computing KL divergence with advantage estimate distribution.

4.3 Normal Constraint

In the environment of StarCraft II games, the most difficult problem is the huge continuous action space. In AAC algorithm, the policy net of AAC consists of three parts: spatial policy net X, spatial policy net Y and non-spatial policy net. The update mode of non-spatial policy net is introduced in Sect. 2.2. Given the particularity of action space in StarCraft II games, the update of spatial policy net X and spatial policy net Y introduce an additional function for the sake of convergence speed, namely the Normal Constraint. To enhance the exploration ability, loss entropy is usually adopted as the loss function in AC network. In the case of large continuous action space and sparse reward, if the loss entropy weight is not well designed, it is easy to enter policy net with the same output probability. In order to solve this problem, a Normal Constraint is hereby designed. In StarCraft II games, the actions are selected in a continuous space, and the selection probability around a specific point should be considered similar. Therefore, when updating an action, the nearby actions should be updated accordingly. In this case, AI can understand the high-dimensional action space regionally. Finally, when converging, the output of spatial policy is a stable distribution that approximately satisfies the Normal Distribution (see Fig. 4, Fig. 5).

The probability of the selected spatial action, i.e., the coordinate in the environment, is used as the weight of the NC loss. The greater probability implies more centralized policy, and the probability of its surrounding will be further enhanced or reduced when updated. Combined with formula (3), the NC loss function is:

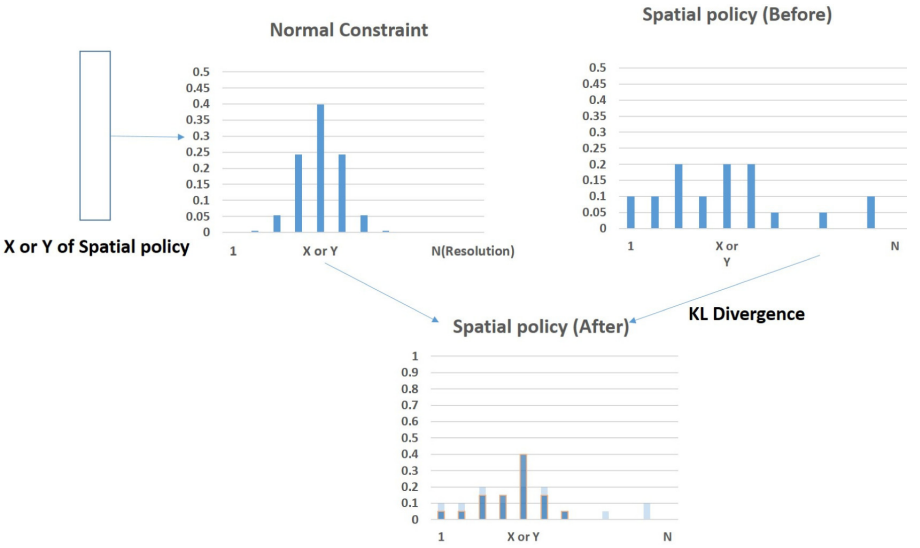

Fig. 4. How NC works.

$$Loss_{NC} = \sum_{t=1}^{n} A_t max(\pi'_\theta) D_{KL}[NC(a_t) \parallel \pi'_\theta(s_t)] \tag{6}$$

Finally, the whole network is updated according to formula (2) (4) (6).

4.4 Reward Shaping

The reward of StarCraft II is very sparse, which requires the agent to have a stronger exploration ability. During the implemented trainings, a negative reward is added to every moment to motivate AI to take actions. Besides, the reward values are standardized to be within the range [0, 1]. Such a reward shaping is very effective in a sparse environment.

5 Result

The agent based on AAC network or A3C network[1] is applied to 4 mini-games, and then the average score in every 100 episode as well as maximum score in 20K steps are compared. In the initial stage of training, i.e., observation episodes, the update of policy net is stopped. Since the training of policy net depends on critic net, critic net must be trained first in this observation.

A part of hyper parameters are shown in Table 2, and the Adam optimizer is adopted [17].

[1] The adopted A3C network refers to the project at https://github.com/xhujoy/pysc2-agents.

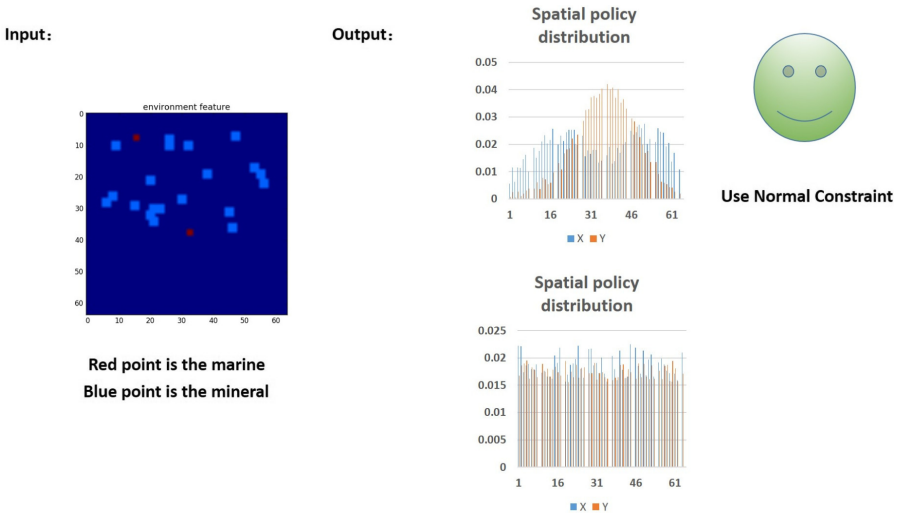

Fig. 5. Comparison of policy net with and without Normal Constraint (the figure above uses NC, while the figure below does not).

Table 2. Hyper parameters.

Hyper-parameter	Value	Notes
Observation episodes	200	Stop updating policy net
Learning rate	10^{-5}	
γ	0.99	Discount of reward
N	32	Resolution of mini-map and screen
Step	60	Steps in each iteration
Distributional advantage	$[-1, 24]$	Max value and Min value (different in each mini-game)
Distributional atoms	51	Numbers in advantage distribution
Adam epsilon	10^{-8}	
σ	2	Sigma of normal constraint
Addition reward	$\frac{-0.0001}{step}$	Reward shaping
ω	0.1	Base of confidence

6 Discussion

Mini-games in StarCraft II are divided into two categories based on their characteristics to test the performance of AAC and A3C. One category mainly focuses on spatial action and another is small scale combat game(see Fig. 6, Fig. 7). After 20K episodes, the average and max score achieved by agent based on AAC are both higher than agent based on A3C across the four mini games (see Table 3).

The key to get a better score depends on the performance of critic in Actor Critic algorithm, namely the accuracy of the advantage estimator. In a certain state (see Fig. 8), the Critic net in AAC estimates the reward by the expected

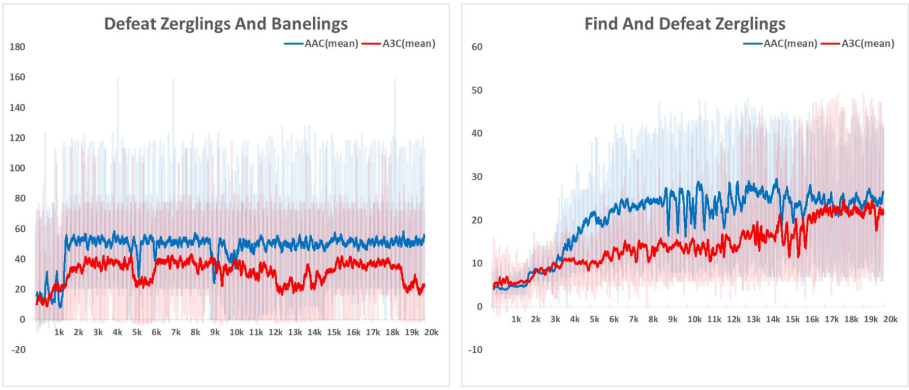

Fig. 6. Performance across 2 mini-games which mainly focus on spatial action for AAC and A3C.

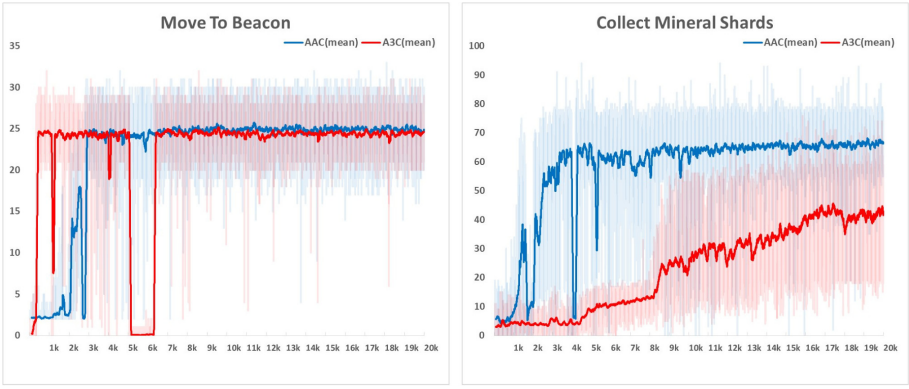

Fig. 7. Performance across 2 small scale combat games for AAC and A3C.

value 16.4 and the true value is 25. On the contrary, the Critic net in A3C estimates the reward by 6.1 and the true value is 24. Thus, the distributional estimator in critic net collects and computes the recent expected reward of a state, which contains more information and is closer to true reward than the estimate of single-valued estimator in A3C. During the experiment, it is proved that the more atoms distribution has, the more accurate the estimate is, because the distribution is more elaborate. However, more atoms need more neurons in the critic net and more time in training. In this paper, 256 feature neurons with 51 atoms are chosen [15].

The convergence speed of AAC and A3C is shown as the episode that the agent achieves scores around the best. It reveals like that the convergence speed of AAC is faster than that of A3C except in the Move to Beacon game at first

Table 3. Best score in each mini-game.

Agent	Metric	Move to beacon	Collect mineral shards	Find and defeat Zerglings	Defeat Zerglings and Banelings
AAC	BEST MEAN	26	68	29	59
	MAX	33	94	49	159
	Std of last 1k episodes	1.8	7.3	6.9	21.5
A3C	BEST MEAN	24	45	25	43
	MAX	31	76	45	118
	Std of last 1k episodes	2.4	13.6	8.6	24.5

glance. However, it should be noticed that the agent based on A3C achieved extremely bad score throughout the middle of the experiment (from episode 5k to 6k), and the reason is that the relatively small probability of spatial action leads to the hesitation but not confidence of agent (see Fig. 8).

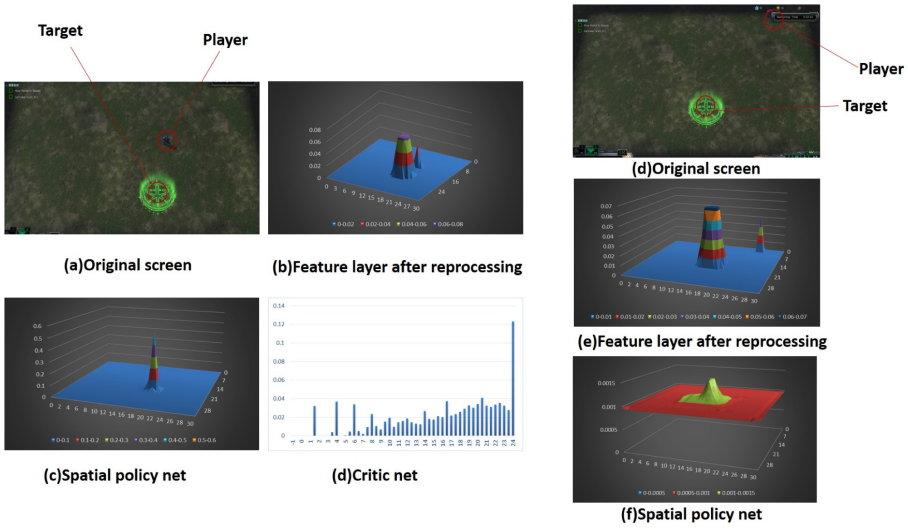

Fig. 8. Performance of AAC and A3C on Move to Beacon in episode 5k, (a) (b) (c) (d) is generated by AAC while (d) (e) (f) is generated by A3C.

In conclusion, around episode 5k, the agent based on A3C indecisively chose the right target due to the max but not dominant probability, and it indicates that the policy net has not converged. As a result, the convergence speed depends not only on when the agent performs a stable and good score, but also on the distribution of policy net which represents the confidence of the agent. In AAC, Normal Constraint encourages and guides the agent in the right direction. Additionally, this positive feedback allows the agent to explore in a certain area of huge action space with more confidence, so as to speed up the convergence. As a

result, agent based on AAC converges faster than A3C across all games. Besides, probability distribution of policy net has not converged in two small scale combat games because training episodes are not enough.

In future work, more episodes and distribution atoms in advantage estimator will be applied in more difficult mini-games, full games and other fields. In addition, whether AAC can perform even better than other AC algorithm in more complex environments will also be tested.

Funding Statement. This research was supported by National Key Research and Development Program of China (Project No. 2018YFF0300300) and National Natural Science Foundation of China (Project No. 61803162).

References

1. Anthony, T., Tian, Z., Barber, D.: Thinking fast and slow with deep learning and tree search. In: Advances in Neural Information Processing Systems, pp. 5360–5370 (2017)
2. Silver, D., et al.: Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815 (2017)
3. Mnih, V., et al.: Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
4. Vinyals, O., et al.: StarCraft II: a new challenge for reinforcement learning. arXiv preprint arXiv:1708.04782 (2017)
5. Vinyals, O., et al.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (2019)
6. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
7. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937 (2016)
8. Wu, Y., Mansimov, E., Grosse, R.B., Liao, S., Ba, J.: Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In: Advances in Neural Information Processing Systems, pp. 5279–5288 (2017)
9. Xingjian, S., Chen, Z., Wang, H., Yeung, D.Y., Wong, W.K., Woo, W.C.: Convolutional LSTM network: a machine learning approach for precipitation nowcasting. In: Advances in Neural Information Processing Systems, pp. 802–810 (2015)
10. Grześ, M., Kudenko, D.: Online learning of shaping rewards in reinforcement learning. Neural Netw. **23**(4), 541–550 (2010)
11. Hessel, M., et al.: Rainbow: combining improvements in deep reinforcement learning. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
12. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)
13. Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., De Freitas, N.: Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581 (2015)
14. Hou, Y., Liu, L., Wei, Q., Xu, X., Chen, C.: A novel DDPG method with prioritized experience replay. In: 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 316–321. IEEE (2017)

15. Bellemare, M.G., Dabney, W., Munos, R.: A distributional perspective on reinforcement learning. In: Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 449–458. JMLR.org (2017)
16. Fortunato, M., et al.: Noisy networks for exploration. arXiv preprint arXiv:1706.10295 (2017)
17. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)