2017 16th IEEE International Conference on Machine Learning and Applications

Learning to Coordinate with Deep Reinforcement Learning in Doubles Pong Game

Elhadji Amadou Oury Diallo, Ayumi Sugiyama, Toshiharu Sugawara
Department of Computer Science and Communications Engineering,
Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555, Japan
diallo.oury@fuji.waseda.jp • sugi.ayumi@ruri.waseda.jp • sugawara@waseda.jp

***Abstract*—This paper discusses the emergence of cooperative and coordinated behaviors between joint and concurrent learning agents using deep Q-learning. Multi-agent systems (MAS) arise in a variety of domains. The collective effort is one of the main building blocks of many fundamental systems that exist in the world, and thus, sequential decision making under uncertainty for collaborative work is one of the important and challenging issues for intelligent cooperative multiple agents. However, the decisions for cooperation are highly sophisticated and complicated because agents may have a certain shared goal or individual goals to achieve and their behavior is inevitably influenced by each other. Therefore, we attempt to explore whether agents using deep Q-networks (DQN) can learn cooperative behavior. We use doubles pong game as an example and we investigate how they learn to divide their works through iterated game executions. In our approach, agents jointly learn to divide their area of responsibility and each agent uses its own DQN to modify its behavior. We also investigate how learned behavior changes according to environmental characteristics including reward schemes and learning techniques. Our experiments indicate that effective cooperative behaviors with balanced division of work load emerge. These results help us to better understand how agents behave and interact with each other in complex environments and how they coherently choose their individual actions such that the resulting joint actions are optimal.**

I. Introduction

Many real-world systems are achieved by collective and cooperative effort, and thus, the need for collaboration between agents which are intelligent computational entities having specialized functionality becomes even more evident when looking at examples like traffic control [26], task allocation [24], ant colonies [23], or biological cells. Thus, multi-agent systems (MASs) arise in a variety of domains including robotics [16], distributed control [17], telecommunications [27], and economics [28]. The common pattern among all of the aforementioned examples is that the system consists of many agents that wish to reach a certain global or individual goal. These agents can often communicate with each other by various means, such as observing each other and sending messages. However, decision making in an intelligent MAS is a challenging issue since their appropriate behavior is inevitably influenced by the behaviors of others that often contain uncertainty. Furthermore, the goal can only be reached if the agents work together, the self-interested agents should be prevented from ruining the global task for the rest.

One question related to MASs is how much cooperation is required and can be achieved by the agents. On one end, we have independent learners trying to optimize their own behavior without any form of communication with the others agents, and they only use the feedback received from the environment. Whereas on the other end, we have joint learners where every agent reports every step they take to every other agent before proceeding to the next step.

Multi-agent learning is a key technique in distributed artificial intelligence, and thus, computer scientists have been working on extending reinforcement learning (RL) [10] to multi-agent systems to identify appropriate behavior, where Markov games have been recognized as the prevalent model of multi-agent reinforcement learning (MARL). However, many real-world domains have very large state spaces and complex dynamics, requiring agents to reason over extremely high dimensional observations, and thus, making optimal decisions in MAS intractable. Recently, the Google DeepMind research team presented a novel technique, so-called the deep-Q-network, which achieved breathtaking results by playing a set of Atari games, receiving only visual states [3] [5]. The same model architecture was used to learn different Atari games, and in some of them, the algorithm performed even better than a human player.

Multi-agent deep RL (MADRL) is the learning technique of multiple agents trying to maximize their expected total discounted reward, while coexisting within a Markov game environment, whose underlying transition and reward model usually are unknown or noisy. In MADRL, the optimal policy of an agent depends not only on the environment but on the policies of the other agents as well. Thus, cooperativeness in multi-agent learning is a problem of great interest within game theory and AI. Despite the success of deep RL, research has not sufficiently been done to extend these techniques in a multi-agent context except [30], [31], [29].

This paper is an attempt to explore how agents can dynamically learn cooperative and coordinated behavior using deep RL in a cooperative MAS. We also investigate how learned behavior changes according to the environmental factors including reward schemes and learning techniques. These results will help us better understand how agents behave and interact with each other in complex environments and how agents coherently choose their actions such that the resulting joint actions are optimal. Moreover, the findings of this research can be applied to solve a wide range domain specific problem with little effort.

0-7695-6321-X/17/31.00 ©2017 IEEE
DOI 10.1109/ICMLA.2017.0-184

II. Related Work

The research in the (multi-agent) RL field has been exponentially growing over the last decade. We focus only on the most related and recent work, specifically, on MADRL. It has been demonstrated that using neural networks can lead to good results [4] when playing complex games from raw images. However, the proof of the convergence does not hold anymore. It is known that using neural networks to represent the Q-values is unstable. Several ideas have been recently introduced to face with this issue. In DQN, experience replay [5] lets online RL agents remember and reuse experiences from the past. In general, experience replay can reduce the amount of experience required to learn, by replacing it with more computation and memory, which are often cheaper than the interactions between RL agents. Prioritized experience replay [6] investigates how prioritizing which transitions are replayed can make experience replay more efficient and effective than if all transitions are replayed uniformly.

Leibo et al. [7] analyze the dynamics of policies learned by multiple independent learning agents, each of which used its own DQN. They introduced the concept of sequential social dilemmas on fruit gathering and wolfpack hunting games. They show how behavior in conflict situations can emerge from competition over shared resources. He et al. [13] presented a neural-based model that jointly learns a policy and the behavior of its opponent and they indicated that opponent modeling is necessary for competitive multi-agent systems. Then, they proposed a method that automatically discovers different strategy patterns of the opponents without an extra supervision.

Independent DQN were used to investigate the cooperation and competition in a two players pong game [9]. They demonstrated that agents controlled by autonomous DQN are able to learn a two-player video game from raw sensory data. Their agents independently and simultaneously learn their own Q-function. The competitive agents learn to play and score efficiently while the collaborative agents find optimal strategies to keep the ball in the game as long as possible. However, work on applying deep RL in multi-agent contexts is limited to *independent*, *homogeneous* or *heterogeneous team learning* with the centralization of the learning algorithm. In this paper, we use *concurrent learning*, an alternative to team learning in cooperative multi-agent systems where both agents attempt to improve parts of the team.

III. Problem and Background

A. Problem

Pong game is a form of tennis where two paddles move to keep a ball in play. It has been studied in a variety of contexts as an interesting RL domain. In pong game, agents can easily last 10,000 time steps compared to 200-1000 in other domains such as gridworld, predator-prey game, and so on; its observations are also complex, containing the players' score and side walls. It has been shown that fully predicting the player's paddle requires knowledge of the last 18 actions [14]. Each agent corresponds to one of the paddles situated on the left and right side of the screen. There are three (3) actions that each agent can take: move up, move down, stay at the same place. The game ends when one team reach the maximum score of 21.

We developed a complex environment where we can easily add the number of players and modify the dynamics of the game to explore the agents' strategies (Fig. 1). In our doubles pong game, two agents form a team and try to win against a sophisticated hard-coded AI, by learning cooperative or coordinated behavior. The hard-coded AI can move faster than our agents but it may occasionally lose the ball. Our purpose is to investigate whether agents can learn to dynamically and appropriately divide their responsible area and stop the gap between them, as well as to avoid collision, by MADRL. The boundary between agents is shown by a dotted horizontal line in Fig. 1.

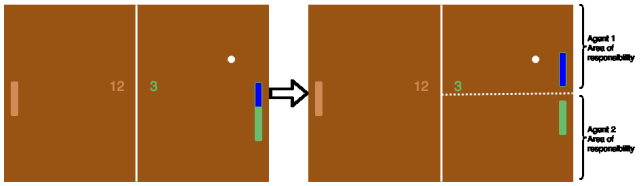


Fig. 1: Doubles pong game

B. Multiagent Reinforcement Learning

RL is to learn what to do next, and how to map situations to appropriate actions, so as to maximize a cumulative numerical reward signal through the continuous interaction between agents and the environment. The main goal of RL is to find the optimal action-selection policy. The generalization of the Markov decision process to multi-agent case is the Markov game.

Definition 1*: A Markov game is defined as* $\langle n, S, A, R, T \rangle$*, where* n *is a set of finite number of agents;* S *a finite state space;* $A = A_1 \times ... \times A_n$ *is a joint action space of agents (where* A_i *is the action space of agent* i*);* $R : S \times A \to \mathbb{R}$ *is the common expected reward function; and* $T : S \times A \times S \to [0, 1]$ *is a probabilistic transition function.*

The objective of the n agents is to find a deterministic joint policy or joint strategy $\pi = (\pi_1, ..., \pi_n)$, where $\pi : S \to A$ and $\pi_i = S \to A_i$, so as to maximize the expected sum of their discounted common reward. A non-trivial result, proven by [15] for zero sum games and by [2] for general sum games, is that there exist equilibria solutions for stochastic games just as they do for matrix game.

C. Deep Reinforcement Learning in a Single-Agent Case

In RL, agent i learns an optimal control policy π_i. At each step, i observes the current state s, chooses an action a using π, receives a reward r. Then, the environment transits to a new

state s'. The goal of the agent is to maximize the cumulative discounted future reward at time t_0 as:

$$R_t = \sum_{t=t_0}^{T} \gamma^{t-t_0} r_t, \tag{1}$$

where T is the terminal timestep and $\gamma \in [0, 1]$ is the discount factor that weights the importance of rewards. The action-value of a given policy π represents the utility of action a at state s, where the utility is defined as the expected and discounted future reward.

$$Q^{\pi}(s, a) = \mathbb{E}\left[R_t | s_t = s, a_t = a\right] \tag{2}$$

The optimal Q^* is defined as:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[R_t | s_t = s, a_t = a\right] \tag{3}$$

Q-learning [11] is an approach that iteratively estimate the Q-function using the Bellman equation

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

D. Deep Q Network in a Single-Agent Case

Originally, Q-learning uses a table containing the Q-values of state-action pairs where we assumed that a state is the screen image of a pong game. If we apply the same preprocessing as [4], which takes four last screen images, resizes them to 84×84 and converts to grayscale with 256 gray levels, we would have $256^{84\times84\times4} \approx 10^{67970}$ possible game states. In the problem we try to solve, the state also usually consists of several numbers (position, velocity, RGB values) and so our state space is almost infinite.

Because we cannot use any table to store such a huge numbers of values, we can directly extended it to deep reinforcement learning framework by using a neural network function approximator with the collection of weights θ. The weights θ can be trained by minimizing a sequence of loss functions $\mathcal{L}_t(\theta_t)$ that changes at each timestep t.

$$\mathcal{L}(\theta_t) = \mathbb{E}_{s,a,r,s'}\left[\left(y_t - Q(s, a; \theta_t)\right)^2\right], \tag{5}$$

where y_t is the target Q-value and defined as :

$$y_t = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s', a'; \theta_{t-1}) | s, a\right]. \tag{6}$$

The derivative of the loss function with respect to the weights is given by the following gradient:

$$\nabla_{\theta_t}\mathcal{L}(\theta_t) = \mathbb{E}_{s,a,r,s'}\left[\left(y_t - Q(s, a; \theta_{t-1})\right) \nabla_{\theta_t} Q(s, a; \theta_t)\right] \tag{7}$$

instead of using an accurate estimate of the above gradient, it is shown that using the following approximation stabilizes the weights θ_t in practice:

$$\nabla_{\theta_t}\mathcal{L}(\theta_t) = \left[\left(y_t - Q(s, a; \theta_{t-1})\right) \nabla_{\theta_t} Q(s, a; \theta_t)\right]. \tag{8}$$

In mini-batch training, we usually use the mean squared error (MSE) loss function that is defined as $MSE = \frac{1}{n}\sum_{t=1}^{n}(y_t - p_t)^2$, where y_t and p_t are the target and predicted Q-values. The error term or difference term $(y_t - p_t)$ can be huge for the sample that is not in arrangement with the current network prediction. This may cause very large changes to the network due to the *back-propagation* that impacts the weights during the backward process. To smoothen these changes, we could clip the derivative of MSE in the $[-1, 1]$ region and use the mean absolute error (MAE) outside. MAE is defined as: $MAE = \frac{1}{n}\sum_{t=1}^{n}|y_t - p_t|$, where y_t and p_t are the target and predicted Q-values in i-th sample.

In order to achieve such smoothness, we combine the benefits of both MSE and MAE. This is called Huber loss function [19], which can be defined as:

$$\mathcal{L}_{\delta}(y - p) = \begin{cases} \frac{1}{2}(y-p)^2 & \text{for } |y-p| < \delta \\ \delta|y-p| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \tag{9}$$

For all experiments below, we set $\delta = 1$. This treats small errors quadratically to gain high efficiency, while counting larges ones by their absolute error for robustness.

IV. Proposed Method

This section describes the approaches taken in this work. We extend the DQN algorithm to MAS and we empirically evaluate the effectiveness of applying deep reinforcement learning to concurrent MAS.

A. Deep Multi-Agent Concurrent Learning

In our study, agents should discover a coordinated strategy, which allows them to maximize their rewards. However, when multiple agents apply RL in a shared environment, the optimal policy of an agent depends not only on the environment but also on the policies of the other agents. In previous work [9], agents are not able to observe actions or rewards of other agents even though these actions interfere with each other (have a direct impact on their own rewards and environment). Our challenge is that each agent adapts its behavior in the context of another co-adapting agent without explicit control.

We proposed a framework (Fig. 2) in which two agents form a team to win against a hard-coded AI. We consider that agents are autonomous entities, that have *global goals* and *independent* decision making capabilities, but that also are influenced by each others decision. We assume that agents

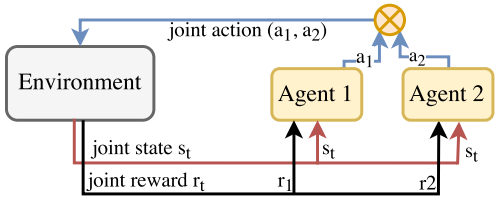


Fig. 2: Multi-agent concurrent DQN

do not have a predefined area of responsibility but they should learn to divide the area for an effective collaborative play. This is modeled as a concurrent learning, where agents attempt to improve the team score. Each agent has its own DQN to learn its behavior with the ε-greedy strategy to balance between

TABLE I: Rewarding scheme

Events	Agent 1 reward	Agent 2 reward
Left player loses	+1	+1
One agent loses	-1	-1
Collision	-1	-1

exploration and exploitation. It becomes hard to predict the teammate behavior since the two agents may not share the same reward structure.

Independent or joint learning agents can in principle lead to convergence problem since one agent's learning process makes the environment appear non-stationary to other agents. Agents modify their behaviors, which in turn can ruin other agents' learned behaviors [25] by making obsolete the assumptions on which they are based. Thus, they can smoothly forget everything they have learned.

B. Reward Structure

To achieve the desired behavior, we use reward shaping which is an attempt to mold the conduct of the learning agent by adding localized rewards that encourage the behavior consistent with some prior knowledge. It is known that with a good reward scheme, the learning process will easily converge in the single–agent case [18] .

Table I describes the reward used in our experiments. Every time the right players (our agents) wins, they receive a positive reward (+1). We punish both agents by giving them a negative reward (-1) when one of them loses the ball. They also receive a negative reward (-1) if they collide into each other. The minimum possible reward is -2 and this happens when both agents simultaneously lose the ball and collide. With this rewarding scheme, our agents jointly learn to cooperate with their teammate to win against the AI player and properly divide their area of responsibility.

C. Structure of Deep Q Networks

The DQN architecture used in our experiment is shown in Table II. We used convolutional layers to consider regions of an image and to maintain spatial relationships between the objects. The network takes screen pixels as input and estimates the actions. The images are rescaled to 84 $\times$ 84 pixels. The four last screens implicitly contain all of the relevant information about the game situation, with the speed and direction of the ball. By using a frame skip of 4, we fix the problem of the environment observability. The output of our network would be three (3) actions: up, down, and stay at same place.

D. Experience Replay

The learning process becomes smooth by storing an agent's experiences, and uniformly sampling batches of them to train the network. This helps the network to learn about immediate actions and from past experiences. We stored the experiences as a tuple of $\langle s, a, r, s' \rangle$. Instead of randomly sampling, prioritized experience replay [6] takes transitions that are not

TABLE II: DQN Architecture

Layer	Input	Filter size	Stride	Num. filters	Activation	Output
Conv1	84x84x4	8x8	4	32	ReLu	20x20x32
Conv2	20x20x32	4x4	2	64	ReLu	9x9x64
Conv3	9x9x64	3x3	1	64	ReLu	7x7x64
FC4	7x7x64			512	ReLu	512
FC5	512			3	Linear	3

in alignment with our network current estimate. Furthermore, we store the residual error between the actual Q-value and the target Q-value of a transition. This error ($e = |y_t - p_t|$) will be converted to priority of i-th experience z_i using this formula:

$$z_i = (e + \beta)^\alpha, \tag{10}$$

where $0 \leq \alpha \leq 1$ determines how much prioritization is used and β is a very small positive value. We then use the priority as a probability distribution for sampling an experience i [6], in which i is selected by the following probability:

$$Z(i) = \frac{z_i}{\sum_k z_k}, \tag{11}$$

where $\sum_k z_k$ is the total of all priorities and k is the size of the mini-batch. This is called the *proportional prioritization*. Each agent stores the most recent experiences into its own experience replay memory. This may not be optimal for a huge number of agents because the system would require a huge amount of memory and computation but it is efficient for the learning of cooperation with a few or several agents, like our cases.

V. Experimental Result and discussions

In this experiment, we pre-train the agents' networks to ensure a smooth convergence in the learning process by using *stochastic gradient descent*. We then initialize our network by feeding it the learned weights without using ε-greedy or *learning rate decay*; thus, the agents no longer need to explore their environment with this kind of initialization. We use the following parameters for all our experiments: learning rate $\alpha = 0.00025$, discount factor $\gamma = 0.99$, a screen size of 84×84 pixels and we concatenate the last four images to constitute one state. The loss function is optimized by using Rmsprop optimizer [20] with $momentum = 0.95$ and $\varepsilon = 0.01$. We update the target network every $10,000$ timesteps. To stabilize learning, we feed the network with small mini-batches of 32 samples. We terminate a game after $50,000$ timesteps (which is called an episode) even if none of the players did achieve a score of 21. The experimental results in the following sections describe the average (or mean values) of five experimental runs.

A. Learning Convergence

Fig. 3 plots the loss value during 11 million of timesteps and shows that there is no discrepancy between networks' predictions in agents 1 and 2. We can see that our agents learning process smoothly converged and they could learn the

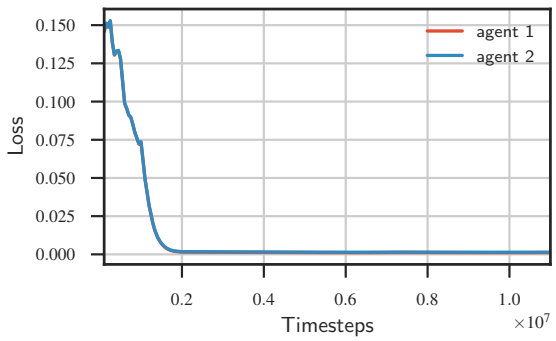


Fig. 3: Training Loss function

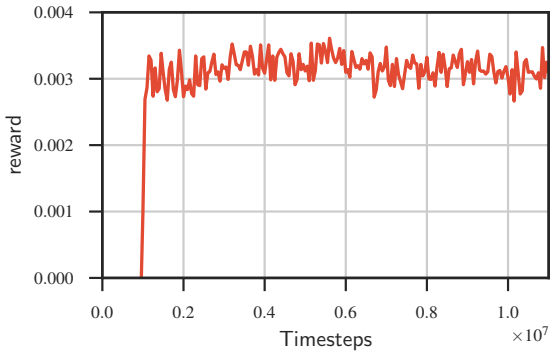


Fig. 5: Average reward per step

desired behavior in a very short time even in an apparent non-stationary environment. Note that the use of the Huber loss function was quite efficient in our experiments, but this is not always the case in some RL applications.

B. Q-values

The Q-values of agents 1 and 2 are plotted in Fig. 4. We see that the Q-values of agents 1 and 2 are almost identical and they have both learned acceptable policies that help them to win against the hard-coded AI. This can be explained by the fact that each agent is trying to adapt its behavior on the basis of the other agent.

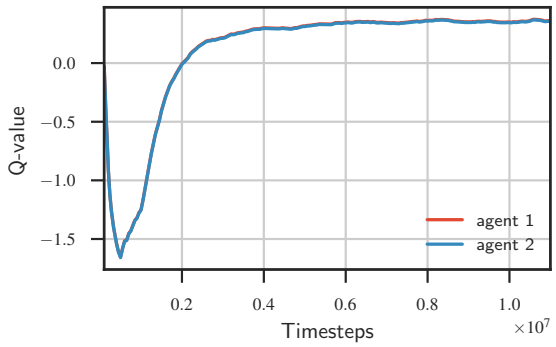


Fig. 4: Average Q-value per step

They first learned to avoid collision and only after that they started playing and winning the game. Their target networks predicted the right actions to take by following a near-optimal strategy. We could change this behavior by making them share the same experience replay memory and target network.

C. Reward

The reward is an important metric that shows the impact of each agent. We gave a great weight to future reward and decreased the importance of previous feedback because the discount factor γ was close to 1 ($\gamma = 0.99$). This helps us to overcome the credit assignment problem. Fig. 5 plots the average reward per step for both agents. It also indicates that our agents have good rewards per step. In this experiment, we gave the same reward to both agents whenever they win or lose. It might be desirable to assign credit in a different fashion. If one agent did the most of the job, it might be helpful to specially reward that agent for its actions, and punish the other for laziness.

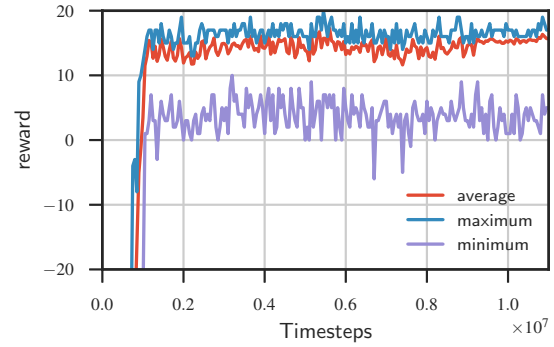


Fig. 6: Game reward per episode

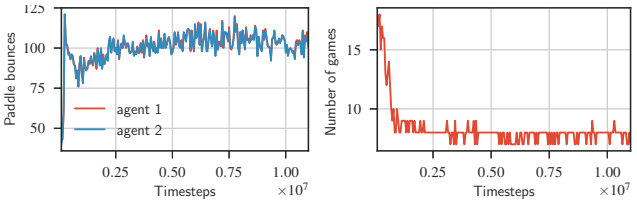
Fig. 6 represents the average, maximum, and minimum game rewards per episode. We did not show the very small rewards (≤ -21) for clarity. It also shows that our agents could easily achieve a reward of 15. We see that our agents could occasionally achieve the maximum possible reward of 20 and they generally avoid getting a negative reward after the learning process became stable.

D. Number of paddle bounces

Fig. 7a represents how many times an agent hits the ball per episode. In the beginning, agents that are randomly playing hit the ball only a few times, but after training for few millions of timesteps, both agents learn good policies which help them to keep the ball in play as long as possible. These values are correlated with the time a game lasts. We know more agents hit the ball, more the game lasts.

E. Number of games

We are interested to know how long one game can last. There are many ways we could do that. We decided to record the number of games every episode of 50,000 timesteps. Fig. 7b represents the number of games played by our agents. It also shows that our agents were playing so many games per



(a) Paddle bounces per episode (b) Games per episode

Fig. 7: Number of games and paddle bounces per episode

episode in the beginning. They often missed the ball during the first stage of training. After the learning process converges, they play around 7-8 games per episode. This means the game lasts for a relatively long time.

F. Remark

Our agents could approximately achieve the same game reward as in the case where the responsible area is given (the results are not shown). Note that if the reward for collision is less than -1 the agents will never learn to play the game, they would only avoid collision. If the collision reward is comprised in $(-1, 0]$ interval, the agents could achieve a reward of 20 per game but they collided a lot, and thus, the learning process became very long before they could adaptively and appropriately divide their responsible areas.

VI. Conclusion

We demonstrated that multi-agent concurrent DQN can smoothly converge even if the environment is non-stationary. To fully evaluate the effectiveness of deep reinforcement learning in multi-agent systems, we could extend double DQN [21], dueling network architectures for deep reinforcement learning [22], and different optimizers [12] and hyper-parameters to see which one is more suitable for multi-agent. We also show the details of how each agent chooses its strategies and adapts its behavior. We pointed out carefully and clearly some possible limitations of this method. One drawback of our method is both agents hesitate to take an action if the ball is on the frontier of their responsible areas. Obviously, this is not an optimal behavior. We will fix this strategy by making them communicate or negotiate in order to save time and energy for one of them.

In real-world applications, agents do not have the same resources and abilities and they do not have a full view of the environment. Our near future work is to improve our proposed method in a *partially observable Markov decision processes* where each agent has a limited view of the environment. We also plan to evaluate it in an environment where agents are heterogeneous in which they do not have the same speed, abilities, and size of paddles.

References

[1] P. Stone and M. Veloso, "Multiagent systems: a survey from machine learning perspective," Autonomous Robot, vol.8, no.3, pp.345–383, 2000

[2] J. Filar and K. Vrieze, "Competitive Markov decision processes," Springer-Verlag, New York, USA, 1996

[3] V. Mnih, et al., "Playing Atari with deep reinforcement learning," CoRR, abs/1312.5602, 2013

[4] D. Silver, et al., "Mastering the game of Go with deep neural networks and tree search," Nature 529, pp.484–489 , 2016

[5] V. Mnih, et al., "Human-level control through deep reinforcement learning," Nature 518, pp.529–533, 2015

[6] T. Schaul, J. Quan, I. Antonoglou and D. Silver, "Prioritized Experience Replay," CoRR, abs/1511.05952, 2015

[7] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel, "Multi-agent reinforcement learning in sequential social dilemmas," Proc. of the 16th Conf. on Autonomous Agents and MultiAgent Systems (AAMAS '17), pp.464–473, Brazil, 2017

[8] J. Perolat, F. Strub, B. Piot and O. Pietquin, "Learning Nash equilibrium for general sum Markov games from Batched Data," Proc. of the 20th AISTATS 2017, USA

[9] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, and R. Vicente, "Multiagent cooperation and competion with deep reinforcement learning," CoRR, abs/1511.08779, 2015

[10] R. S. Sutton and A. G. Barto, "Reinforcement learning: an introduction," MIT press, 1998

[11] C. J. C. H. Watkins, "Learning from delayed rewards," PhD thesis, University of Cambridge England, 1989

[12] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," CoRR, abs/1412.6880, 2014

[13] H. He, J. Boyd-Graber, K. Kwok and H. Daume III, "Opponent modeling in deep reinforcement learning," Proc. of the 33rd ICML, pp.1804–1813, 2016

[14] M. G. Bellmare, Y. Naddaf, J. Veness and M. Bowling, "The Arcade learning environment: an evaluation platform for general agents," Journal of Artificial Intelligence Research, 47, pp.253–279, 2013

[15] L. S. Shapley, "Stochastic Games", PNAS 1953 39 (10) 1095-1100, 1953

[16] G. Dudek, M. Jenkin, E. Milios and D. Wilkes, "A taxonomy for multi-agent robotics," Autonomous Robots, vo.3, no.4, pp.375–397, 1996

[17] J.C. Aguilar , M. Cerrada, G. Mousalli, F. Rivas and F. Hidrobo, "A multiagent model for intelligent distributed control systems," KES 2005. LNCS (LNAI), vol. 3681, pp. 191197. Springer, Heidelberg (2005)

[18] A. Y. Ng, "Shaping and policy search in reinforcement learning," PhD. Dissertation, University of California, Berkeley, 2003

[19] P. J. Huber, "Robust Statistics," Wiley, N. Y., 1981

[20] T. Tieleman and G. Hinton, Lecture 6.5, slide 29 - rmsprop, COURSERA: Neural Networks for Machine Learning

[21] H. V. Hasselt, A. Guez and D. Silver, "Deep reinforcement learning with double Q-learning," CoRR, abs/1509.06461, 2015

[22] Z. Wang, N. de Freitas and M. Lanctot, "Dueling network architectures for deep reinforcement learning," CoRR, abs/1511.06581, 2015

[23] C. W. Yeh and T. Sugawara, "Solving coalition structure generation problem with double-layered ant colony optimization," Proc. of the 5th IIAI Int. Cong. on Advanced Applied Informatics, pp. 65-70, Japan, 2016

[24] N. Iijima, M. Hayano, A. Sugiyama and T. Sugawara, "Analysis of task allocation based on social utility and incompatible individual preference," Proc. of the 21st Conf. on Technologies and Applications of Artificial Intelligence (TAAI 2016), pp. 24-31, Taiwan, 2016.

[25] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," In Proc. of the 15th national/10th Conf. on Artificial intelligence/Innovative applications of Artificial Intelligence (AAAI '98/IAAI '98), USA, 746-752, 1998

[26] K. Dresner and P. Stone, "Multiagent traffic management: an improved intersection control mechanism," In the 4th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 05), 471-477, 2005.

[27] A. L. Hayzelden and J. Bigham, "Heterogeneous multi-agent architecture for ATM virtual path network resource configuration," In Int. Workshop on Intelligent Agents for Telecommunication Applications (pp. 45-59), Springer Berlin Heidelberg, 1998

[28] T. Kaihara, "Multi-agent based supply chain modelling with dynamic environment," Int. J. of Production Economics, 85(2), 263-269, 2003.

[29] S. Omidshafiei, J. Pazis, C. Amato, J. P. How and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," CoRR, abs/1703.06182, 2017

[30] P. Peng, Q. Yuan, Y. Wen, Y. Yang, Z. Tang, H. Long and J. Wang, "Multiagent bidirectionally-coordinated nets for learning to Play StarCraft combat games," CoRR, abs/1703.10069, 2017

[31] J. N. Foerster, et al., "Stabilising experience replay for deep multi-agent reinforcement learning," CoRR, abs/1702.08887, 2017