MLDRL: A multi-layer distributed reinforcement learning framework with multiple trainers

Jiarui Zhang¹, Yuan Li²* and Xuewen Wang¹*

¹School of Information Science and Technology, Northwest University, Xian 710127, China
²Academy of Military Science, Beijing, 100091, China
*Corresponding author’s e-mail:yuan.li@nudt.edu.cn, wangxuew@nwu.edu.cn

Abstract. Synchronous distributed sampling is widely used in various distributed reinforcement learning algorithms. However, it is difficult to scale to a large case due to significant synchronization and communication costs. In this paper, we propose a scalable layered distributed reinforcement learning framework by introducing multiple trainers. We use a parameter processor to coordinate parameters obtained from multiple trainers and introduce worker nodes to reduce the communication cost among the parameter processor and the trainers. We do experiments in a multi-agent environment, and the results show that the proposed framework significantly accelerates the training efficiency of the reinforcement learning algorithm.

1. Introduction

Reinforcement learning (RL) has gained widespread attention in recent years as it achieves great success in some challenging tasks such as Dota2 [1] and StarCraft II [2]. It works in a trial-and-error manner, which always needs a large number of interactions with the environment. To improve the training efficiency, different distributed methods have been studied for RL algorithms [3].

A widely used distributed method is the worker-trainer framework, which consisting of multiple workers with a centralized trainer. The worker is responsible for interacting with environments and then obtaining training samples. The trainer receives those samples and is responsible for updating the neural network [4]. However, most of the existing worker-trainer frameworks are only equipped with one centralized trainer [5], which greatly limits the speed of training and eventually slows down the data processing speed of the whole reinforcement learning system. Meanwhile, adding more trainers is non-trivial. This is because different learners have to coordinate their model updates to ensure the training process can be stable and convergent.

To address this challenge, we use multiple independent trainers and employ the parameter processor for coordinating distributed learning behaviors. In particular, each trainer is associated with several dedicated workers, and different trainers continuously synchronize their weights. With this multi-level distributed learning framework, the underlying reinforcement learning system can achieve high data throughput and learning efficiency.

For evaluation purposes, we use PPO, a popular policy-gradient reinforcement learning algorithm, as the baseline algorithm. We provide a novel parallel implementation of PPO using our multi-level distributed framework. We compare the performance of this novel implementation with distributed PPO (DPPO) [6], which is a popular distributed version. Experiments on a multi-agent combat task
verify that our framework can achieve a faster convergence rate when provided with the same amount of computation resources.

2. MLDRL: A distributed framework with multiple trainers

2.1. The structure of MLDRL

The distributed framework with multiple trainers is shown in Figure 1, which includes four layers. The bottom layer is composed of multiple workers, which is divided into different groups. Each group corresponds to a trainer in the third layer. We could limit the number of workers in each group such that samples obtained from those workers could be disposed in-time by the trainer. The third layer consists of many worker nodes. Each worker node plays the role of the relay station, which is used as a buffer between multiple trainers and the parameter processor. It stores parameters of neural networks obtained from a trainer. The parameter processor takes parameters from multiple worker nodes regularly and then merges them as new parameters for a neural network through certain computations. The new parameters are also stored in the worker node, which could be loaded by the trainer regularly. All work nodes are controlled by a common scheduler located at the service node. It assigns tasks to workers and monitors their progress. If a work program is added or deleted, it will reschedule unfinished tasks.

Figure 1. The structure of the distributed framework with multiple trainers

The main benefits of this distributed framework are summarized below.

(1) Each trainer is associated with a limited number of workers such that samples from those workers could be disposed in time.

(2) Worker node is introduced as a buffer of parameters to avoid the synchronization of parameter sharing among trainers.

(3) A parameter processor is introduced for parameter sharing among trainers such that multiple trainers could work cooperatively with high efficiency.
2.2. Design of communication mechanism
The worker node needs to communicate with the parameter processor and the trainer frequently to exchange weights of neural networks [7]. The efficiency of the communication becomes critical for the performance of the framework.

We adopt the RPC (Remote Procedure Call) protocol [8] for transmitting data among the worker node, the parameter processor, and the trainer. The RPC transmission method includes two modes: BIO and NIO [9]. BIO means a thread initiates an IO request. Regardless of whether the kernel is ready for IO operations, the thread blocks until the operation is completed and a thread is connected. NIO indicates that the thread initiates an IO request and returns immediately. After preparing for the IO operation, the kernel notifies the thread to do the IO operation by calling the registered callback function, and the thread starts blocking until the operation is completed.

In the considered framework, different worker nodes migrate parameters to the parameter processor simultaneously. In this case, both BIO and NIO will cause too much time overhead, which will slow down the efficiency. Considering that the time spent on the trainer's learning process is much bigger than that spent on the communication, we introduce a buffer layer called a parameter pool for each worker node. In terms of reducing IO blocking, the trainer will not directly transfer the data to the server node. Instead, they put the parameters into the buffer and continue to learn. At this time, each worker node will transfer data to the processor with NIO. The parameters provided by the parameter processor are put into the buffer. After the trainer is completed, the parameters are directly replaced from the buffer layer, thereby greatly saving the communication overhead blocked through the asynchronous method.

3. Experiments
In this section, we make experiments to show that the proposed distributed framework has better convergence speed than DPPO, a state-of-the-art distributed version of PPO. Then we also give some analysis on the impacts of some factors on DRLMT.

3.1. Environment
We conduct experiments on a multi-agent combat platform, Multi-agent Combat Arena (MaCA) [10]. It is an open-sourced platform that has attacked many RL researchers. There are two types of agents in the environment: detection units and attack units. The detection unit can simulate L and S-band radar for Omni-directional detection and support multi-frequency point switching. The attack unit has the ability of reconnaissance, detection, interference and attacking. In each scenario, the aircraft would attack the enemy unit whose position is detected by radar. If an enemy aircraft is killed, the player can get the score. The team with the highest score will win the game. We take a ten-to-ten combat scenario for our experiments.

3.2. Comparison results
In this section, we firstly show the benefits of MLDRL with PPO over DPPO. Then we make some ablation studies to analyze the factors related to the performance of the structure.

Figure 2(a) and Figure 2(b) show the cumulated average reward for MLDRL+PPO and DPPO, respectively, during the training process. Each point in the figure is the averaged reward over 10 rounds. The x-axis represents the training round. As we can see, MLDRL+PPO greatly accelerates the convergence speed compared to DPPO. MLDRL+PPO could get around 5000 rewards at 2000 rounds, while DPPO only gets around 3000 rewards. MLDRL is converged at around 7000 rounds, while DPPO is converged at about 8000 rounds. This comparison clearly shows that the proposed framework could accelerate the convergence speed of RL algorithms greatly.
Figure 2 The comparison of MLDRL+PPO with DPPO

Figure 3 The effects of the update frequency and the number of worker nodes

Next, we evaluate the effect of the update frequency of the parameter server and the number of working nodes on the performance of MLDRL+PPO.

Figure 3(a) shows the impact of different update frequencies 5, 10, 50 on the performance of the training result. The update frequency refers to the communication frequency between the parameter processor with the working node. As we can see, it has a slight impact on the performance of MLDRL. The higher the update frequency, the lower the performance. The reason is that if the update is too fast, it may generate more network communication and synchronization costs, which may offset the revenue generated by each update. However, due to the design of the efficient communication protocol, it does not lead to too much loss.

Figure 3(b) shows the impact of the number of worker nodes on the performance of the training result. This experiment aims to study the balance between synchronization overhead and efficiency brought by extra nodes, that is, how performance changes with the number of working nodes. We trained our agent with different numbers of worker nodes 4, 8, 16, respectively. As we can see, the curve for 8 workers shows the fastest converge speed, which is better than the curve for 4 and 16. This illustrates that it is no always good to add more worker nodes. There is a threshold for the number of worker nodes of the parameter server. This threshold should be set properly according to concrete scenarios.
4. Conclusions and Future Work
This paper proposes a multi-layer distributed reinforcement learning framework (MLDRL) for accelerating the convergence speed of the training process. This framework has many merits. Firstly, it supports more number of workers for sampling, as it introduces more number of trainers. Each trainer could handle a group of workers. Secondly, we introduce the parameter processor to handle the parameters obtained by different trainers. Since each trainer will deliver different parameters for a trained neural network. The parameter processor is used to coordinate all trainers. Thirdly, we introduce worker nodes to reduce the communication cost among trainers and the parameter processor. In this way, we do not have to synchronize trainers every time. The worker node plays the role of buffer to store parameters for all trainers. At last, we also make some improvements to the communication protocol between the parameter processor and the trainer, which reduce the communication cost further. As future work, it would be interesting to investigate the application of MLDRL on more environments and more algorithms.

Acknowledgments
This work is supported by National Natural Science Foundation of China with the funding number 61902425.

References
[1] Berner, C., Brockman, G., Chan, B., Cheung, V., & Zhang, S. (2019) Dota 2 with large scale deep reinforcement learning. CoRR, vol. abs/ 1912.06680
[2] Vinyals, O., Babuschkin, I., Czarnecki, W. M., etc. (2019) Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature, 575(7782), 350–354.
[3] Mousavi, S. S., Schukat, M., & Howley, E. (2018) Deep Reinforcement Learning: An Overview. Springer, Cham.
[4] Sutton, R. S., Mcallester, D., Singh, S., & Mansour, Y. (2000) Policy gradient methods for reinforcement learning with function approximation. In: Neural Information Processing Systems. Denver. p. 12.
[5] Sergey, I., Alexander D. (2019) Modern Deep Reinforcement Learning Algorithms. CoRR, vol. abs/1906.10025
[6] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017) Proximal policy optimization algorithms. CoRR, vol. abs/1707.06347.
[7] Li, M., Andersen, D. G., Smola, A., & Yu, K. (2014) Communication efficient distributed machine learning with the parameter server. In: Neural Information Processing Systems, Montreal. pp. 19-27.
[8] Bershad, B. N., Anderson, T. E., Lazowska, E. D., & Levy, H. M. (1990) Lightweight remote procedure call. ACM Transactions on Computer Systems., 37–55.
[9] Birrell, A. D., & Nelson, B. J.(1983) Implementing remote procedure calls. Acm Sigops Operating Systems Review, 17(5), 3.
[10] Multi-agent Combat Arena, (2020) https://github.com/SJTUwbl/MaCA