A Deep Q-Network Algorithm with Two-Level Neural Network in Real-Time Strategy Games

Anonymous Author(s)
Submission Id: 39*

ABSTRACT

We use a two-level deep neural network to improve the traditional Deep Q-Network algorithm used in real-time strategy games. The traditional DQN, which uses DNN instead of Q-tables, is an ideal solution for gaming, robot controlling, traffic controlling, etc. However, the state-action value function and the target network can only focus on each single units' behavior choices, ignore the overall situation, which is of much importance in a RTS game. To improve this, we use an additional three layers' DNN to handle it, give a global perspective to the value network. The whole project runs on MAgent.

CCS CONCEPTS

• **Computing methodologies → Q-learning**.

KEYWORDS

Multi-agent, Reinforcement Learning, Deep Q-Network, Real-Time Strategy Games

ACM Reference Format:
Anonymous Author(s). 2023. A Deep Q-Network Algorithm with Two-Level Neural Network in Real-Time Strategy Games. In *Proceedings of The Fifth International Conference on Distributed Artificial Intelligence (Conference DAI2023).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Real-time strategy games (RTS) have three main parts: micro-control, operation and reconnaissance, while micro is the most important part. The micro means in a local battle, each side of the player can control their units, attack or move, try to get local advantages over his or her competitors, which requires player have high technique and understanding of the game, for it needs cooperation of multi-units. By micro controlling, players can create man-up situation at some part of the battlefield and avoid fighting at other parts. Also, it can take advantages of each unit's special ability (like remote attack) to create counter-relation. Cooperation of high level needs grasp of whole battlefield's situation, while each agent can only get limited information.

In this paper, we develop a new two-level algorithm from the tradition DQN-algorithm[1] . Before battlefield information is sent into value network of DQN, a neural network will also process the data, add some upper instruction to each unit's operation valuation. We test its performance in three individual experiments which need the insight of different usages of all kinds of agents.

The whole project runs on MAgent[4], which is a platform can support many-agent reinforcement learning. This platform can support hundreds to millions of agents in one battlefield while can host up to million agents on a single GPU. Also, it provides configurations for users to design different kinds of agents and battlefields.

Conference DAI2023, Nov 30-Dec 3, 2023, Singapore, Singapore
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND AND MODELS

In a real-time strategy game, we are supposed to control each unit in a battle strategic to gain advantage. Since it's an incomplete information game, information each unit can get depends on its position in the battlefield. Thus, the controller need to make decision under information it can get, which cause a trade-of that whether to take a risk to make units disperse to get more information, to know more about battle situation. Since there exists need of cooperation, the ideal operation of each unit may not be the best choice of itself, but can help the whole army the best. So we need to improve the DQN algorithm on this problem, make it take global influence under consideration.

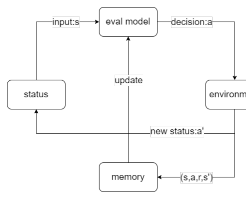

Figure 1: Structure of basic DQN

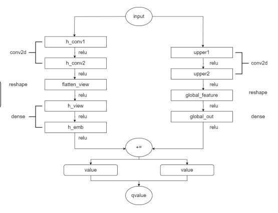

Figure 2: Network of advanced model

The basic model in the test use a standard deep Q-network algorithm, which contains optimization of target network[2], dueling DQN[3] and experience replay.

The advanced model has another neural network with two layers of convolution, a flatten layer and a dense layer. The upper network also take in the input, and the result of it will be added up to the result of basic model, to give a global influence. And after the flatten layer, we use $tf.reduce\_mean()$ to make influence on each unit is same.

The operation of each unit the model gives out will be sent to simulator. Then, the legitimacy of move instruction will be tested and the damage of every attack instruction will be calculated. After that, the simulator will flush the position and status of each unit and sent them back to observer, be the input of next flame.

3 EXPERIMENTS

The experiments were conducted on Ubuntu 22.04 with Intel(R)Core(TM) i5-13600K CPU @ 3.50GHz, 64GB 5200MHz RAM and an Nvidia GeForce RTX4080 GPU.

The environment of all experiments is python 3.9.16, tensorflow v1, cuda 11.2, cudnn 8.1, numpy 1.23.4, and the simulation platform is MAgent 1.0.

In each experiment, the size of battlefield is 125×125, There will be 2000 rounds of battle and training. A battle will be terminated rather when units of one player are all killed or reaches the maximum flames limit.

All training code and results can be seen in the github repo: https://anonymous.4open.science/r/TL-DQN-13F4/

There are three kinds of units in experiments. There configures are as follow:

Table 1: configures of units

name	small	marine	tank
width	1	1	2
length	1	1	2
hp	10	50	300
speed	2	2	1.5
view range	6	10	10
attack range	1.5	6	2
damage	2	6	30
step recover	0.1	0.1	0.1
step reward	-0.005	-0.005	-0.005
kill reward	5	10	50
dead penalty	-0.1	-0.1	-0.6
attack penalty	-0.1	-0.1	-0.1

3.1 Experiment 1

There are 625 "small" units at each side, and each time a "small" unit attacks another one, it can get 0.4 reward.

In the battle, we can see the cooperation of surrounding and anti-surrounding, and the side with advantage tries to hunt the rest units after the main confrontation.

3.2 Experiment 2

There are 8 "marine" at each side, and once a "marine" attack another one, it can get 0.4 reward.

Since units in experiment2 are all remote units, the battle decision becomes much more. During the training period, we can see strategies which have hugh difference.

For example, since view range of units are larger than attack range, and range of each side are equal, so when a unit observes an enemy, it tries to keep distance with it. Reasons for that are not hard to explain. As in one flame, one unit can rather attack or move, so when a "marine" tey to get close to an enemy try to attack, it will be attacked first. And in this "defind" strategy, keep close to you allies is also important. In this way, if a unit try to attack you, it would be also attacked by all your allies, creates a man-up advantage.

3.3 Experiment 3

In this experiment, the red side has 4 "marine"s and the blue side has 4 "tank"s. If "marine" attack "tank" successfully, the reward would be 0.4. And if "tank" attack "marine" successfully, the reward would be 2.4. Also, to make the battle more balance, we raised the hp and attack damage of "tank" and the moving speed of "marine",

Since battles in experiment3 are not equal, so the behavior of units change during the training progress.

After about 1000 rounds of training, the blue side learns to use its advantages in hp and attack damage. As "tanks" move slower that "marines", the blue controller tried to drove units of red to the corner, make red difficult to escape.

However, as the rounds of training accumulated, the red side began know to avoid being driven to corner and learned to diffuse and surround the units of blue player. Once a "tank" move to a "marine", the "marine" would move back and other "marines" nearby would move ahead, keeping a certain distance so that "marines" can attack "tanks" but "tank s" can't reach "marine". In human games, we call this technique "kiting".

3.4 replay

To better understanding the decision-making process during battle, you can go to "./battle replay" for replays of some battles in experiments.

4 CONCLUSION AND FUTURE WORK

In this paper, we developed a new two-level Deep Q-Network learning algorithm for Real-Time Strategy Games situation. In various battle situations, our algorithm shows high level of strategy and intelligence. And compared to origin algorithm, shows significantly stronger ability in two-team cooperation fight cases.

However, bounded by the platform, we can't design units with multi-attack, units can attack while moving or units with active abilities. In the future researching, we may update the simulation platform, give support to units above.

5 ACKNOWLEDGEMENTS

I would like to thank Weinan Zhang, Shanghai Jiao Tong University, my ML course teacher, who gave me much conduction and inspiration on this project, Han Zhang and Siyuan Feng, who helped me a lot in setting the environment.

I would also like to thank all contributors of project MAgent, which gave platform support to this project.

REFERENCES

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[3] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[4] Lianmin Zheng, Jiacheng Yang, Han Cai, Ming Zhou, Weinan Zhang, Jun Wang, and Yong Yu. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.