An investigation of StarCraft II Learning Environment

Xingming Qu Bowei Tian

Abstract—With the development of artificial intelligence, researchers are constantly proposing new methods for machine intelligence of making decisions. Real-time strategy games, which is the simulation of the real military combats, is getting more and more attention and becoming one of the most popular research areas in artificial intelligence. StarCraft II - a classic of RTS game which contains complex issues such as multi-agent collaboration, multi-task learning, and macro-strategy planning - is undoubtedly a perfect test environment for artificial intelligence community, especially in the reinforcement learning community. This paper investigates the StarCraft II Learning Environment SC2LE and improves an existing reinforcement learning methods for StarCraft II mini game by implement epsilon-greedy strategy. Experimental results show that our agent could achieve almost the same performance as the one we investigated but ues less training time.

Index Terms—Artificial Intelligence, Reinforcement Learning, Real-time strategy games, StarCraft II, PySC2

1 Introduction and Related Work

Artificial intelligence (AI) has made tremendous progress in the past decade. At the time when AI was born, games were always an excellent test platform for researching AI. After the classic Atari video games [1] and Go [2], real-time strategy (RTS) games like StarCraft II (SC2)[1], which has always been a classic of RTS game among gamers, becomes the next important area of artificial intelligence for researchers [3]. Containing complex issues such as multi-agent collaboration, multi-task learning, and macro-strategy planning[2][4], SC2 even becomes a major platform and tool for artificial intelligence algorithm research. Once some breakthroughs and progress are made in the game, it will have a great impact on both business and social development [3] in the future.

On August 10, 2017, DeepMind and Blizzard released the StarCraft II Artificial Intelligence Research Environment SC2LE with "PySC2"[4], which is DeepMind's python environment wrapper, bringing artificial intelligence research to a whole new stage. This release provides the industry with a mature and open AI decision system test platform. In the artificial intelligence community, especially in the reinforcement learning community, this environment is undoubtedly providing a lot of research opportunities.

Reinforcement learning (RL) [5] has a relatively short but rich history. It is an important branch as well as the hottest research area of machine learning today. The essence of reinforcement learning is to solve decision-making problems and get an optimal policy for a specific problem, so that the rewards obtained under this policy are the biggest. This makes RL an intelligent learning method. Next we will review recent related works of using RL to play StarCraft II.

Unlike the tradition build-in "Agent" in SC2, which uses either manually defined macro action and scripts or search and planning approaches [6], RL methods can achieve more sophisticated operation. DeepMind, as the pioneer, first designs several mini-games to investigate different elements of SC2 gameplay. They propose several agent architectures such as FullyConv Agent and FullyConv LSTM Agent [3], which could achieve comparable performance of a novice player. Pang *et al* use hierarchical architecture to automatically learn macro

1. https://starcraft2.com/
2. https://us.battle.net/forums/en/sc2/topic/20743555960
3. https://us.battle.net/forums/en/sc2/topic/628238127
4. https://github.com/deepmind/pysc2

actions from human reply data, and rely on RL to automatically learn base operation and battle scheduling [7]. Their approach achieves the state-of-the-art performance and beats the built-in AI. Peng *et al* proposes a bidirectionally-coordinated net (BiCNet) to implement agent communication, and a multi-agent actor-critic framework to implement learning [8]. The results demonstrate the good performance of the method that the agents could collaborate and master diverse combats.

As an investigation report of SC2LE, this paper focuses on relative "easy" tasks in SC2. SC2 mini game mentioned in [3] will be used. An existing implementation of [3] made by "XHUJOY"[5] will be used as the base model to modify. The rest of the paper is organized as follows. In section 2 we describe the project goal, as well as the task to do. Section 3 introduces the agent architecture and how we improve it by implementing epsilon-greedy strategy. Section 4 explains the experiment details and gives analysis of experimental results. In the end, we draw a conclusion of the paper and discuss some future work.

2 Project Goal

In this paper, we will first investigate the implementation from "XHUJOY" (since DeepMind did not open source their code). Three SC2 mini games will be used as examples. There are:

" 1. MoveToBeacon: The agent has a single marine that gets +1 each time it reaches a beacon. This map is a unit test with a trivial greedy strategy.

2. CollectMineralShards: The agent starts with two marines and must select and move them to pick up mineral shards spread around the map. The more efficiently it moves the units, the higher the score.

3. DefeatRoaches: The agent starts with 9 marines and must defeat 4 roaches. Every time it defeats all of the roaches it gets 5 more marines as reinforcements and 4 new roaches spawn. The reward is +10 per roach killed and -1 per marine killed. The more marines it an keep alive, the more roaches it can defeat." [3]

By observing the game replay and studying the code, we will try do modify it to achieve better performance.

5. https://github.com/XHUJOY/pysc2-agents

3 Method

3.1 Original method

DeepMind introduced three different reinforcement learning agents in their papers, namely Atari-net Agent, FullyConv Agent, and FullyConv LSTM Agent. Since these three agents have similar learning algorithms and parameters, only neural networks of different architectures are used to extract features. So in the following content, we will only focus on the FullyConv Agent.

In the original paper, the author used the Asynchronous Advantage Actor Critic (A3C) algorithm in [9], where the actor acts as a strategy function to generate the action to be performed, and critic is used as a value function to evaluate the quality of the strategy function. The author then use established architectures from [1], [9] and did some adaptions to fit the specifics of the SC2 environment, specifically, the action space. The author uses a a fully convolutional network, which has no stride and uses padding at every layer. Therefore, the resolution of the spatial information in the input will be preserved. Figure 1 shows the the proposed FullyConv architectures, which uses a sequence of resolution-preserving convolutional layers to directly predict both spatial actions and non-spatial actions.

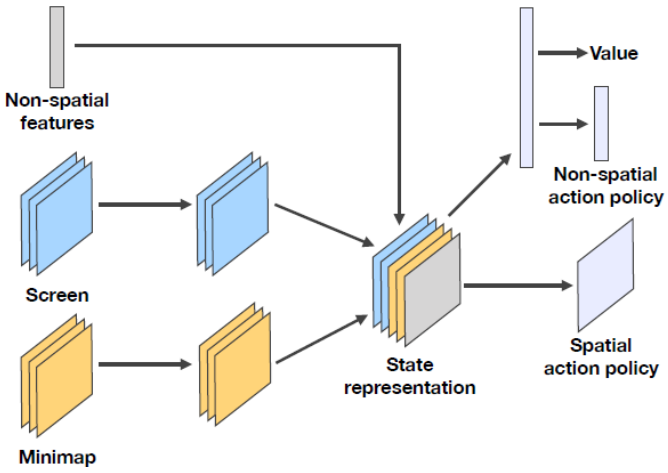
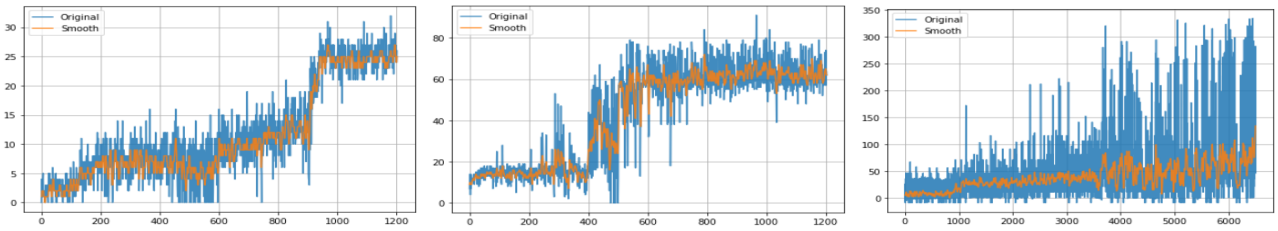


Figure 1: FullyConv architecture from [3]

3.2 Improved method

The implementation from "XHUJOY" can be considered as a simple implementation of [3]. When in-



(a) (b) (c)

Figure 2: (a), (b), (c) are results for MoveToBeacon, CollectMineralShards and DefeatRoaches respectively. Our agent converges at around 1k episodes in (a) and (b). We only run (c) for 6.5k episodes due to our limited computing power. Detailed analysis will be proved in section 5. Lines are smoothed for improved visibility.

vestigating the code, we notice that they set the initial "Epsilon schedule"(A list contains 2 elements. The first element is the probability of randomly selecting an action while the second element is the probability of randomly selecting a spatial target) to very small values. By observing the game replay, this makes the agent choose not to move most of the time. Therefore, the agent needs to play a huge amount of episodes to get one "valid random action" that could lead the agent to target. In addition, one action is so trivial for the agent to learn and update. That is the reason millions of episodes are need to train the agent.

Inspired by the epsilon-greedy strategy we learned in class, we will let our agent to select random action and spatial target most of the time in the early stages of training. This will speed up our exploration of the environment and accumulate more "valid random actions". Then as episode increase, we decay the epsilon until it reaches a pre-defined minimum value, which means we will no long act random and gradually let the agent to dominate and select the actions and spatial target (but still act random with a small probability sometimes). We follow the equation 1 to update epsilon.

$$\epsilon = Max(T_{min}, T_{init} - episodes/Decay\_factor) \quad (1)$$

Where T_{min}, T_{init} and $Decay\_factor$ are all hyper-parameters. The most significant hyper-parameter is $Decay\_factor$, whose ideal value will be select by our posteriori method. We will give detail explanation about parameters selection in section 4.

4 EXPERIMENT SETTINGS

4.1 Parameters selection

We provide some range of parameter choices. T_{init} is the probability of randomly selecting an action at beginning. To speed up our exploration of the environment, T_{init} should be high. Therefore, $[0.7 - 1]$ is an ideal range. T_{min} is the min probability of randomly selecting an action when we no long decay ϵ. We are assuming that the agent has already gathered enough "experience" to exploit when ϵ no long decay. Therefore, T_{min} should be low, which means $[0 - 0.05]$ is an ideal range.

$Decay\_factor$ could be the most "tricky" parameter to set. The most significant reason is that it is a task specific parameter, which means the best $Decay\_factor$ changes according to the environment. Failure to select the appropriate value will cause the experiment to fail. While training, if $Decay\_factor$ is set too small, the less episodes the agent will explore the environment, which means the agent may not obtain enough "valid random action" to learn. If $Decay\_factor$ is set too large, the more episodes the agent will explore the environment, which means it will take more time to let the agent to dominate and select action.

Here we use a posteriori method to set $Decay\_factor$. We first set $T_{init} = 0.9, T_{min} = 0.05, Decay\_factor = 1000$ and run 1k episodes as "warm up". After that we watch the game replay. If we found the agent is already able to select valid action (means the environment is relatively simple and the agent do not need to spend long time to explore), we could decrease $Decay\_factor$. If we

Table 1: Comparison between our method and other implementations

	MoveToBeacon		CollectMineralShards		DefeatRoaches	
	Average	Training Time (episodes)	Average	Training Time (episodes)	Average	Training Time (episodes)
Human	26	N/A	133	N/A	46	N/A
xhujoy	*25*	100K	*62*	100K	87	100K
pekaalto	*25*	1.8k	91	60K	*70-90*	200K
Deepmind	26	1M	103	400M	100	600M
Ours	*25*	**1.2K**	*63*	**1.5K**	*80*	**6.5K**

found the agent still performs meaningless action (means the environment is relatively complicated and the agent needs to spend long time to explore), we could increase $Decay\_factor$.

4.2 Training

The experiment is conducted on a computer with Intel i7-8750H CPU and NVIDIA GeForce RTX 2070 MAX-Q. The learning rate α is set to 0.0005 using a RMSPropOptimizer with $decay = 0.99$. Total agent steps of each episode is limited to 240. In order to speed up the learning process, parallel training is used. We only trained our agents on each mini-game for $[1.2k - 6.5k]$ episodes due to our limited computing power.

5 Results and Analysis

Training results are shown on Figure 2. We also include the game replay videos at Google drive.

To measure the performance of our improvement, we compare our result with FullyConv agent in [3], "XHUJOY" and "pekaalto"[6]'s implementation and human baseline. The results can be found in Table 1. The average we use in the table is the average score of the last 200 episodes. Next we will give analyses for each map after watching the game replay.

1. MoveToBeacon: The agent first perform random actions which sometimes could make marine get to the beacon. It is relatively easy because the map is not big and the probability of selecting random action and spatial target is high at the early stages of training. As the episodes increasing and ϵ decreasing, the agent begins to dominate and keep moving itself to the beacon with high confidence (no longer be interfered by randomly selecting action). This probably explain why Figure 2 (a) shows the scores increase a lot after 900 episodes.

2. CollectMineralShards: Similar to 1 but it is a little bit different, mineral shards spread around the map, which means performing random actions has large probability to make marine get to mineral shards. Therefore, the agent could learn a lot from "valid random action" and exploit it when ϵ decreases. This probably explain why Figure 2 (b) shows the agent only need 800 episodes to converge.

3. DefeatRoaches: Different from the previous two maps, DefeatRoaches takes more time to train the agent. Besides, the agent may converge to sub-optimal policy due to the randomness and the choice of parameters directly determines whether the experiment is successful or not. One interesting failure cases is: Marines always stay in the same position and not attack roach. One possible reason is that at the beginning, marines perform randoms action and did not even kill one single roach, which will always receive negative score. In order to maximize the reward, the agent chooes to do nothing to keep 0 score rather than get negative score.

6 Conclusion

This paper introduces and investigates the StarCraft II Artificial Intelligence Research Environment SC2LE. By analyzing the existing implementation, we successfully identify the imperfect place and improve it with epsilon-greedy strategy. We train our agents in mini-games and achieve comparable results but used less training time. Training agents requires high performance on the computers. Moreover, there is a large randomness in training, which means we will get different results even using the

6. https://github.com/pekaalto/sc2aibot

exactly same parameters. We will continue improving our method and try to train our agent in more types of maps with more complex scenarios in the future.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[3] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[4] K. Shao, Y. Zhu, and D. Zhao, "Starcraft micromanagement with reinforcement learning and curriculum transfer learning," *IEEE Transactions on Emerging Topics in Computational Intelligence*, no. 99, pp. 1–12, 2018.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[6] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 5, no. 4, pp. 293–311, 2013.

[7] Z.-J. Pang, R.-Z. Liu, Z.-Y. Meng, Y. Zhang, Y. Yu, and T. Lu, "On reinforcement learning for full-length game of starcraft," *arXiv preprint arXiv:1809.09095*, 2018.

[8] P. Peng, Y. Wen, Y. Yang, Q. Yuan, Z. Tang, H. Long, and J. Wang, "Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games," *arXiv preprint arXiv:1703.10069*, 2017.

[9] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.