Tackling Micromanagement tasks in Starcraft With Fully Convolutional Network and Deep Reinforcement Learning

Mingkun Yu, *University of Waterloo*

Abstract—The abstract goes here.

Index Terms—Artificial Intelligence, Reinforcement Learning.

1 Introduction

Artificial Intelligence on video game playing has been a rapidly growing active field of research during the past twenty years [1]. Comparing to traditional objects such as chess [2], Othello [3] and Go [4], video games often involve multiple agents, have a lot of hidden information, develops quickly and have gigantic states and branching factors. Therefore a good video game AI should be capable of dealing with high uncertainty, reacting fast and accurate, and be proficient at distributing the actions across agents. This posts a great challenge to the existing AI technique. To provide test fields for video game AI research, there have been several video games that are made open-source with AI-friendly API, including Atari 2600 Games [7], DOOM [5], StarCraft: Brood War [**?**] and so on. In this project, we will be looking at StarCraft: Brood War(SC:BW).

SC:BW is developed and published by Blizzard Entertainment™ in 1998. It is a real-time strategy game in which players gather resources, build armies and conquer enemies. Typically the game-play is divided into two parts, macro-management(macro), in which players plan when and where to build units, as well as how many of them should they build, and micro-management(micro), in which players manipulate their units to gain an upper hand in specific tasks, such as engagement.

Fig. 1: A screenshot of Starcraft: Brood War

Brood War API (BWAPI) [6] was made public 2010, and have been used as several competition platforms ever since. In these competitions, participants are asked to submit bots that play one-against-one match against bots from other participants following standard human tournament rule. Occasionally there are also show matches between bots and human professional players following the same rule. Currently, human pro players significantly outperform bots [8].

In this project we attempt to tackle micromanagement, in which player controls units to perform low-level tasks, such as moving to a designated location, attacking specific enemies, and so on. Previously the most popular technique is potential field [9], who originated from robotics. It works by constructing ”forces” to agents, who are vectors with directions and magnitude. These forces are functions of the type of object and distance, with parameters either hardcoded or tuned. The final actions of agents are decided by the direction and magnitude of the net forces. Being an elegant method that requires very low computational resources, potential field lacks expressive power. Most of the functions used in potential field agents are piecewise linear, quadratic or exponential functions with predefined number of pieces.

A natural idea to extend potential field into machine learning is to find an end-to-end differentiable representation of potential energy and force that has high expressive power. Intuitively, the process ”input applying circular values to output” is very similar to the effect of a transposed convolution function, sometimes called deconvolution function. Therefore, a multilayer deconvolutional network should be able to represent a potential field with higher expressive power. To reduce the amount of computation and to suit the mainstream deep learning architectures better, we also use regular convolution layers in our model, making it a fully convolutional network (FCN) [10].

All source codes of this project can be found at https://github.com/RikonYu/4PoolDeepLearning/.

2 Past work

The field deep reinforcement learning has gained a large amount of attention in the past few years. Traditional Q learning are extended with deep learning architectures

including Deep-Q-Network (DQN) [11] as well as its successors, Double-DQN [12], Dueling DQN [13], and others. Actor-critic methods are also paired with neural network and asynchronous learning frameworks, results in what is now called asynchronous advantageous actor-critic (A3C) [14], and later UNREAL [15]. For a detailed review of the application of reinforcement learning techniques on video games, see [21].

SC:BW didn't draw much attention from AI researchers until the publish of BWAPI [6], which enables playing the game with codes and algorithms. The earliest attempt to apply RL techniques on SC:BW can be traced to [17]. They used a multilayer perceptron with online Sarsa and neural-fitted Sarsa to learn a policy on specified micro tasks against the built-in AI. [18] used a very similar approach, also with MLP and Sarsa and Q learning but on different tasks.

Modern deep learning pipelines became available after the publish of massive data collection and tools such as TorchCraft [19] and StarData [20]. One work that utilizes these data is [22], who uses a DNN to learn and predict build order of a player, which is the most essential part of macro-management. Recent approaches that use DRL for micromanagement includes [23], who proposed a novel framework for specified micro-tasks. They used a model consists of several convolutional layers followed by fully connected layers. They chose the action space to be moving to 8 different directions plus attacking one of the possible enemies within reach. Having low dimension, this action space is not stable as the number of enemies within reach varies from time to time.

Recently, Google Deepmind published an API framework for Starcraft II(SC2) [**?**], with some of their approaches. Being the direct successor of SC: BW, SC2 is actually quite a different game in terms of game pace, mechanism and strategy. This causes lots of trouble for researchers who want to transfer their SC: BW code to SC2. Therefore SC: BW is still a popular task.

The potential field was originally used for robot navigation to provide a fast and reliable substitution of A*. It was later extended as a grid world navigation technique by replacing forces with energies. Each object on the field assigns positive or negative energy to grids around it according to specific energy functions, and the final action of the agent is pointed toward the grid with the highest energy. Energy functions are often piecewise linear functions constructed by hand, e.g. [9], or tuned with an evolutionary algorithm, e.g., [**?**]. Its behavior is smoother than knowledge-based systems and is stable in multi-agent environments while requiring a low amount of computational sources. Thus it is a popular method under the low computational resource setup of SC: BW bot tournaments.

A fully convolutional network is first used at object segmentation in computer vision [10]. It has the characteristic that it could assign a label(or value) to each input pixel, thus constructing an output with same width and height as input. Right now its usage is very limited, only being an object or instance segmentation technique in computer vision. In this project, we mainly utilize deconvolution layers.

3 Problem Description

SC: BW can be treated as a partially observable Markov-decision process(POMDP). The environment can be viewed as a set of isolated variables plus a 2D grid world, on which multiple units exist. A unit can either be under the player's control, or under the opponent's control, or neutral. On each frame, a unit can take the action of moving to a specific location, attacking another unit, or standing still. Thus under POMDP settings, we use a decentralized style of learning that sets the agent to be each unit that is under the player's control, hence having multiple agents in one environment. The observation of an agent is everything within a square window centering at it. In this project we set the size of the window to be 512 * 512. We refer this window as the size range of agents. All agents obtain their own observation of the environment and act independently without communicating with each other. They commit actions on a discrete time scale, namely one action each frame, and the action space is all valid actions for the agent at the current time frame.

In this project, we propose two tasks that are generally handled with the potential field in previous SC: BW full game bots, one with a static setup and one with a dynamic setup.

The first task, called Geyser Finding, is relatively simple: The environment consists of terrains, drones, a hatchery, mineral clusters and a vespene geyser. Agents are all the drones, and they are supposed to wrap around obstacles and goes to a static object, namely the vespene geyser, as close as possible. Here the observations for each agent are all the units within its sight range, and action space is the movement to all valid locations as well as standing still. Everything except drones cannot change its location, nor can they perform any action. Therefore the potential field it generates should remain static.

The second task, called Vulture Kiting, is dynamic: The environment contains a plain terrain with fixed boundary, one vulture and a dynamic number of enemy zerglings. New zerglings are spawned every few frames at a fixed location. The agent is set to be the vulture, who is supposed to navigate its way through zerglings, killing them while staying alive. The agent has an attribute called hitpoints. It starts at 16 and decreases by 1 every time it gets attacked by an enemy zergling. The task ends when hitpoints reach zero. Its action space is all the possible moves including moving to locations where no zerglings occupy and attacking zerglings. Below is an illustration of the layout of these two tasks:

(a) Geyser Finding

(b) Vulture Kiting

Fig. 2: Top(a): A screenshot of the first task. Bottom(b): A screenshot of the second task. All units are labeled with their name. Red boxes indicate the range in which the agent can observe. Purple-ish shaped ground in (a) is called creep, red blood in (b) is the corpse of zerging, neither of them has any actual impact on the gameplay of these tasks.

.

4 MODEL

4.1 Architecture

Different from the popular model architecture which consists of a few convolution layers followed by several fully-connected layers, we use a neural network only consists of convolutional layers, deconvlutional layers, pooling layers and upsampling layers, thus making it a fully convolutional network. Bringing convolutional layers can help to detect patterns in the environment, and deconvolutional layers would provide a deep representation of energies. Below is an illustration of the model.

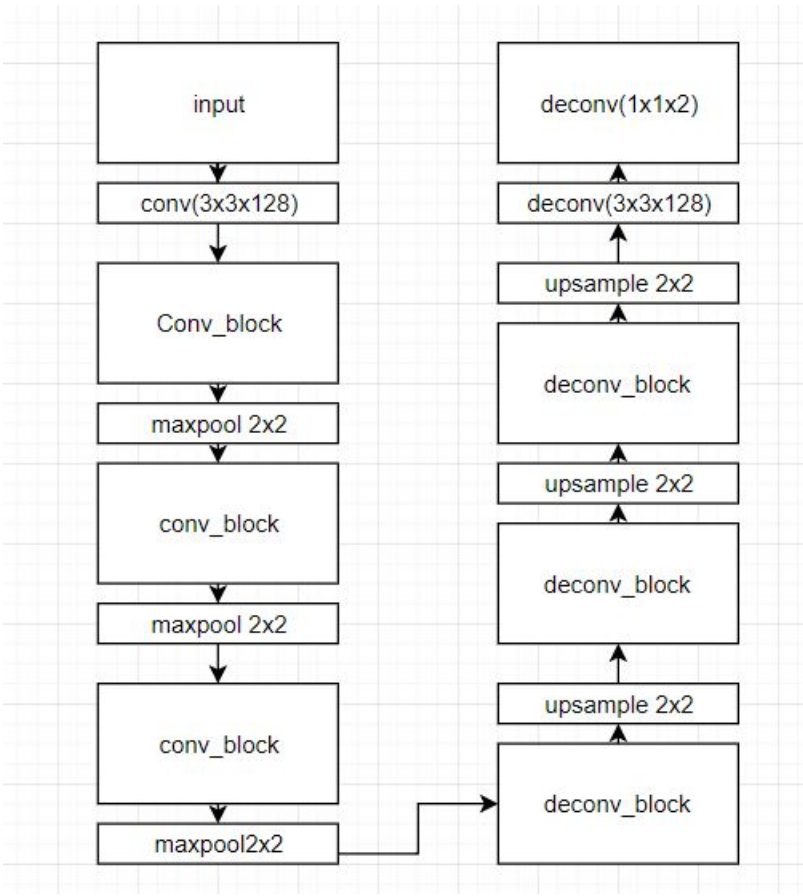
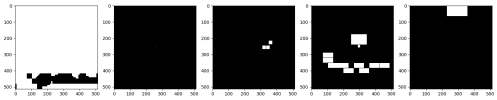


Fig. 3: Model architechure

4.2 Input

We pack up all information that our agents need into a 3D matrix of size 512 * 512 * 5. All coordinates mentioned below are coordinates relative to the agent ($(x, y) = (x_{absolute} - x_{agent} + 256, y_{absolute} - y_{agent} + 256)$).
In task one, channel 0 is for the visible terrain, where 1 at (x,y,0) indicates (x,y,0) is accessible(thus the agent can move to) for agent and 0 otherwise. Channel 1 is the location of the agent itself. Since all coordinates are transferred into relative coordinates centering at the agent, (256,256,1) is the only pixel in the second channel that is one. Although this seems unnecessary, In practice we find that keeping this pixel helps speed up convergence. Channel 2 is the location of all visible allies, (x,y,2)=1 if and only if there is an ally drone that covers (x,y). Channel 3 is the boundary of mineral clusters and the hatchery, they are not controllable, nor do they interact with agents, thus they can be treated as obstacles in this task. (x,y,3)=1 if and only if (x,y) locates inside one of these obstacles. Channel 4 is the location of vespene geyser. (x,y,4)=1 if and only if (x,y) locates within the geyser. Below is a frame of task 1 and how it is interpreted.

(a) A frame of task 1

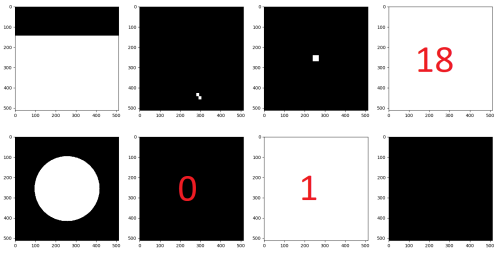


(b) Interpretation of the frame.

Fig. 4: A frame and its interpretation of task 1. Top(a): in-game screenshot, the agent here is the leftmost drone. Bottom(b): states that the agent receives. Left to right are the 5 channels we described above.

In task two, there are 8 channels. Channel 0 and 1 are the same as what they are in task 1. Channel 2 are bounds of all enemy zerglings. All pixels in channel 3 share the same values, which is the weapon cooldown of the agent. Channel 4 is the weapon range of the agent. (x,y,4)=1 if and only if (x,y) is within the maximum weapon range of the agent. Channel 5 is the weapon cooldown of enemy units. Channel 6 is the current hit points of the agent divided by its maximum hitpoints. Channel 7 is the center of all zerglings. Although this information can be inferred from channel 2, in practice we found adding this improves performance. Below is a frame of task 2 and how it is interpreted.

(a) A frame of task 2



(b) Interpretation of the frame.

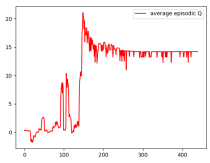
Fig. 5: A frame of the task Vulture Kiting and its interpretation. Top(a): the frame. Bottom(b): the 8 channels of the interpretation of this frame. Shared-value layers have yellow numbers written in them, who indicates the values of that layer.

4.3 Output

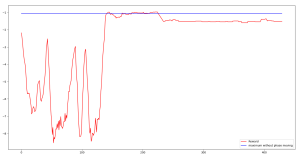
Output of the model is in the form of (x,y,z), where $0, y < 512,\ z \in 1, 2$.(x,y,0) should be interpreted as an order for the agent to move to the place (x,y). (x,y,1)should be interpreted as an order for the agent to attack the enemy centers at (x,y). In task 1 only moving is allowed, whereas in task 2 both moving and attacking are allowed.
Notice that there is a default AI in SC: BW. Units move to a location that is not adjcent to it based on a build-in pathfinding AI. When a unit is standing still, it will automatically attack the closest enemy within its seek range. To decrease the model complexity, we keep the pathfinding AI but we disables the automatic attack behavior by forbidding the agent to stand still to demonstrate the policy learned by RL.

5 Experiment

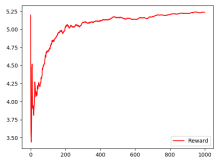
In this project, the algorithm we use is Deep Q Learning with target network and prioritized experience replay [16] with a buffer size of 100k. We use an exploration factor of 0.3, a discount factor of 0.95, batch size of 64(so the algorithm extracts 64 samples from the relay buffer to train every time), and use Adam optimizer on the neural net with a learning rate of 0.001. We replace the target network every 50 training epochs. We use a frameskip of 10 (which means that the agent makes new action once every 10 frames, and repeat the action for the other frames) for task one and a frameskip of 5 for task two. Task one has a maximum frame of 150, which means that the task ends when frame count reaches 150. There is no maximum frame for task two. We define a value function for task one: v_t=-euclideanDist((x_t, y_t), (x_{gas}, y_{gas}))/(3.72*10) where euclideanDist is the Euclidean distance between two points and 3.72 is the speed of a drone. That is, the value at time t is the distance between the position of agent at time t and the target object(vespene geyser). The reward for action a_t in task one is $v_t - v_{t-1}$. For the second task, the agent receives a reward of 1 when it kills an enemy, and a reward of -0.2 when it loses 1 hitpoint. Below are the training Q value and reward curve for these tasks. An illustration of the agent's behavior in task 2 can be found here https://github.com/RikonYu/4PoolDeepLearning/blob/master/results/task2.gif.



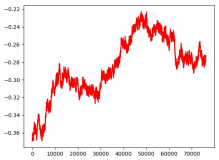
(a) Q value for task one



(b) Episodic reward for task one



(c) Episodic reward for task two



(d) Q value for task two

Fig. 6: Q values of each action and episodic rewards for each task. In task two we average the reward and Q value to smooth out the curve.

6 Discussion

We can see from the curve that our model indeed has the expressive power needed to present a good interpretation of the state, thus the agent can learn the correct behavior even with the most simple RL algorithm. In task one, our model seems to converge to a suboptimal policy, but because of the collision mechanism in CS:BW (units cannot stack on each other), any reward between -1.06 and -1.49 can be considered optimal (another agent might take the closest spot thus the agent recorded by the curve have to stay further). The second task is more complicated, thus more time is required for the agent to converge. We can see that the Q value is oscillating even if we take an average of 10k actions. In both tasks, the speed of training is lower than the speed of obtaining new data, resulting in an interesting phenomenon that it is very likely for PER to give data in the same order that we obtain them, which means that the replay buffer is actually nonexistent. This is caused by the fact that every new data has the highest priority(max priority of all previous data), and during each training procedure, we obtain data more than a sample batch. Because of this, we modify PER to give it highest priority plus the absolute value of its reward, and this seems to improve the performance.

Due to the massive action space (512*512 on average), exploration becomes a slow process, and thus decreasing the exploration factor is a bad idea. In fact, even in task 1 decreasing the exploration factor gradually will result in suboptimal policy. This also results in the huge spike of Q value at the beginning of the training phase. Treating this action space as continuous and apply the corresponding technique is worth trying. Restricted by the time factor, we couldn't implement models from other works to make a comparison, this should be covered in the future.

In the second task, once the gets close to its enemies which is an unavoidable scenario, any suboptimal move would lead to receiving negative reward. Unfortunately due to the huge action space, the probability of performing a suboptimal move at exploration stage is very high. Thus exploring during this "tightrope walking" phase almost always causes the agent to receive negative reward. This is another interesting phenomenon that worth tackling.

Despite the fact that task one involve multiple controllable agents, neither of these two tasks are a real multi-agent system. Drones do not collide with each other while they are moving to a resource field, although they do collide and split up after they reach it. Tasks that involve multi-agent cooperation, such as defusing mines with multiple dragoons, require specific RL algorithms. Those tasks also involve efficient multiagent exploration, which is an active area of research where performances are not as good as single agent RL. To solve those tasks, and eventually to master SC: BW, we need more complex model structure and RL algorithms.

References

[1] Buro, M. (2004, July). Call for AI research in RTS games. In Proceedings of the AAAI-04 Workshop on Challenges in Game AI (pp. 139-142). AAAI press.

[2] Campbell, M., Hoane Jr, A. J., Hsu, F. H. (2002). Deep blue. Artificial intelligence, 134(1-2), 57-83.

[3] Buro, M. (1997). The Othello match of the year: Takeshi Murakami vs. Logistello. ICGA Journal, 20(3), 189-193.
4

[4] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), 484-489.

[5] Kempka, M., Wydmuch, M., Runc, G., Toczek, J., Jakowski, W. (2016, September). Vizdoom: A doom-based ai research platform for visual reinforcement learning. In Computational Intelligence and Games (CIG), 2016 IEEE Conference on (pp. 1-8). IEEE.

[6] Heinermann, A. (2012). BWAPIAn API for interacting with StarCraft: BroodWar. available from (accessed 2014-02-03).

[7] Bellemare, M. G., Naddaf, Y., Veness, J., Bowling, M. (2013). The Arcade Learning Environment: An evaluation platform for general agents. J. Artif. Intell. Res.(JAIR), 47, 253-279.

[8] Kim, Y. (2017, November 01). Humans are still better than AI at StarCraft-for now. Retrieved April 15, 2018, from https://www.technologyreview.com/s/609242/humans-are-still-better-than-ai-at-starcraftfor-now/

[9] Hagelbck, J. (2012, September). Potential-field based navigation in starcraft. In Computational Intelligence and Games (CIG), 2012 IEEE Conference on (pp. 388-393). IEEE.

[10] Long, J., Shelhamer, E., Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3431-3440).

[11] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529.

[12] Van Hasselt, H., Guez, A., Silver, D. (2016, February). Deep Reinforcement Learning with Double Q-Learning. In AAAI (Vol. 2, p. 5).

[13] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581.

[14] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937).

[15] Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. arXiv preprint arXiv:1611.05397.

[16] Schaul, T., Quan, J., Antonoglou, I., Silver, D. (2015). Prioritized experience replay. arXiv preprint arXiv:1511.05952.

[17] Shantia, A., Begue, E., Wiering, M. (2011, July). Connectionist reinforcement learning for intelligent unit micro management in starcraft. In Neural Networks (IJCNN), The 2011 International Joint Conference on (pp. 1794-1801). IEEE.

[18] Wender, S., Watson, I. (2012, September). Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft: Broodwar. In Computational Intelligence and Games (CIG), 2012 IEEE Conference on (pp. 402-408). IEEE.

[19] Synnaeve, G., Nardelli, N., Auvolat, A., Chintala, S., Lacroix, T., Lin, Z., ... Usunier, N. (2016). Torchcraft: a library for machine learning research on real-time strategy games. arXiv preprint arXiv:1611.00625.

[20] Lin, Z., Gehring, J., Khalidov, V., Synnaeve, G. (2017). STARDATA: A starcraft AI research dataset. arXiv preprint arXiv:1708.02139.

[21] Justesen, N., Bontrager, P., Togelius, J., Risi, S. (2017). Deep learning for video game playing. arXiv preprint arXiv:1708.07902.

[22] Justesen, N., Risi, S. (2017). Learning macromanagement in starcraft from replays using deep learning. arXiv preprint arXiv:1707.03743.

[23] Hu, Y., Li, J., Li, X., Pan, G., Xu, M. (2018). Knowledge-Guided Agent-Tactic-Aware Learning for StarCraft Micromanagement. In IJCAI (pp. 1471-1477). SC2Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., ... Quan, J. (2017). Starcraft ii: A new challenge for reinforcement learning. arXiv preprint arXiv:1708.04782. MAPFHagelbck, J., Johansson, S. J. (2009, October). A Multi-Agent Potential Field-based Bot for a Full RTS Game Scenario. In AIIDE. overmindHuang, H. (2011). Skynet Meets the Swarm: How the Berkeley Over-Mind Won the 2010 StarCraft AI Competition. Ars Technica, 18.