JAWS 2017

Coordinated Behavior by Deep Reinforcement Learning in Doubles Pong Game

Elhadji Amadou Oury DIALLO[†a)], **Ayumi SUGIYAMA**[†b)], ***and*** **Toshiharu SUGAWARA**[†c)]

SUMMARY This paper discusses the emergence of cooperative and coordinated behaviors between independent agents using deep Q-learning. Multi-agent systems (MAS) arise in a variety of domains including robotics, distributed control, telecommunications, and economics. 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 learn independently, each agent uses its own DQN to modify its behavior. We also investigate how learned behavior changes according to functions of environmental factors including reward schemes and learning techniques. Our experiments indicate that they can emerge effective cooperative behaviors with a balanced division of work load. These results will help us better understand how agents behave and interact in complex environments and how they coherently choose their individual action such that the resulting joint action is optimal.
key words: *Artificial Intelligence, Cooperative Systems, Deep Reinforcement Learning, Multi-agent Systems*

1. Introduction

Nowadays, many real-world systems are achieved by collective and cooperative effort, and thus, the need for collaboration between agents 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 (MAS) 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 communicate with each other by various means, such as observing each other and sending messages. However, sequential decision making under uncertainty in an intelligent multi-agent system is a challenging issue since their appropriate behavior is inevitably influenced by the behavior of others. Furthermore, the goal can only be reached if the agents work together, or at least the self-interested agents can be prevented from ruining the global task for the rest.

One question related to multi-agent systems 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) to multi-agent systems to identify appropriate behavior, where Markov games have emerged 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, without any change, was used to learn different Atari games, and in some of them, the algorithm performed even better than a human player.

The multi-agent deep RL (MADRL) problem is the problem of multiple agents trying to maximize their expected discounted total 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 been done to extend these techniques in a multi-agent context except [29], [30], [31].

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 functions

[†]Department of Computer Science and Communications Engineering, Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555, Japan
a) E-mail: diallo.oury@fuji.waseda.jp
b) E-mail: sugi.ayumi@ruri.waseda.jp
c) E-mail: sugawara@waseda.jp

of environmental factors including reward schemes and learning techniques. These results will help us better understand how agents behave and interact in complex environments and how agents coherently choose their individual action such that the resulting joint action is optimal. Moreover, the findings of this research can be applied to solve a wide range domain specific problem with little effort.

2. 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 more memory, which is often cheaper than the interactions between RL agents in their environment. 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 find an optimal strategy to keep the ball in the game as long as possible. For a more general introduction of RL, see [10]. However, work on applying deep RL to multi-agent context is limited to *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.

3. Problem and Background

In this section, we introduce some background materials that will be used throughout this paper.

3.1 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 players' 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 the same place. The game ends when one team reach the maximum score of 21.

We develop 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.

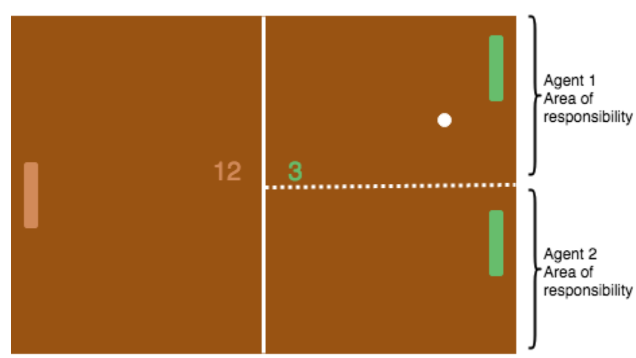

Fig. 1 Doubles Pong game
The agents explicitly know their area of responsibility

In this initial research, each agent has a given responsible area that limits their views and actions. We make sure that the agents never collide. This helps us to establish a baseline results for more dynamic environment. Our environment can be thought as an abstraction of a multi-robot patrolling [33] system in which each robot has its own camera to see the world. The agents or robots are going to use convolutional neural networks [32] to detect and recognize the scenes and new objects from the environment. The learned features will be used by the robots for the navigation through reinforcement learning techniques.

3.2 Multi-agent Reinforcement Learning

Reinforcement learning 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 cycle cycle of RL in a MAS is shown in Fig. 2. At each time step, the agents observe the current state from the environment, take a joint action among all available actions in that state and receive an immediate joint scalar reward and then the environment enters a new state. Agents observe the new state after taking an action. 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.

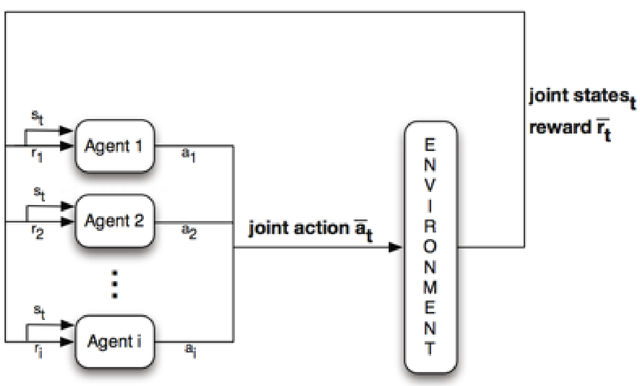

Fig. 2 Multi-agent reinforcement learning framework

***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 (joint strategy or strategy profile) $\pi = \pi_1, ..., \pi_n$ (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.

3.3 Deep Reinforcement Learning in a Single Agent

In RL, the agent learns an optimal control policy π. At each step, the agent i observes the current state s, chooses an action a using π_i, 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 as:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{1}$$

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

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

3.4 Deep Q Networks in a Single Agent

We apply deep reinforcement learning to the learning of cooperative behavior in a doubles pong game. 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], take four last screen images, resize them to 84 × 84 and convert to grayscale with 256 gray levels, we would have $256^{84 \times 84 \times 4} \approx 10^{67970}$ possible game states. In the problem we are trying 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 extend 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.

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

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:

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

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

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

In mini-batch training, mean squared error (MSE) loss function is usually used. It is defined as:

$$MSE = \frac{1}{n}\sum_{t=1}^{n}(y_t - p_t)^2 \tag{9}$$

where y_t and p_t are the target and predicted 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 way *back-propagation* impact the weights during the backward process. In order 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:

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|y_t - p_t| \tag{10}$$

where y_t and p_t are targets and predicted values in i-th sample.

In order to achieve such smoothness, we combine the benefits of both (9) and (10). This is called Huber loss function [19]. It can be defined as:

$$\mathcal{L}_\delta(y-p) = \begin{cases} \frac{1}{2}(y-p)^2 & \text{for } |y-p| < \delta \\ \delta|y-p| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \tag{11}$$

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.

3.5 Exploration and Exploitation

Exploration versus exploitation is one of the main dilemmas in RL. The optimal policy for an agent is to always select the action found to be most effective based on history (exploitation). On the other hand, to improve or learn a policy, the agent must explore new state it has never observed before (exploration). We use ε-greedy to balance between exploration and exploitation with $0 \leq \varepsilon \leq 1$. At every timestep, the agent acts randomly with probability ε and acts according to current policy with probability $1 - \varepsilon$. In practice, it is common to start with $\varepsilon = 1$ and to progressively decay ε.

4. 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.

4.1 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, agents are not able to observe actions or rewards of other agents even though these actions have a direct impact on their own rewards and their environment.

We proposed a framework (Fig. 3) in which two agents form a team in order to win against a hard-coded AI. We consider that agents are autonomous entities, that have individual goals and independent decision making capabilities, but that also are influenced by each others decision. We assume that each learning agent has the area

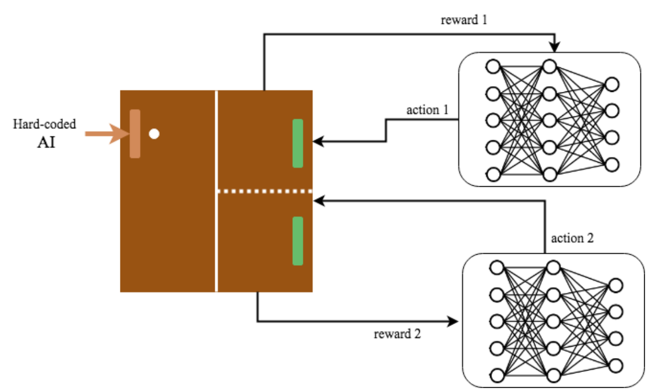

Fig. 3 Multi-agent concurrent DQN

of responsibility, a predefined region assigned to an agent. This is modeled as a concurrent learning, where our agents attempt to improve parts of the team score. Our challenge is that each agent adapts its behavior in the context of another co-adapting agent over which it has no control. Each agent has its own DQN to modify its behavior. It becomes hard to predict the teammate behavior since the two agents may not share the same reward structure.

Independent 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.

4.2 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 a 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 1 describes the reward used in our experiments. Every time the right players (our agents) wins, each of them receives a positive reward (+1). If agent 1 misses the ball in her area of responsibility, she gets a negative reward of -1. Agent 2 gets a negative reward of -1 if she misses the ball in her area of responsibility. An agent is not punished if her teammate misses the ball. With this rewarding scheme, our agents jointly learn to cooperate with their teammate to win against the AI player.

Table 1 Rewarding scheme

	Agent 1 reward	Agent 2 reward
Left player loses	+1	+1
Agent 1 loses	-1	0
Agent 2 loses	0	-1

4.3 Network structure

The DQN architecture used in our experiment is shown in Table 2. 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 84×84 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. While the idea is quite intuitive, in practice there are numerous challenges.

4.4 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 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:

$$z_i = (e + \beta)^{\alpha} \tag{12}$$

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:

$$Z(i) = \frac{z_i}{\sum_k z_k} \tag{13}$$

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.

5. 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.

5.1 Learning Convergence

Fig. 4 plots the loss value during 9 million of timesteps and shows that the discrepancy between the network's prediction and the actual Q-values exists and actually we can see a slightly better performance for one agent. We show that our

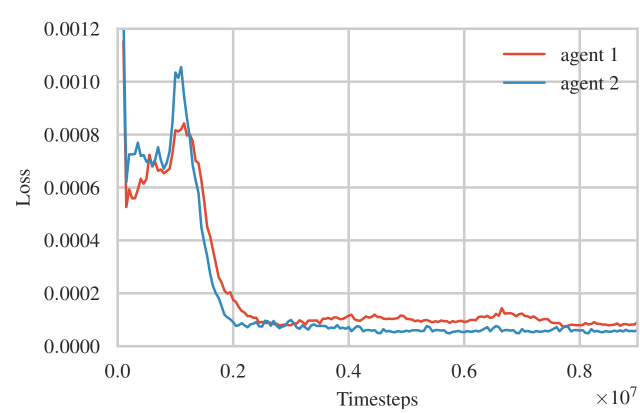

Fig. 4 Training Loss function
Performance of agents 1 and 2 until 9 millions of timesteps

agents' learning process smoothly converges. They could learn the desired behavior in a very short time. This is due to an apparent non-stationary environment and to the opponent strategy which often hits the ball towards the weaker agent whose learning process is slightly slow. We can note that the use of the Huber loss function was quite efficient in our

Table 2 DQN Architecture

Layer	Input	Filter size	Stride	Num of 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

experiments. However, this is not always the case in some RL applications.

5.2 Q-values

The Q-values of agents 1 and 2 are plotted in Fig. 5. We can see that the Q-values of agent 1 is slightly lower than the Q-values of agent 2, but we may note that they have both learned an acceptable policy that helps them to win against the hard-coded AI. This can be explained by the fact that the learning process of agent 2 is faster than agent 1 and agent 1 adapted to agent 2's behavior. Although, agent 1's network is predicting the appropriate Q-values. Agent 2 received less ball in her area, its target network more often predicted to stay at the same place, thus taking the best action in those situations by following a near-optimal policy. In the other end, agent 1 received many balls in her area, took many different actions and its learning process became noisy. We could change this behavior by making them share the same experience replay memory.

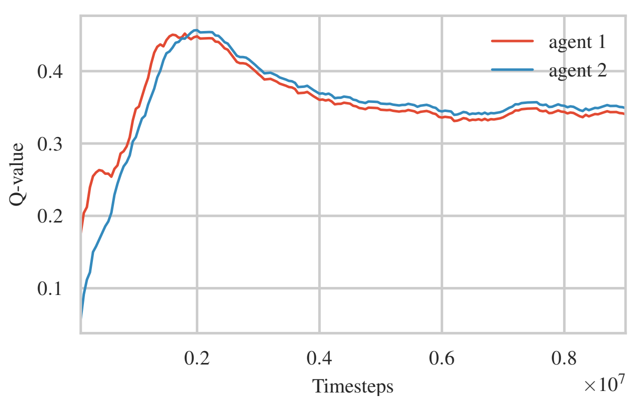

Fig. 5 Average Q-value per step

5.3 Reward

The reward is an important metric that shows the impact of each agent. Fig. 6 represents the average reward per step for both agents. In our experiments, the discount factor γ is close to 1 ($\gamma = 0.99$) that means we gave a great weight to future reward and decreased the importance of previous feedback. This helps us to overcome the credit assignment problem. We see that both agents already have a good reward per step. However, agent 1 is likely to continue missing the ball. In all experiments, we gave a positive reward (+1) to both agents whenever they win. It might be desirable to assign credit in a different fashion. If one agent did the most of the job, it might especially be helpful to reward that agent for its actions and punish the other agent for laziness.

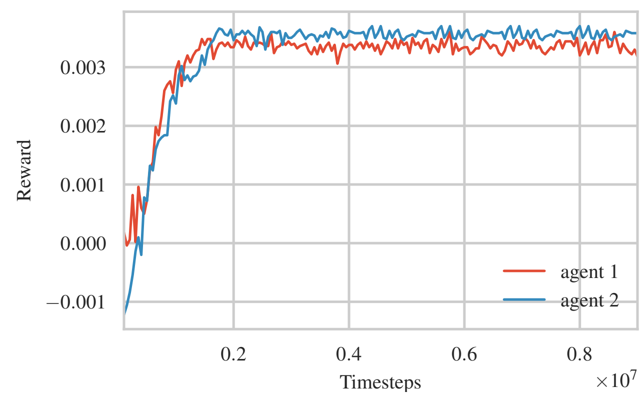

Fig. 6 Average reward per step

An episode is a sequence of 50,000 timesteps. The average reward per episode (Fig. 7) shows that the DQN agents are doing well against the hard-coded AI. They could easily achieve a reward of 15. A reward of 15 generally means our agents are taking the right decisions by following an optimal policy. A reward of 20 means the agents perfectly won the game without losing the ball at all. This phenomenon happens sometimes.

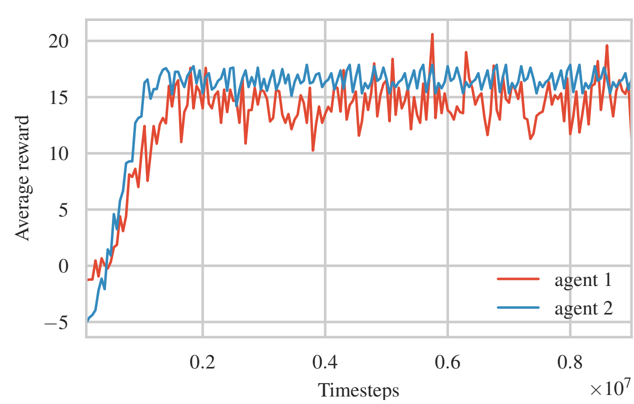

Fig. 7 Average game reward per episode

Fig. 8 plots the maximum reward per episode of each agent. It also shows that our agents could occasionally achieve the maximum possible reward of 20. This proves that the learning process converges. Fig. 9 represents

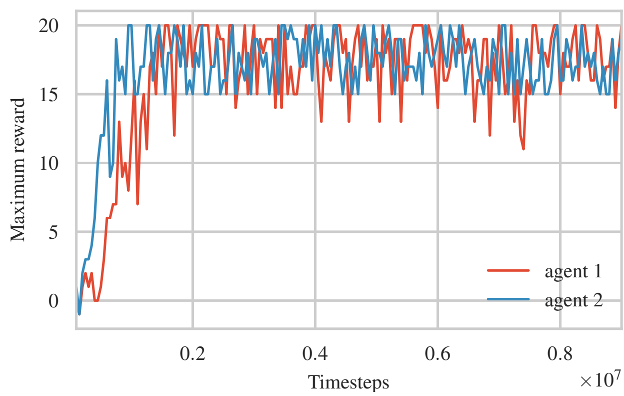

Fig. 8 Maximum game reward per episode

the minimum reward per episode of each agent. We see that after the learning process becomes stable, the agents completely avoid getting a negative reward.

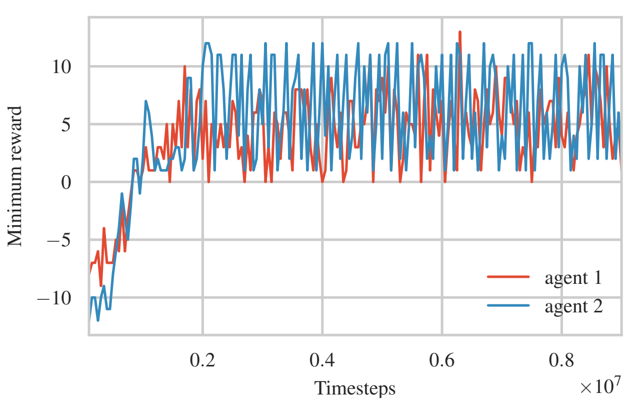

Fig. 9 Minimum game reward per episode

5.4 Number of paddle bounces

Fig. 10 represents how many times an agent hits the ball per episode. It also shows that randomly playing agents hit the ball few times. After training for few millions of timesteps, both agents had learned 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 our agents hit the ball, more the game lasts.

5.5 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. 11 represents the number of games played

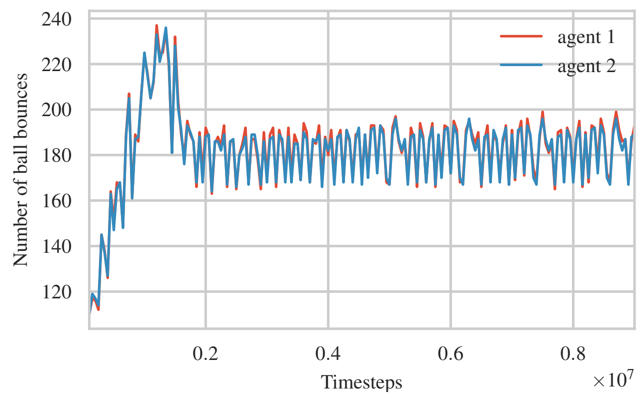

Fig. 10 Number of ball bounces per episode

by our agents during an episode. It also shows that our agents were playing so many games per episode in the beginning. They often missed the ball during the first stage of training. After the learning process converges, they play

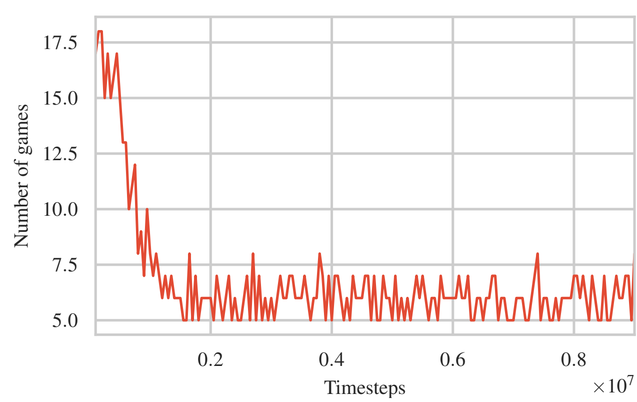

Fig. 11 Number of games per episode

around 6-7 games per episode. This means the game lasts for a relatively long time.

6. Conclusion

We demonstrated that independent DQN agents populating a multi-agent context can smoothly converge even if the environment is non-stationary. 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 need to take an action if the ball is at the frontier of their area of responsibility. This is not an optimal behavior. We could fix this strategy by making them communicate or negotiate in order to save time and energy for one of them.

To fully evaluate the effectiveness of deep reinforcement learning in multi-agent systems, we could extend double DQN [21], dueling network architectures [22], and different optimizers [12] to see which one is more suitable for an

MAS.

In real-world applications, agents do not know their area of responsibility, they should dynamically and appropriately adapt their area of responsibility on the basis of the other agent strategies. One possible extension is to let agents learn and divide their areas of responsibility and this is our future work. We also plan to extend this framework to a dynamically changing environment. The agents will jointly learn their area of responsibility by taking a joint action and receiving a joint reward from the environment.

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, USA, 2017.

[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, 2012.

[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.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, 1998.

[33] L. Iocchi, L. Marchetti, and D. Nardi, "Multi-robot patrolling with coordinated behaviours in realistic environments," In Int. Conf. on Intelligent Robots and Systems (IROS), pp. 2796-2801, 2011.