Learning Coordination in Adversarial Multi-Agent DQN with dec-POMDPs

Elhadji Amadou Oury Diallo and 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, sugawara@waseda.jp

Abstract

We examine whether a team of agents can learn geometric and strategic group formations by using deep reinforcement learning in adversarial multi-agent systems. This is a significant point underlying the control and coordination of multiple autonomous and intelligent agents. While there are many possible approaches to solve this problem, we are interested in fully end-to-end learning method where agents do not have any prior knowledge of the environment and its dynamics. We propose a scalable and distributed DQN framework to train adversarial multi-agent systems. We show that a large number of agents can learn to cooperatively move, attack and defend themselves in various geometric formations and battle tactics like encirclement, guerrilla warfare, frontal attack, flanking maneuver, and so on. We finally showed that by using only local views from the environment, agents create an emergent and collective flocking behaviors.

1 Introduction

Multi-agent systems (MAS) [26] are systems that contain many agents in the same environment. Although they can be used to collectively solve a problem that is too complex and costly to solve by a single agent [2, 3, 17]. We can often learn some interesting coordinated behaviors from nature and society such as flocking [19, 16, 5, 10] and group formation [4, 6, 7]. For instance, in an adversarial environment, agents learn how to minimize their encounters with opponents by strategically forming a group and by combining their knowledge of the environment to maximize the probability of avoiding opponents and obstacles.

In this paper, we address how deep reinforcement learning agents find strategic group formation by combining the local views of individual agents in an adversarial environment. When we apply reinforcement learning techniques in MAS, it is still unknown what will occur; for example, whether their learning converges and becomes stable and they can learn useful policies that facilitate achieving their goal. We also explore how agents adapt to the changes of opponent strategies.

2 Problem and proposed method

2.1 Adversarial MAS environment

A decentralized partially observable Markov decision process (Dec-POMDP) [1, 18] is defined as a tuple $\langle \mathcal{D}, \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \boldsymbol{\mathcal{O}}, \mathcal{O}, h, \mathcal{I} \rangle$. $\mathcal{D} = \{1, ..., n\}$ is a set of n agents. $\mathcal{S}$ is a finite set of states s in which the environment can be. $\boldsymbol{\mathcal{A}}$ is the finite set of joint actions of agents, $\boldsymbol{\mathcal{A}} = \mathcal{A}^1 \times ... \times \mathcal{A}^n$. $\mathcal{T}$ is a probabilistic transition function and $\mathcal{R}$ is the immediate reward function. $\boldsymbol{\mathcal{O}}$ is the finite set of joint observations with $\boldsymbol{\mathcal{O}} = \mathcal{O}^1 \times ... \times \mathcal{O}^n$. $\mathcal{O}$ is the observation probability function, h is the horizon of the problem, and $\mathcal{I}$ is the initial state distribution at time $t = 0$. When a joint action $\mathbf{a}_t = \langle a_t^1, \dots a_t^n \rangle$

32nd NeurIPS 2018. Workshop on Reinforcement Learning under Partial Observability, Montréal, Canada.

is executed, the environment transits from s_t to s_{t+1} with a probability $p(s_{t+1}|s_t, \mathbf{a}_t) \in \mathcal{T}$ and agent i receives a reward $r_t^i = \mathcal{R}(s_{t+1}|s_t^i, a_t^i)$. The goal of the agents is to find a deterministic joint policy $\pi = \langle \pi^1, \dots \pi^n \rangle$ so as to maximize the sum of their individual reward $r_t = \sum_{i=1}^{n} r_t^i$.

In this paper, we used a well-known adversarial multi-agent domain, the so-called *Battle game* [21], in which two teams of many agents are fighting against each other. Agents of the same team cooperate with their teammates to find some strategies in order to defeat the opponent team. The main goal is to kill as many opponents as possible by invading the neighbor territories and using some warfare strategies. Our environment operates on two teams of n agents and runs from some initial positioning of the agents inside a square in two-dimensional space for each team. An agent can take up to 7 actions: going up, going down, going left, going right, turning left, turning right and shooting. An agent is dead after being attacked three times.

2.2 Proposed learning framework

We proposed a distributed and parallel training framework (Fig. 1) for adversarial multi-agent systems with a large number of homogeneous agents. Each agent has its own local view o_t^i, its own action space a_t^i and its own reward r_t^i. At every time step t, an individual agent observes its local environment, takes an action, and receives a local scalar reward. There is no need for communication between agents of the same team.

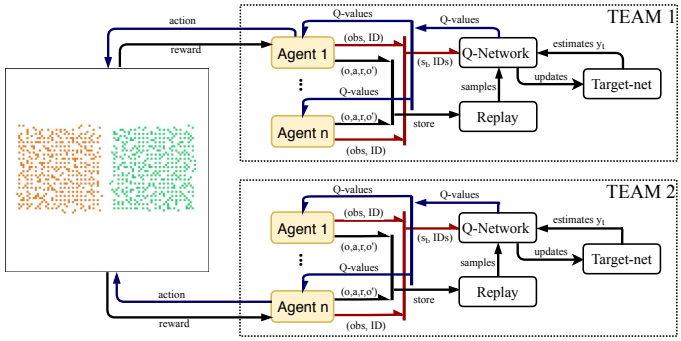

Figure 1: Scalable adversarial MAS architecture

The environment constitutes of two teams of n agents each ($n = 100, 200, 300, 400$) at the beginning of each episode. A single network computes the joint action and subsequently distributes it to the agents of the same team.

The neural net of each team receives a *tensorized* observation at times t and $t-1$, and the actions taken at time $t-1$. Each agent senses distances to neighboring agents within a radius k from its current position. The observation is of shape $(n, 2k+1, 2k+1)$ where n is the number of agents. Agents asynchronously infer their actions from the output of the network which contains the joint Q-values of every possible action for each team. Also, agents simultaneously take a joint action $\mathbf{a}_t$) before the network received its next observation $\mathbf{o}_{t+1}$.

3 Results

Both teams learn the same strategies, can kill approximately the same number of opponents, and rapidly co-adapt themselves based on the opponents' behaviors by finding some good defense strategies. Note that all networks have approximately the same values as the number of agents increases. The details are described in Appendix B. We can conclude that our proposed learning framework is scalable in non-stationary and adversarial learning environments with a large number of agents.

Table 1: Average reward and number of alive agents at the end of each episode during the test.

	n	**Random**		**DQN**		**DDQN**		**DuelDQN**		**DuelDDQN**	
		Team 1	Team 2	Team 1	Team 2	Team 1	Team 2	Team 1	Team 2	Team 1	Team 2
Reward	100	-4034 ±13	-4027 ±12	281 ±42	360 ±60	393 ±51	335 ±39	223 ±128	332 ±114	330 ±46	383 ±38
	200	-7990 ±24	-7987 ±21	649 ±145	506 ±121	716 ±85	658 ±92	498 ±99	607 ±102	604 ±142	729 ±94
	300	-11792 ±27	-11785 ±27	842 ±140	824 ±167	1010 ±88	1106 ±164	915 ±123	914 ±87	1102 ±91	1038 ±111
	400	-16274 ±39	-16267 ±33	1318 ±140	1412 ±175	1514 ±111	1643 ±138	1440 ±113	1450 ±141	1556 ±125	1708 ±110
No. alive agents	100	97 ±0	96 ±0	23 ±4	35 ±5	32 ±5	20 ±5	25 ±4	38 ±5	24 ±5	32 ±7
	200	189 ±0	189 ±0	58 ±11	37 ±8	57 ±12	40 ±9	48 ±6	69 ±9	39 ±11	52 ±10
	300	278 ±0	277 ±0	65 ±11	59 ±16	33 ±10	77 ±11	82 ±11	64 ±10	59 ±16	50 ±17
	400	381 ±1	380 ±1	54 ±15	93 ±17	47 ±13	76 ±18	75 ±14	88 ±12	39 ±13	75 ±18

APPENDIX A: METHODS

Reward scheme

Each agent receives a reward of -0.01 after each step. An agent gets $+5$ when it kills an opponent, and receives -5 when the agent is killed by an opponent. An agent receives a small negative reward of -0.01 whenever it attacks an empty position. This reduces the number of attempts an agent can attack empty positions, and thus making the training slightly faster. By reward shaping [15, 20], an agent receives $+1$ for attacking an opponent, and -1 if it is being attacked. The global reward of each team is the sum of all the rewards received by agents of the same team at any time step t, $r_t = \sum_{i=1}^{n} r_t^i$.

Exploration vs exploitation

Exploration vs. 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 new policy, the agent must explore new state it has never observed before (exploration) by taking non-optimal actions. We use ε-greedy [22] to balance between exploration and exploitation with $0 \leq \varepsilon(t) \leq 1$. At every timestep, the agent acts randomly with probability ε and acts according to current policy with probability $1 - \varepsilon(t)$. In practice, it is common to start with $\varepsilon = 1$ and to progressively decay ε as the training is going until we reach the desired ε. In general, we could use a fixed ε value. However, this is not always an optimal choice, because after many episodes of training, we would have a more accurate estimation of the policies and we could reduce the amount of exploration. The initial and final value of ε are, respectively, 1 and 0.01. In this case, our agents will be able to extensively explore during the first episodes and use the optimal policy without much exploration in the end of the training as agents get more knowledge about the environment.

DQN

In RL, agent i learns an optimal control policy π_i. At each step t, i observes the current state s ($s_t = \langle \mathbf{o_t}, \mathbf{o_{t-1}}, \mathbf{a_t} \rangle$), chooses an action a using π_i, receives a reward r. Then, the environment transits to a new state s_{t+1}. The goal of the agent is to maximize the cumulative discounted future reward at time t_0 as $R_t = \sum_{t=t_0}^{T} \gamma^{t-t_0} r_t$, where T is the terminal time step 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, $Q^\pi(s, a) = \mathbb{E}\left[R_t | s_t = s, \mathbf{a}_t = \mathbf{a}\right]$. The optimal Q^* is defined as $Q^*(s, \mathbf{a}) = \max_\pi Q^\pi(s, \mathbf{a})$.

Q-learning [25] is an approach that iteratively estimate the Q-function using the Bellman optimality $Q^*(s, \mathbf{a}) = \mathbb{E}\left[r + \gamma \max_{\mathbf{a}} Q^*(s_{t+1}, \mathbf{a}) | s, \mathbf{a}\right]$. We can use a parameterized value function $Q(s, \mathbf{a}; \boldsymbol{\theta}_t)$ when the task is complex. Then, we update the parameters by using the following formula: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(y_t^Q(s_t, \mathbf{a}_t; \boldsymbol{\theta}_t))\nabla_{\boldsymbol{\theta}_t} Q(s_t, \mathbf{a}_t; \boldsymbol{\theta}_t)$, where α is a scalar step size and y_t^Q is the target value function.

$$y_t^Q = R_{t+1} + \gamma \max_{\mathbf{a}} Q(s_{t+1}, \mathbf{a}; \boldsymbol{\theta}_t) \tag{1a}$$

$$= R_{t+1} + \gamma Q(s_{t+1}, \arg\max_{\mathbf{a}} Q(s_{t+1}, \mathbf{a}; \boldsymbol{\theta}_t); \boldsymbol{\theta}_t) \tag{1b}$$

DQN [13, 14] is an extension of Q-learning that uses a stack of convolutional neural network [8, 11] and fully connected layers to estimate $Q(s, \mathbf{a}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the weights of the network. We use a separate network to estimate the target Q-values, y_t^{DQN}. The target network has the same architecture as the DQN network but with frozen parameters $\boldsymbol{\theta}^{target}$ that are updated after every τ from the online network. This leads to more stable training because it keeps the target $\boldsymbol{\theta}^{target}$ fixed for a while. Now, we can can rewrite Eq. 1 as:

$$y_t^{DQN} = R_{t+1} + \gamma \max_{\mathbf{a}} Q(s_{t+1}, \mathbf{a}; \boldsymbol{\theta}_t) \tag{2a}$$

$$= R_{t+1} + \gamma Q(s_{t+1}, \arg\max_{\mathbf{a}} Q(s_{t+1}, \mathbf{a}; \boldsymbol{\theta}^{target}); \boldsymbol{\theta}^{target}) \tag{2b}$$

And we finally update the neural network weights by:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(y_t^{DQN} - Q(s_t, \mathbf{a}_t; \boldsymbol{\theta}_t))\nabla_{\boldsymbol{\theta}_t} Q(s_t, \mathbf{a}_t; \boldsymbol{\theta}_t) \tag{3}$$

Double DQN

The traditional Q-learning [25] algorithm overestimates the action-state values [9]. This is due to the use of the $\max$ function when updating the Q-values. One solution is Double DQN [23] that consists of using two independent value functions that are randomly updated, resulting in two sets of weights, $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$. Then we use one set of weights to determine the greedy policy and the other to determine its value. Given the fact that we already have two different value functions in DQN, y_t^{DDQN} becomes:

$$y_t^{DDQN} = R_{t+1} + \gamma Q(s_{t+1}, \arg\max_{\mathbf{a}} Q(s_{t+1}, \mathbf{a}; \boldsymbol{\theta}_t); \boldsymbol{\theta}^{target}) \tag{4}$$

We stored the experience $(s, \mathbf{a}, r, s_{t+1})$ in a replay buffer $\mathcal{D}$ and sampled mini-batches [12] from it to train the network by using the standard squared error loss. In the beginning, the buffer is filled with experiences played by random agents.

$$\mathcal{L}(\boldsymbol{\theta}_t) = \mathbb{E}_{s,\mathbf{a},r,s_{t+1}\sim\mathcal{D}}\left[\left(y_t^{DDQN} - Q(s, \mathbf{a}, \boldsymbol{\theta}_t)\right)^2\right] \tag{5}$$

We might note that Double DQN is shown to sometimes underestimate rather than overestimate the expected Q-values. However, the chances of both estimators underestimate or overestimate at same action is lesser.

Dueling Network Architectures

In many reinforcement learning problems, it is unnecessary to estimate the action value for each action as some actions have little to no value for a given state. Dueling networks [24] decompose the Q-Network into a value stream $V(s; \boldsymbol{\theta}, \beta)$ and an advantage stream $A(s, \mathbf{a}; \boldsymbol{\theta}, \alpha)$. The value stream expresses how good it is for an agent to be in any given state while the advantage stream compares how much better taking a certain action would be compared to the others. The relationship between value and advantage streams is expressed in the following formula:

$$A_\pi(s, \mathbf{a}) = Q_\pi(s, \mathbf{a}) - V_\pi(s). \tag{6}$$

This decomposition helps to generalize the learning for the state values. The output layer of this architecture is a combination of the value stream output and the advantage streams output. The aggregator is given by

$$Q(s, \mathbf{a}; \boldsymbol{\theta}, \zeta, \beta) = V(s; \boldsymbol{\theta}, \beta) + \left(A(s, \mathbf{a}; \boldsymbol{\theta}, \zeta) - \frac{1}{|A|}\sum_{\mathbf{a}} A(s, \mathbf{a}; \boldsymbol{\theta}, \zeta)\right), \tag{7}$$

where β is a parameter of V, ζ is a parameter of A, and $|A|$ represents the length of the advantage stream.

APPENDIX B: RESULTS

Experimental settings

We use the following parameters for all our experiments: learning rate $\alpha = 0.00025$, discount factor $\gamma = 0.99$. We update the target network every $10,000$ timesteps. To stabilize learning, we feed the network with medium size mini-batches of 250 samples. The DQN [23] network architectures are shown in Fig. 2. The experimental results in the following sections describe the average values of twelve (12) experimental runs with different random seeds.

Detailed experimental results

Fig. 3 represents the training performance of different networks with different number of agents per team with a narrow local field of view ($k = 3$). After more or less than $3,000$ training episodes, each network's training loss converges to a very low loss value. The environment is non-stationary because both teams are simultaneously learning. Hence, the best way to know if teams are learning strategic formations is probably to check if both networks converge to approximately the same values. In this case, the difference between the two errors is negligible. By using double DQN [23],

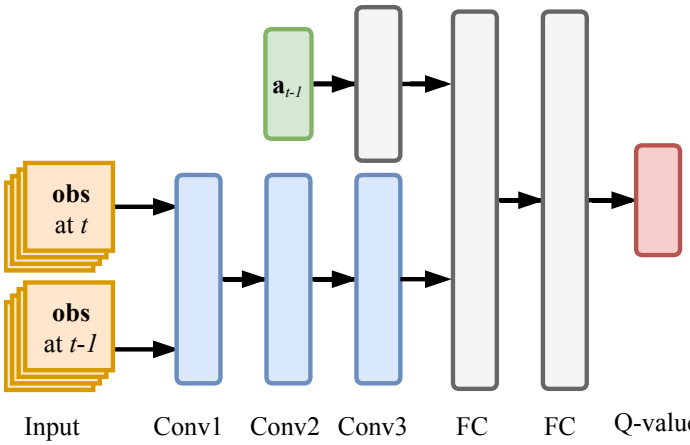

Figure 2: DQN architecture

we make sure that the networks do not overestimate the Q-values, and consequently, the action-state values will converge to the actual Q-values. The DQN networks have lower losses because they have less parameters to tune and the network structure is much simpler than the others. Dueling network architectures were useful in our environment due to the fact that the networks generate all possibles actions per team. It make sense to calculate to decompose the Q-values into a state value and an advantage stream because the action space is huge. Finally, we can see that dueling DDQN agents have the lowest discrepancy between the losses of the two teams.

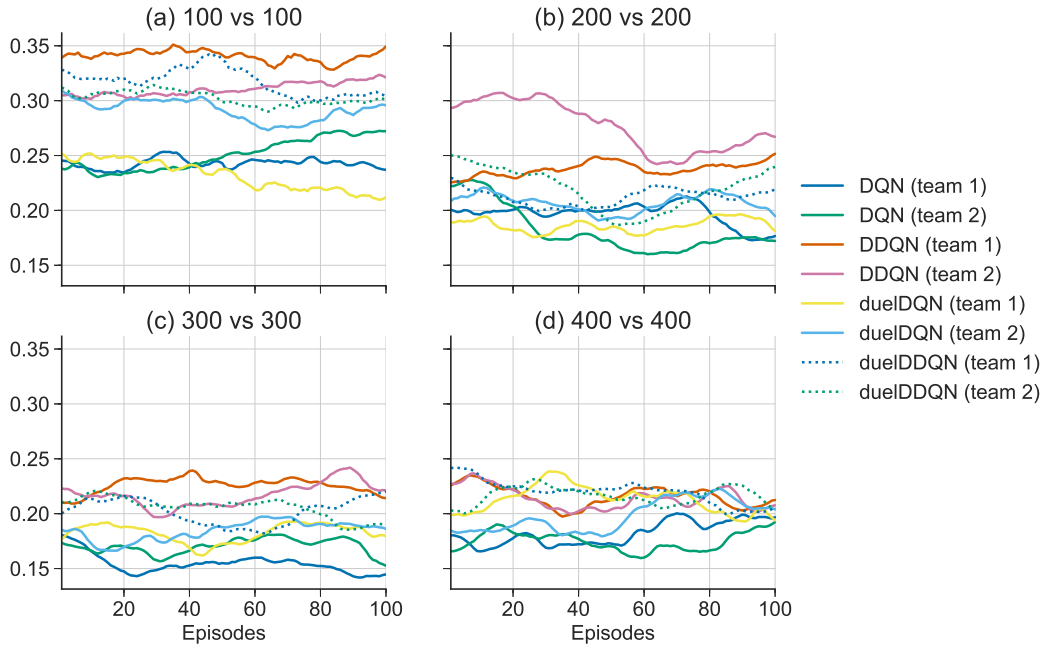

Figure 3: Loss curves during test phase. Average values of 12 runs with different random seeds.

Fig. 4 shows the average reward per episode for each team during test time. We see that both networks tend to have approximately the same reward. Our reward scheme could make a group of agents learn how to efficiently form various strategic group formations based on their opponents' strategies. The cumulative reward for random agents is always negative because agents are not learning any useful strategies.

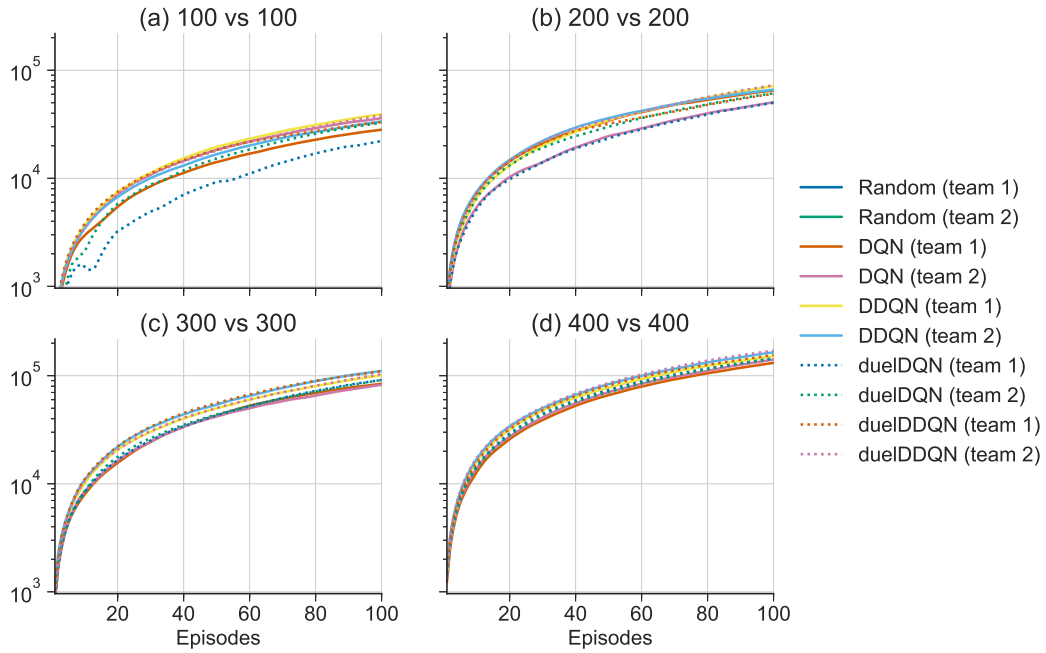

Figure 4: Cumulative reward during test phase. Average values of 12 runs with different random seeds.

It seems like most of the following strategies shown in Fig. 5 always appeared in all experiments. As we train agents, we might observe slightly variant tactics which do have the same roots. For instance, it is not unusual to observe an improved attack and/or defense tactics appear again. First, the team with the largest number of agents at time t uses some tactics to eliminate as many opponents as possible in a very short time before the opponents' counter-attack. Then, both teams reach a kind of equilibrium in which they use similar group formation strategies. And finally, agents periodically update their strategies to effectively defend themselves against the opponents. Agents often prefer short-term and aggressive strategies to long-term and safe ones.

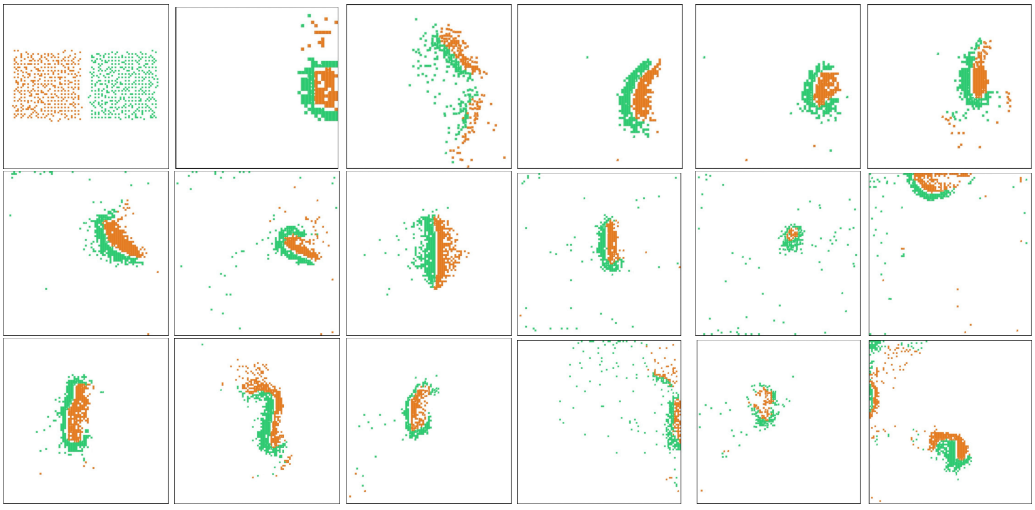

Figure 5: Some learned team strategies. Please download the videos from `https://goo.gl/cTwPZk`

See Fig. 6 for a density estimation of the number of visits by each team.

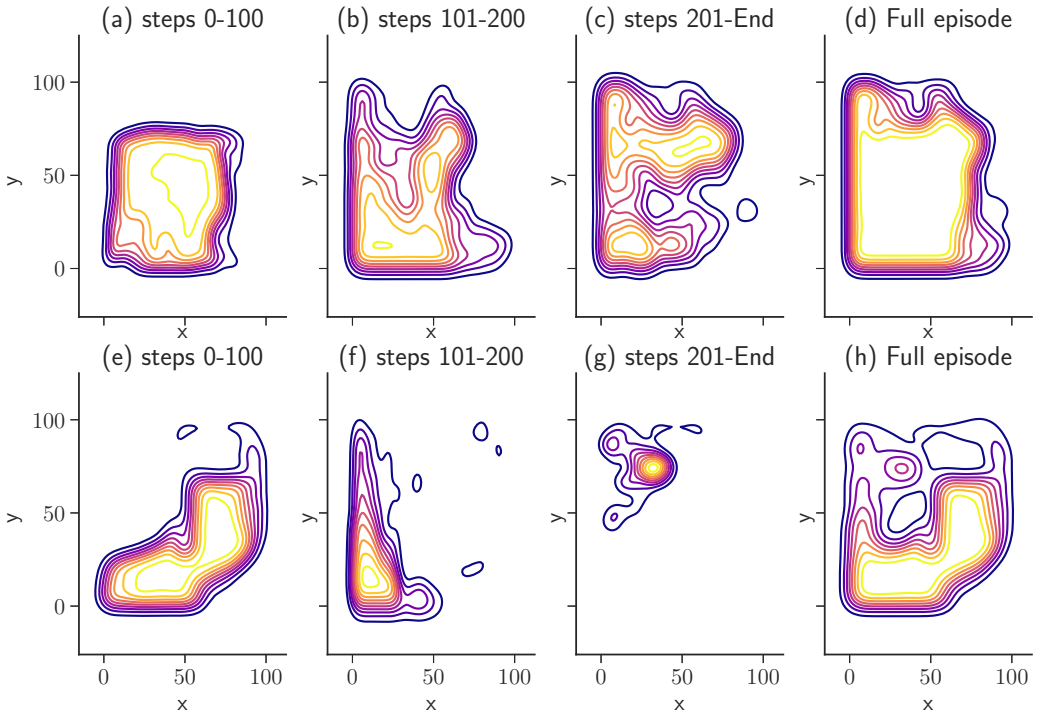

Figure 6: Kernel density estimation of the number of visits by team. DDQN, after $2,000$ episodes, $n = 400$. Team 1: top row, team 2: bottom row.

Conclusion

We demonstrated that agents keep forming groups whenever new ones are necessary. Also, the formations are dissolved whenever it is beneficial to do so. The networks have learned how to effectively position and move agents for the emergence of group formation during and after training. This proves that grouping provides greater protection against opponents. We confirmed that local behavior of an individual can conspire to determine very complex global behaviors of multi-agent systems.

While we believe that our results show how agents optimally choose their strategies to form alliances, we also recognize that further investigations are needed. These include investigating what will happen if we have obstacles inside the environment and if agents don't have the same speed or field of views. Our near future work is to address the previous points.

Acknowledgments

This work is partly supported by JSPS KAKENHI Grant Number 17KT0044.

References

[1] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27 (4):819–840, 2002.

[2] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. In *Innovations in multi-agent systems and applications-1*, pages 183–221. Springer, 2010.

[3] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998:746–752, 1998.

[4] J. P. Desai, J. Ostrowski, and V. Kumar. Controlling formations of multiple mobile robots. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, volume 4, pages 2864–2869 vol.4, May 1998.

[5] Elhadji A. O. Diallo and Toshiharu Sugawara. Learning strategic group formation for coordinated behavior in adversarial multi-agent with double dqn. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 458–466. Springer, 2018.

[6] Elhadji A. O. Diallo, A. Sugiyama, and T. Sugawara. Learning to coordinate with deep reinforcement learning in doubles pong game. In *16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 14–19, Dec 2017.

[7] Elhadji A. O. Diallo, A. Sugiyama, and T. Sugawara. Coordinated behavior by deep reinforcement learning in doubles pong game. In *The Japanese Joint Agent Workshops and Symposium (JAWS)*, Sep 2017.

[8] Kunihiko Fukushima and Sei Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern recognition*, 15(6):455–469, 1982.

[9] Hado V Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[10] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Deep reinforcement learning for swarm systems. *arXiv preprint arXiv:1807.06613*, 2018.

[11] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[12] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

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

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

[15] Andrew Y Ng. *Shaping and policy search in reinforcement learning*. PhD thesis, University of California, Berkeley, 2003.

[16] R. Olfati-Saber. Flocking for multi-agent dynamic systems: algorithms and theory. *IEEE Transactions on Automatic Control*, 51(3):401–420, March 2006. ISSN 0018-9286. doi: 10.1109/TAC.2005.864190.

[17] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

[18] David V Pynadath and Milind Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of artificial intelligence research*, 16:389–423, 2002.

[19] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH computer graphics*, volume 21, pages 25–34. ACM, 1987.

[20] Burrhus Frederic Skinner. *The behavior of organisms: An experimental analysis*. BF Skinner Foundation, 1990.

[21] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. In *NIPS*, 2016.

[22] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[23] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 16, pages 2094–2100, 2016.

[24] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[25] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[26] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.