Applying Online Expert Supervision in Deep Actor-Critic Reinforcement Learning

Jin Zhang, Jiansheng Chen(✉), Yiqing Huang, Weitao Wan, and Tianpeng Li

Department of Electronic Engineering, Tsinghua University, Beijing, China
{jinzhangl6, huang-yql7, wwtl6}@mails.tsinghua.edu.cn,
jschenthu@mail.tsinghua.edu.cn, lllttppp@qq.com

Abstract. Deep reinforcement learning (DRL) has been showing its strong power in various decision making and controlling problems, e.g. Atari games and the game of Go. It is inspiring to see DRL agents to outperform even human masters. However, DRL algorithms require a large amount of calculation and exploration, making DRL agents hard to train, especially in problems with large state and action spaces. Also, most DRL algorithms are very sensitive to hyper parameters. To solve these problems, we propose A3COE, a new algorithm combining the A3C algorithm with online expert supervision. We applied it on mini-games of the famous real-time-strategy game StarCraft II. Results show that this algorithm greatly improved the agent's performance with fewer training steps while acquiring more stable training processes with a greater range of hyper parameters. We also proved that this algorithm works even better with curriculum learning.

Keywords: Deep reinforcement learning · Expert supervision
A3C · Curriculum learning

1 Introduction

1.1 Deep Reinforcement Learning and Its Weakness

Deep reinforcement learning, combining neural networks with traditional tabular reinforcement learning to improve its generalization ability, has been proved successful in multiple decision making problems: Atari games [1], the game of Go [2], and physic simulators [3]. However, DRL algorithms require enormous training steps and exploration data, and this problem becomes more serious when the state and action space is large. For example, Deepmind's Rainbow algorithm requires over 200 million frames of Atari game play to train an agent [1], and AlphaGo Lee used as many as 48 TPUs to train [2].

Although the convergence of traditional reinforcement learning algorithms like Q-learning have been proved mathematically [11], there is no evidence that these algorithms remain convergent when combined with neural networks. Also, evidence

The first author of this paper is an undergraduate

The original version of this chapter was revised: The name and the email of the fourth author were incorrect. The correction to this chapter is available at https://doi.org/10.1007/978-3-030-03335-4_45

J.-H. Lai et al. (Eds.): PRCV 2018, LNCS 11257, pp. 469–478, 2018.
https://doi.org/10.1007/978-3-030-03335-4_41

showed that most popular DRL algorithms are very sensitive to hyper parameters, and even the randomly initialized network weights can affect the training process greatly [12]. This is definitely what we do not want to see: a reliable algorithm should be robust and reproducible. A natural idea is to make use of demonstrations, as human learn faster and better with demonstrations than exploring alone.

1.2 StarCraft II and Its Large Decision Space

Our experiment takes StarCraft II as the training environment and we use Deepmind's PYSC2 training platform. In PYSC2, the game's state is represented by 24 feature maps, with 17 main screen features, and the rest 7 minimap features. The game's actions are organized into 524 different action functions, each taking 0 to 2 parameters, and there are more than 10^8 possible actions in total. Compared with the game of Go, which has a state space of 19 * 19 * 5 (according to the training settings of [2]) and an action space of approximately 361, StarCraft II is obviously more complex. So we conclude that StarCraft II is a hard reinforcement learning problem, as it has a vast decision space, which will require considerable exploration to train an agent.

It is worth mentioning that StarCraft II is also difficult for its long time scale, strategy complexity and partial observations, but we do not discuss it because we avoid these problems in our environment choosing, to focus on the point we want to solve: the large decision space.

2 Backgrounds and Related Works

DRL algorithms are based on the concept of Markov Decision Processes (MDPs) [13]. An MDP can be described by a tuple $\{\mathcal{S}, \mathcal{A}, \mathrm{P}, \mathrm{R}, \gamma\}$, with $\mathcal{S}$ the set of states, $\mathcal{A}$ the set of actions, $P(s'|s,a)$ the transition probabilities, $R(s,a)$ the set of rewards, and a reward discount factor γ. Our aim is to find a good policy π(s), which gives out the next action given the current state. Based on the rewards received each step, we can define the Q-value function and the V-value function as follows:

$$\mathrm{V}^{\pi}(s) = \sum_{s' \in S} P(s'|s, \pi(s))[R(s, \pi(s)) + \gamma V^{\pi}(s')] \tag{1}$$

$$\mathrm{Q}^{\pi}(s,a) = \sum_{s' \in S} P(s'|s,a)[R(s,a) + \gamma V^{\pi}(s')] \tag{2}$$

And these equations are known as the Bellman Equations. The aim of reinforcement learning is to find the optimal policy π^*, and π^* can be acquired with the following Bellman optimality equations:

$$\mathrm{V}^{*}(s) = max_a E[R(s,a) + \gamma V^{*}(s')] \tag{3}$$

$$Q^{*}(s,a) = E\left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) max_{a'}\ Q^{*}(s',a')\right] \tag{4}$$

Value-based DRL methods (e.g. DQN [14]) try to solve Eq. (4) and give optimal actions according to Q-values, while actor-based methods try to directly give the optimal action. In our experiment we use A3C, an actor-critic algorithm which contains a critic network to estimate V-values and an actor network to give actions. The critic network's loss and actor network's loss are designed as follows:

$$\text{loss}_\text{c} = (V_{actual} - V_{estimate})^2 \tag{5}$$

$$\text{loss}_\text{a} = -(V_{actual} - V_{estimate}) * \log(p(a|s)) + \beta * E_a \tag{6}$$

with $p(a|s)$ the possibility of taking action a under state s, E_a the entropy of the actions' possibilities ($\sum_{a \in A} p(a|s) * \log(p(a|s))$) and β a positive hyper parameter to encourage exploration.

In August of 2017, Deepmind released the PYSC2 platform with some baseline results using A3C. Although they tried various network structures, all of them took up to 300 million training steps to achieve good behavior in mini-games [8]. What's more, they hardly made any progress in full game and could not win the easiest built-in AI, even with shaped rewards. However, agents trained with human replay data managed to perform better and produced more units. That suggested that imitation learning may be a 'promising direction' for AI training. Their work proved that StarCraft II is now a challenging problem for DRL.

The idea of implementing demonstration data to pre-train network has been widely used, for example, on DQN [4], A3C [5], DDPG and ACER [6]. However, these work focuses on offline demonstration data, and little work looks on online expert data. In some cases, it may be convenient to give expert policy in the training process, e.g. mini-games in StarCraft II (you can easily hard-code an agent to give expert policies) and automatic driving (provided with images of roads, people can easily give the right action). Under these circumstances, training will be faster if we introduce online expert supervision than offline pre-training as it is more straightforward.

Curriculum learning is a training method to help speed up training. Human beings learn better if they learn easier tasks before hard ones, so it is possible to train networks with easier tasks in the beginning to acquire better performance. Results have proved that curriculum learning helps get better results in shape recognition and language modeling using neural networks [9]. In DRL, this method has also been used to train agents. For example, in the first-person shooting game Doom, curriculum learning has helped agents to perform better, and one agent using A3C and curriculum learning won the champion of Track1 in ViZ-Doom AI Competition 2016 [10].

3 Methods

3.1 Environment Settings

In our discussion we apply Collect Mineral Shards, a simple StarCraft II mini-game, as our training environment. In this task, the agent controls two units to reach and get the 20 mineral shards on the map to get scores. The units and mineral shards are all

randomly placed on the map. We only consider controlling the two units together (that is, the two units receive the same instructions at the same time). We choose this task because it simplifies the state and action space, lowering the total number of actions to 4097 (64 * 64 possible destinations for the agent to move units to and one action to select the two units). It is also a small map which excludes camera moving, fog of war and partial observation. Also, we can discard unused feature maps, thus reducing the state space's dims to 64 * 64 * 5 (64 is the screen's resolution, and there are five feature maps involved). We use the default settings of making a decision every 8 frames, and an episode lasts for 120 s. In an episode, when all the mineral shards have been collected, the units and the mineral shards will be reset randomly until reaching the time limit. For reward shaping, the agent gets +1 for every mineral shard collected, and −0.001 for every action to encourage faster collections. This setting is similar to that in Deepmind's work [8] (Fig. 1).

Fig. 1. A screenshot of Collect Mineral Shards (left) and the feature maps (right).

3.2 Asynchronous Advantage Actor-Critic

We are facing a reinforcement training task with a vast action space, so it is difficult to apply value-based DRL algorithms, in which one has to give an estimate of all the 4097 Q-values in one state while only a few Q-values are updated in one training step. So we decide to apply Deepmind's A3C [7] (Asynchronous Advantage Actor-Critic), an actor-critic method which avoid these problems. In A3C, the critic network estimates the V-value of the state while the actor gives out possibilities for taking each action. In the training process we used 64 asynchronous local threads to learn together, each of them sharing and updating parameters via a shared global net (Fig. 2).

For the network structure, we used convolutional neural networks similar to Deepmind's: two convolutional layers of 16, 32 filters of size 8 * 8 and 4 * 4 with strides 4 and 2, connected with two dense layers [8]. For spatial actions (e.g. choosing a point in the map), instead of outputting the possibility for choosing each point, we assume the points' coordinates to be normally distributed and output their means and variances. This setting introduces a nice prior, that is, destination points near the

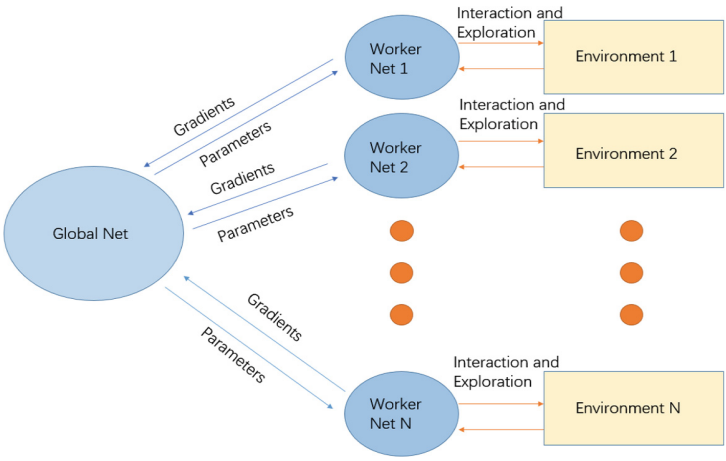

Fig. 2. Illustration of how A3C works. Notice that all worker nets work asynchronously, which avoids the relevance of experience data.

optimal point should also be good, and that should contribute to a more stable training process.

As for hyper parameter settings, all network weights are initialized with a variance scaling initializer with factor 0.5, and all the worker nets share two common Adam optimizers with the same learning rate (one for the critic net, and one for the actor net). The parameters are updated every 40 game steps.

3.3 Introducing Online Expert Supervision: A3COE

For this DRL task, we can easily hard-code an agent which performs nicely: always moving the units to the nearest mineral shard. This strategy may be sub-optimal under some circumstances, but it is still a strategy nice enough to learn from.

With this hard-coded agent, we can introduce expert supervision while training. For each training step, apart from feeding states, actions and rewards to the network, we also feed expert's action to the network, and to utilize this expert demonstration, we change the loss of the actor network (6) to:

$$\text{loss}'_{\text{a}} = -(V_{actual} - V_{estimate}) * \log(p(a|s)) + \alpha * \log(p(a_E|s)) + \beta * E_a \tag{7}$$

where a_E represents the expert's actions, and α is a positive hyper parameter to control the degree of supervision. It is worth noticing that in this loss, β should be negative, for the network tends to enlarge variance to minimize the loss and an negative β encourages the network to converge. β was set positive in the original loss to encourage exploration, but we do not need to encourage it any more (for the supervision loss has already told us where to explore). We name this algorithm A3COE, which is short for Asynchronous Advantage Actor-Critic with Online Experts.

Instead of using the distance between the agent's actions and the expert's actions, we implement the log probability of the expert action as the form of the supervision loss for several reasons: it is a natural form of policy gradient [11]; it updates both the mean and variance; and it is easier to balance between the expert loss and the original policy gradient loss. This new loss combines reinforcement learning loss with supervision loss, and instructs the agent to explore certain actions, so this loss should be easier to converge, thus making the training process more stable.

3.4 Sparse Rewards and Curriculum Learning

One common problem in DRL is sparse rewards. In some states that are rarely reached, only a small number of actions will manage to get a good reward, so training will be slow in these states. One solution is to introduce curriculum learning, which manually designs a series of tasks in which difficult states appear more frequently, and trains agents on this series of tasks to improve performance. This training strategy is reasonable because it is similar to human's learning progress: to learn easy tasks first and then try harder ones.

In our experiment, we found that when there are few mineral shards left, the agent makes progress obviously slower, for these states (few mineral shards on the map) appear less frequently, and actions that acquire positive rewards are pretty rare under these states. So we applied curriculum learning to speed up training, designing maps of 2, 5, 10, 15 and 20 mineral shards in it, and set a score limit to decide which difficulty the agent should be trained with. Agents who get high scores will be trained with a more difficult map, while agents who fail to reach certain scores will be trained with easier maps. Detailed score settings are listed in the following chart (Table 1).

Table 1. Difficulties and score settings

Mineral shards on the map	Minimum scores required to enter next stage	Minimum scores required to stay in this stage
2	6	–
5	15	5
10	25	10
15	35	15
20	–	20

4 Results

4.1 A3C and A3COE

We tested the performance of A3C and A3COE in the Collect Mineral Shards task without curriculum learning. We set $\beta = -0.5$ and $\alpha = 5.0$ for A3COE and $\beta = 0.05$ for the original A3C. We tried three different learning rates and the results are in Fig. 3. The horizontal axis is the total episodes trained, and the vertical axis is the smoothed reward of each episode.

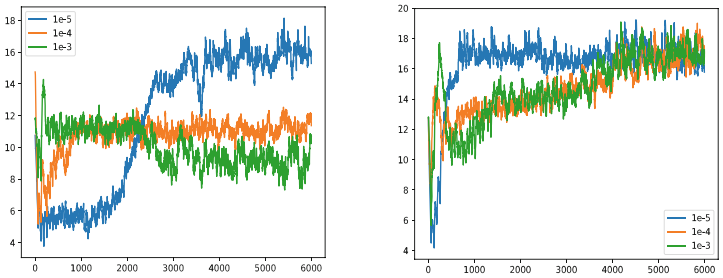

Fig. 3. Performances of the original A3C algorithm (left) and A3COE (right) with different learning rates.

The original A3C algorithm preformed just fine with a suitable small learning rate (1e−5), but with 1e−4 and 1e−3, the training process became unstable, and the agent was trapped in a local minimum, unable to perform better. In contrast, A3COE was more stable and performed nicely with all three learning rates. But the agent trained with the small learning rate still outperforms the other two in training speed.

Also, with a proper learning rate, A3COE learned very quickly, reaching the score of 18 at about 1,000 episodes, while the original A3C takes about 5,300 episodes. Supervision has speeded up the learning process as we expected.

4.2 Degree of Supervision

We also tested the influence of different supervision parameter α, also without curriculum learning, and the results are in Fig. 4. We can see that $\alpha = 1.0$ acquired the greatest learning speed, while the agent trained with $\alpha = 5.0$ performed slightly better in the end. The $\alpha = 0.1$ agent performed the worst, even unable to outperform the original A3C agent, and the $\alpha = 5.0$ agent was slower in the beginning because of overfitting. However, all of the A3COE agents outperformed the original one in training speed. So an appropriate parameter α will help A3COE to perform better.

4.3 Curriculum Learning

In the experiments showed below, agents learned significantly slower when they got a score of about 17, mainly because of sparse rewards. To boost training, we introduced curriculum learning and tested its effect.

We focused on the agent's ability to get a score more than 20 in a single episode, which means that the agent must overcome the states with sparse rewards and manage to collect all the mineral shards on the map, and that is exactly what we want curriculum learning to do. In the following statements, if an agent gets more than 20 points in an episode, we consider this performance to be 'nice', for it overcomes the hard state of having few mineral shards on the map. We trained four agents each taking a different training method, and all the agents were trained for 3,000 episodes. Then we tested them for 2,000 episodes each, and counted the times they got a score over 20. The results are in Table 2.

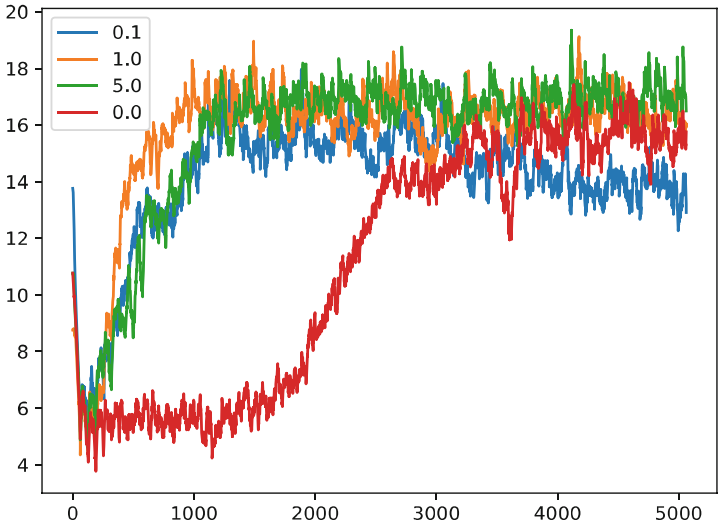

Fig. 4. Training agents with different supervision parameters.

Table 2. Performance of agents with different training methods

Agent training method	Times of getting over 20 points ('nice' performances) in 2,000 episodes
Original A3C without curriculum learning	34
Original A3C with curriculum learning	36
A3COE without curriculum learning	104
A3COE with curriculum learning	389

With supervision, curriculum learning indeed helped agents to overcome sparse reward dilemmas, enabling the agent to perform 'nicely' two times more frequently. However, the effect of curriculum learning in the original A3C case was rather weak, for the training without supervision was slow and the agent had not fully explored its policies.

4.4 Case Study

How does curriculum learning helps overcome sparse rewards? We inducted a case study to have a closer look. We tested the 'A3COE without curriculum learning' agent and the 'A3COE with curriculum learning' agent mentioned in Sect. 4.3 in two states: the first state is an 'easy' case with 20 mineral shards on the map and a large number of actions can get good rewards. And the other state is a 'harder' one with only two

mineral shards on it. We recorded position of mineral shards, position of the units the agents control, and the first action chosen by the agents and the expert, and the results are shown in Fig. 5.

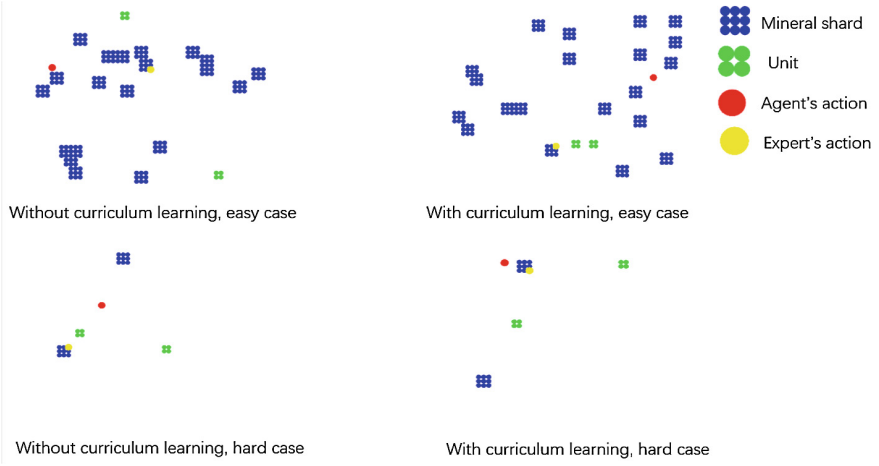

Fig. 5. The first action chosen by the agents in different states.

We can see that in the easy cases, it is acceptable to take actions different from the expert's for they will get good rewards as well, and so the two agents both performed nicely. However, in the hard sparse reward settings, few actions can get good rewards, so the agent without curriculum learning performed worse due to lack of exploration. In contrast, the agent with curriculum learning acted very similar to the expert because it had already been well-trained in sparse reward states.

5 Conclusions and Discussions

Human beings learn much faster with experts' instructions than exploring alone. Similarly, DRL algorithms that do not rely on expert data require a large amount of exploration, and supervision is a reasonable way to speed up training. In our experiment, supervised A3C outperforms the unsupervised algorithm, and is more stable as well.

In offline pre-training, the distributions of expert demonstrations and exploration experience may vary a lot, and that affects the effect of pre-training. However, online expert supervision is more flexible and reliable, and speeds up training much more directly. So in tasks which we can easily provide online expert policy, our method will train faster. Also, our method can deal with suboptimal expert demonstrations. After the agent performs nicely enough, we can train it with the original unsupervised method. As the suboptimal policy is more close to the optimal policy than random policies, this training process will be faster than completely unsupervised training. In that condition, our method serves as a kind of pre-training.

We observed a 'peak' in the training curve of supervised A3C algorithm with the learning rate 1e−4 and 1e−3 at about 400 episodes, but the score failed sharply after the peak before climbing up again slowly. Our explanation is that the actor network might be over-fitted at the beginning of training, and that causes the drop in the training scores that follow.

From the experiments, we can see the effect of curriculum learning and difficulty designing. However, there are still remaining questions. For example, how to design difficulties to maximize its effect? Up till now, there is no theory instructing on difficulty design, and we are applying it relying only on our observations and instincts. Also, is the 'difficulty' for us the same as the 'difficulty' for agents? In supervised learning, the difficulty may be the noise in data, but in reinforcement learning problems, there is still no clear definition for 'difficulty'.

This work was supported by the National Natural Science Foundation of China (61673234). This work was also supported by SaturnLab in FITC of Tsinghua University (Tsinghua iCenter).

References

1. Hessel, M., et al.: Rainbow: combining improvements in deep reinforcement learning. arXiv preprint arXiv:1710.02298 (2017)
2. Silver, D., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354 (2017)
3. Duan, Y., Chen, X., Houthooft, R., Schulman, J., Abbeel, P.: Benchmarking deep reinforcement learning for continuous control. In: International Conference on Machine Learning, pp. 1329–1338 (2016)
4. Hester, T., et al.: Deep Q-learning from Demonstrations. arXiv preprint arXiv:1704.03732 (2017)
5. Cruz, J., Gabriel, V., Du, Y., Taylor, M.E.: Pre-training neural networks with human demonstrations for deep reinforcement learning. arXiv preprint arXiv:1709.04083 (2017)
6. Zhang, X., Ma, H.: Pretraining deep actor-critic reinforcement learning algorithms with expert demonstrations. arXiv preprint arXiv:1801.10459 (2018)
7. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937 (2016)
8. Vinyals, O., et al.: StarCraft II: a new challenge for reinforcement learning. arXiv preprint arXiv:1708.04782 (2017)
9. Bengio, Y., Collobert, R., Weston, J.: Curriculum learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, vol. 60, no. 1, pp. 41–48 (2009)
10. Wu, Y., Tian, Y.: Training agent for first-person shooter game with actor-critic curriculum learning. In: 5th International Conference on Learning Representations (2016)
11. Sutton, R.S., Barto, A.G.: Reinforcement learning: an introduction. IEEE Trans. Neural Networks **9**(5), 1054 (1998)
12. Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D.: Deep reinforcement learning that matters. arXiv preprint arXiv:1709.06560 (2017)
13. Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning, vol. 135. MIT Press, Cambridge (1998)
14. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518** (7540), 529 (2015)