The Journal of Engineering

The 3rd Asian Conference on Artificial Intelligence Technology (ACAIT 2019)

IET Journals
The Institution of Engineering and Technology

Distributed multi-agent deep reinforcement learning for cooperative multi-robot pursuit

eISSN 2051-3305
Received on 14th October 2019
Accepted on 19th November 2019
E-First on 30th July 2020
doi: 10.1049/joe.2019.1200
www.ietdl.org

Chao Yu[1] ✉, Yinzhao Dong[2], Yangning Li[2], Yatong Chen[2]

[1]*School of Data and Computer Science, Sun Yat-Sen University, 510006, Guangzhou, People's Republic of China*
[2]*School of Computer Science and Technology, Dalian University of Technology, Dalian 116024, People's Republic of China*
✉ *E-mail: cy496@dlut.edu.cn*

Abstract: As a popular research topic in the area of distributed artificial intelligence, the multi-robot pursuit problem is widely used as a testbed for evaluating coordinated and cooperative strategies in multi-robot systems. This study the problem of multi-robot pursuit game using reinforcement learning (RL) techniques is studied. Unlike most existing studies that apply fully centralised deep RL methods based on the centralised-learning and decentralised-execution scheme, the authors propose a fully decentralised multi-agent deep RL approach by modelling each agent as an individual deep RL agent that has its own individual learning system (i.e. individual action-value function, individual leaning update process, and individual action output). To realise coordination among agents, the limited information of other environmental agents is used as input of the learning process. Experimental results show that both distributed and centralised approaches can ultimately solve the pursuit-evasion problem in different dimensions, but the learning efficiency and coordination performance of the proposed distributed approach are much better than the traditional centralised approach.

1 Introduction

As a branch of machine learning and artificial intelligence (AI), reinforcement learning (RL) is especially suitable for problems when agents learn satisfactory policies through trial-and-error interactions with the environment [1]. RL has achieved successful applications in areas such as finance (Portfolio Management) [2], game (Alpha Go) [3], military (UAV flight strategy) [4] and so on. In some real-life problems that cannot be solved by a single agent, multiple agents must work together to achieve a common goal. When multiple agents conduct their learning in a common environment, which is also termed the multi-agent RL (MARL) problem [5], decision making for each agent becomes even more difficult due to the co-adaptation and concurrent learning of other agents.

In the past few years, many studies extended single-agent deep RL [3, 6, 7] to multi-agent settings. A straightforward way of such an extension is to simply disregard the existence of other agents in the environment and allow the agents to perform learning independently. This kind of independent learning results in a considerable reduction in the state-action representation. However, at the same time, a poor learning outcome might occur due to a lack of coordination, causing the so-called 'moving target' effect complicating general MARL problems [5]. Other studies reported to the *centralised learning and decentralised execution* (CLDE) mechanism to address the instability issue caused by independent learning [8–11]. However, the CLDE mechanism, in its essence, is still a centralised learning method that is based on all the information of agents for training. As a result, it faces the problem of (i) curse of dimensionality: the search space grows rapidly with the complexity of agent behaviours, the number of agents involved and the size of domains; (ii) limited observability and restricted communication capability: agents might not have access to the needed information for the learning update because they are not able to observe the states, actions and rewards of all other agents; and (iii) slow convergence: it may take many steps to explore all joint actions for every state, resulting in a slow convergence to the optimal policy.

In this paper, we study the MARL problem in the well-known multi-robot pursuit-evasion game, which is a classical problem for testing the performance of MARL algorithms. The multi-robot pursuit-evasion game has a variety of practical applications such as anti-terrorism, military, security, and drone operation, etc. We propose a decentralised deep RL method, the distributed duelling-DQN (deep Q network), that conquers the limitations of the existing CLDE mechanism. Unlike the centralised learning procedure in which all the agents share the same learning information, we enable each agent to learn independently using its own learning information (in terms of learning framework, learning parameters and rewards). To enable cooperation, the information of other environmental agents with limited observation range is used as input of the learning process. The feasibility of the distributed duelling-DQN algorithm is verified in different settings of pursuit-evasion games. The main contributions of this article are mainly twofold:

- We propose a distributed duelling-DQN algorithm that enables agents to learn independently, while realising cooperation through agents' local observations and joint reward functions.
- We carry out extensive experiments in different settings of multi-robot pursuit games in order to test the feasibility and efficiency of the proposed learning algorithm.

This paper is organised as follows. Section 2 discusses some related work. Section 3 introduces the basic theoretical knowledge of RL and deep RL. Section 4 presents the proposed distributed multi-agent duelling DQN algorithm, and how it solves the multi-robot pursuit problem. Section 5 verifies the feasibility of the algorithm in different experimental settings. Finally, Section 6 summarises the work and points out areas that still need improvement in the future.

2 Related work

The main research problem in this paper is about how multiple hunter agents can hunt down one or more prey agents. As a classical problem for testing the performance of MARL algorithms, the multi-robot pursuit-evasion problem has a wide range of applications, such as missile tracking, air military strategy, and drone control, and was first proposed by Isaacs in his masterpiece 'Differential Games' [12]. In 1976, Parsons [13] first described the problem graphically. Vidal *et al.* [14] applied the pursuit problem to the unmanned vehicles in the task of pursuing the escaped vehicle in an unknown environment through coordinated control of

multiple unmanned vehicles. Camci and Kayacan [15] applied the pursuit problem to the collaborative strategy of controlling multiple drones, and realised the cooperation of multiple drones to complete the queue task. At the same time, many researchers proposed different solutions to the pursuit problem. Kehagias *et al.* successfully solved this problem using two different methods: graph search and resource allocation [16, 17]. Grizou *et al.* [18] used the *Ad-Hoc* teamwork approach to successfully engage robots with unfamiliar teammates to achieve capture. Their results show that although the *Ad-Hoc* robot was added to an unfamiliar pursuit team, it did not affect the team's capture effect compared to the previous one. Su *et al.* [19] proposed a concentrating strategy for multiple hunter agents to capture multiple prey agents through Q learning and experimented on the capture in different dimensions.

MARL has gained a great deal of interest in RL research [5, 20–23]. Particularly, plenty of studies have focused on extending deep RL to multi-agent settings. Peng *et al.* [9] proposed a way to establish a communication mechanism between agents such that the agents can communicate with each other to complete cooperative tasks. Sunehag *et al.* [10] designed a learning approach that is able to information among agents. Tan [24] used a two-way RNN to model a set of agents and trains the model using the off-policy-critic algorithm. Lowe *et al.* [8] established a value decomposition network structure, which enables each agent to decompose a team value into the value of each body. The experimental results show that with the complexity of the problem, the efficiency of this method is significantly higher than that of the joint reward structure. However, due to the centralised learning procedure, these approaches are still facing significant scalability issues. Leibo *et al.* [11] allowed each agent to learn independently using traditional DQN, and used other agents as environmental information as input to the neural network. The feasibility of the algorithm is verified by two experiments of gathering and wolf packs. How conflicts manifest themselves from the competition of shared resources reveals how the problem of social dilemmas can be solved by cooperation among the agents. However, these evaluations are also limited to only two cooperative agents.

3 Background

3.1 Markov decision process (MDP) and RL

The MDP is a mathematical model used to describe the decision process in RL, which can be defined as a four-tuple: $\langle S, A, P, R\rangle$, where S is a collection of discrete environmental states $(s_i \in S | i = 1, \ldots, n)$, A refers to all discrete sets of executable actions of the agent $(a_i \in A | i = 1, \ldots, n)$, P is the probability that the action a is transferred from the state s to the state s' $(p^a_{ss'} = P[s_{t+1} = s' | s_t = s, A_t = a])$, and R is the reward value obtained by the action in the state s $(R^a_s = E[R | S_t = s, A_t = a])$. The reward function is defined as the expectation of the future reward set $r(t+1)$, $r(t+2)$, $r(t+3)$, …of the agent from time t to time T in the MDP

$$R_t = E\left[\sum_{t'}^{T} \gamma^{t'-t} r_{t'}\right] \tag{1}$$

where $\gamma = [0, 1]$ is a discount factor used to indicate the impact of future bonus values on the expected return status at the moment.

The value function is used to evaluate the value of state s when choosing action a according to strategy π:

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}\{R_t | s_t = s, a_t = a\} \\ &= E_{\pi}\left\{\sum_{k=0}^{T} \gamma^k r_{t+k+1} \middle| S_t = s, a_t = a\right\} \end{aligned} \tag{2}$$

The goal is to learn the best strategy π^* that corresponds to the maximum action-value function $Q^*(s, a)$:

$$\begin{aligned} Q^*(s,a) &= \max_{\pi} Q^{\pi}(s,a) = E[r(s,a) + \gamma V^*(s')] \\ &= \sum_{s' \in S} p(s'|s,a)\left[r(s,a,s') + \gamma \max_{a' \in A} Q^*(s',a')\right] \\ &= \left[r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} Q^*(s',a')\right] \end{aligned} \tag{3}$$

3.2 Main algorithms for RL

3.2.1 Q-learning: The most well-known algorithm for RL is the Q learning algorithm [25], with the update rule given as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r + \gamma \max_{a} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right] \tag{4}$$

where α is the learning rate, indicating the degree of difference between the Q value in the previous step and the newly calculated Q value, to control the rate at which the Q value is updated, γ is the discount factor for future rewards, and the content in square brackets is the time difference error. To enable exploration, the agent will choose the action of max Q value with the probability of $1 - \varepsilon$ and try to explore new actions with the probability of ε.

3.2.2 Deep RL: Deep RL combines the ability of deep learning and RL to extract environmental information with the decision-making ability of RL. The DQN [8] applies neural networks as an approximator of Q values. When the Q value is updated, the target Q value calculated by the bonus value and the Q value is $r + \gamma \max_a Q(s', a', \theta^-)$, and the target Q value is used as a label to make an estimation of Q. θ^- are parameters of the target network, which are updated periodically, and held fixed in between. The value is constantly approaching the target Q value. The loss function of the Q network training is defined as the squared difference for optimisation:

$$L(\theta) = E_{s,a,r,s'}[\left(r + \gamma \max_{a'} Q(s', a', \theta^-) - Q(s, a, \theta)\right)^2] \tag{5}$$

Double-DQN [26] is to solve the problem of overestimation in traditional DQN. Since the previous Q target neural network is updated in the following way: $Y_t^{\mathrm{DQN}} = r + \gamma \max_a Q(S_{t+1}, a; \theta_t^-)$, there is inherently an error in the $Q_{\max}$ based on the target neural network. Then, the neural network parameters are updated toward the Q target with the maximum error, resulting in an overestimation of the maximum Q value. The two networks in DQN can be used to change the action of selecting the maximum Q value to solve the problem of overestimation: use the estimation network (θ) to estimate the Q value of the target network (θ^-), and select the action corresponding to the maximum Q ($\max_a Q$), and then the target network (θ^-) is used to estimate the Q value of the optimal action selected by the estimated network (θ). The target Q value can be rewritten as follows:

$$Y_t^{D-\mathrm{DQN}} = r + \gamma Q\left(S_{t+1}, \arg\max_{a} Q(S_{t+1}, a; \theta_t^-)\right) \tag{6}$$

Later, Hausknecht and Stone [27] proposed the deep recurrent Q network (DRQN), which uses the long-term and short-term memories (LSTMs) unit structure to model partial state information.

4 Model of cooperative multi-robot pursuit

4.1 Distributed multi-agent duelling-DQN algorithm

In duelling-DQN [28], the last convolutional layer is divided into a state value function $V(s)$, which presents the value of the static state itself, and an advantage function $A(s, a)$, which represents the additional value from an action in a certain state. The value function only outputs one value for each state while the advantage layer outputs N actions for each action in a certain state. These two

Initialize experience pool D, the capacity of experience pool M;
for i from 0 to n (Number of agents) **do**
  Initialize the parameters θ of neural network for each agents (action-value function Q) or load the previously trained parameters θ;
end for
for $m = 1$ to $max - iteration$ **do**
  Initialize environment and load the initial observation state s_0^i;
  for t from 0 to T (400) **do**
    for i from 0 to n **do**
      Agent i chooses and performs an action a_t^i as $a_i^t = \begin{cases} random\ action & with\ probability\ \epsilon \\ argmax_a Q(s_t^i, a; \theta) & with\ probability\ 1-\epsilon \end{cases}$
      Agent i performs a_t^i and gets reward r_t^i in the next s_{t+1}^i;
      Store transition $(s_t^i, a_t^i, r_t^i, s_{t+1}^i)$ into experience pool D;
    end for
  end for
  for $k = 1$ to $number\ of\ batches(1000)$ **do**
    Load $(s_t^i, a_t^i, r_t^i, s_{t+1}^i)$ from experience pool;
    Target Q value: $y_t^i = r_t^i + \gamma * max_{a_{t+1}^i} Q(s_{t+1}^i, a_{t+1}^i; \theta')$;
    Use the squared loss function to update the parameter of neural network: $\theta_{k+1} \leftarrow \theta_k + \bigtriangledown_\theta (y_t^i - Q(s_{t+1}^i, a_{t+1}^i; \theta))^2$;
  end for
end for

Fig. 1 *Algorithm 1: Distributed duelling-DQN*

values are aggregated together to obtain the Q value of performing an action in a certain state

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) \tag{7}$$

where θ is the parameters of convolutional layers, while β and α are the parameters of the two fully connected layers for V and A, respectively.

In order to increase the stability and efficiency of this algorithm, we can have

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \max_{a' \in |A|} A(s, a'; \theta, \alpha) \right) \tag{8}$$

Let the advantage function value of the optimal action be 0 such that $Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta)$. Specific action can be calculated as follows:

$$a^* = \arg\max_{a' \in A} Q(s, a'; \theta, \alpha, \beta) = Q(s, a; \theta, \beta) \tag{9}$$

By setting the advantage function as the difference between the advantage function of individual action and the mean advantage function of all actions, the unique value function can be obtained by the following equation:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|A|} \sum_{a'} A(s, a'; \theta, \alpha) \right) \tag{10}$$

In order to model real-world situations where agents only have limited observation capabilities, agents in the multi-robot pursuit game can only receive the state information in a certain observation range with radius r. To enable distributed learning, each agent learns to interact with the environment (other preys and predators) to maximise its own reward. Each agent has an independent Q function $Q_i: S_i \times A_i \rightarrow R$ by using duelling-DQN, and has its own independent experience pool to store the learned information $\{(s, a, r_i, s')_t : t = 1, \ldots, T\}$. As a result, each agent can update its own Q-value neural network independently using the following equation:

$$Q_i(s, a) \leftarrow Q_i(s, a) + \alpha \left[r_i + \gamma \max_{a' \in A_i} Q_i(s', a') - Q_i(s, a) \right] \tag{11}$$

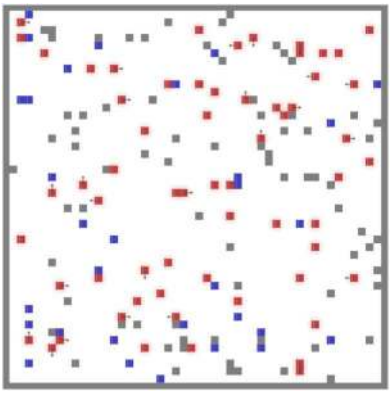

Fig. 2 *Schematic diagram of a pursuit-evasion environment*

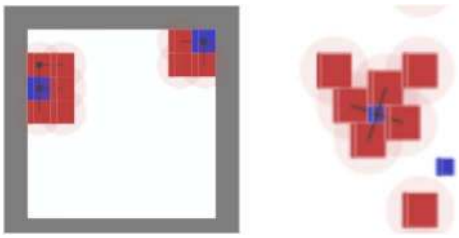

Fig. 3 *Illustration of successful capture*

Each agent is learning independently, by considering other agents as an input of the varying environment. Although each agent has a separate observation and is responsible for its independent behaviours, cooperation can be achieved through the joint reward generated after all the agents perform actions together (Fig. 1).

4.2 Solving the pursuit-evasion game

In the pursuit-evasion game, a group of predator agents are trained to capture the prey agents cooperatively. The pursuit-evasion environment is changing dynamically in each step. Therefore, predators must adapt to the states continuously for better actions. According to a different research focus on cooperation, scheduling or communication problems between predators, the pursuit-evasion problems can be handled in various ways. In this paper, the pursuit-evasion problem is designed to a two-dimensional bounded rectangular environment S, and its size is $N \times N$. In this environment, there are a set of $P = \{P_1, P_2, \ldots, P_{(n_p)}\}$ predator agents (n_p is the number of predator agents), and a set of $E = \{E_1, E_2, \ldots, E_{(n_E)}\}$ prey agents (n_e is the number of prey agents). Both the two types of agents can only run within the boundary of S, and there are m randomly placed obstacles in this area. In the initial state, the agents are randomly distributed, and the predator agents are controlled independently to achieve cooperation. The goal of the predator agents is to avoid obstacles and capture the prey agents together as quickly as possible. Each game lasts 400-time steps. When it reaches 400-time steps, the game ends up and gets the total rewards of all predator agents during the time period. Fig. 2 is a schematic diagram of the environment, and Fig. 3 is an illustration of the successful capture situation. Algorithm 1 gives the main procedure of the proposed distributed duelling-DQN algorithm.

In order to motivate cooperation among the predators, the speed of prey agents is set higher than that of the predator agents. Therefore, these predator agents cannot achieve capture by simply pursuing. There are nine movement directions for the two types of agents: moving up, down, right, left, up-left, up-right, down-left, down-right, and staying still. The predator agent has an attack ability that can attack the prey agent, and the attack range is only one surrounding grid. Each agent has an observation range, which means that they cannot get all the state information in the whole environment. Their inputs are only the information in the range that they can observe in a circle with its own centre radius R. For the predator agents, the distributed duelling-DQN algorithm is used.

Table 1 Parameter of the Agents

Parameter type	Hunter agent	Prey agents
size	1 × 1 or 2 × 2	1 × 1
speed	1 cell every step	1.5 every step
attack range	a circle with a radius of 1	none
observation range	a circle with a radius of 3	none

Table 2 Parameter of the algorithm

Parameter type	Parameter values
total time-steps	400
batch size	512
learning rate	0.0001
reward decay	0.99
target update	1000
memory size	2^{20}

For the prey agents, in order to simulate the intelligence of the prey agents, we also use the same distributed duelling-DQN algorithm to train the prey agents for a period of time. Therefore, they have a certain ability to escape the attack of predator agents autonomously.

5 Experiment

This section first gives the parameter settings for the experiments and then discusses the results and analysis in different experimental settings. The codes and illustration videos are released on the web page: https://github.com/SadAngelF/Distributed-Dueling-DQN.

5.1 Parameter settings

In this research, we adopted the simulation environment Magent [29] developed by the University of London and Shanghai Jiao Tong University. The basic parameters for the environment are given in Table 1. Some parameters of the algorithm are given in Table 2, where the batch size is the data set size for gradient descent update based on neural network, target update is the update frequency of the neural network, and memory size is a measurement of the experience pool.

5.2 Experimental results and analysis

5.2.1 Design of reward function: Among all the basic components, the reward function may be at the core of an RL process. Since it encodes the goal information of a learning task, a proper formulation of reward functions plays the most crucial role in the success of RL. We first test a direct reward function as follows: (i) When four predator agents attack a prey agent at the same time, each agent will get a reward of +4, and (ii) each movement of hunter agents causes a reward of −0.2, in order to make the agents learn to hunt the prey in the shortest path. However, under this reward function, the predator agents could not learn to capture the prey agents successfully, as given in Fig. 4. The result shows that the agents can learn to maximise the reward value, but the reward gradually converges to zero. It indicates that this direct reward function is too sparse for the agents to learn satisfactory strategies in proper time. In order to introduce more intermediate reward, we design the reward function as follows: (i) If a predator agent attacks a prey agent individually, it will get a reward of +1; (ii) If two predator agents attack a prey agent together, each agent will get a reward of +2; (iii) If three predator agents attack a prey agent together, each agent will get a reward of +3, and (iv) If four predator agents attack a prey agent together, each agent will get a reward of +4. All the remaining section of experiments is based on this reward function.

5.2.2 Centralised duelling-DQN versus distributed duelling-DQN: Fig. 5 gives the learning curves of the centralised duelling-DQN and the distributed duelling-DQN algorithms in a 20 × 20

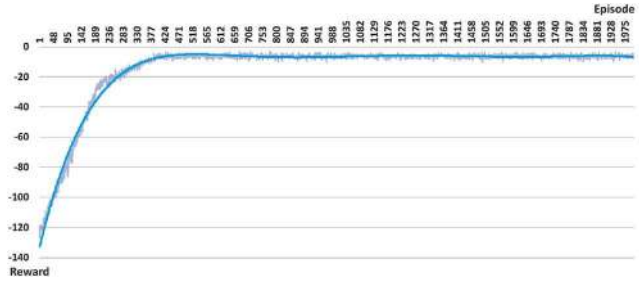

Fig. 4 *Learning curve using the direct sparse reward function*

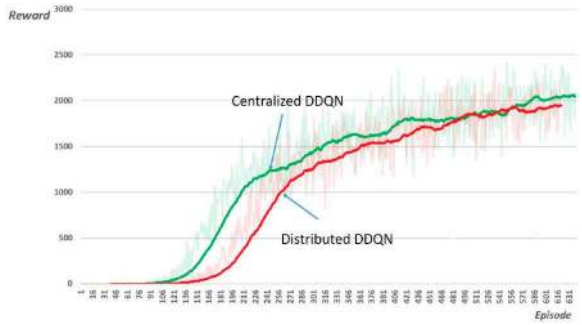

Fig. 5 *Centralised duelling DQN versus distributed duelling DQN*

environment with eight predator agents and two prey agents. Although the total reward value at the end of convergence is similar, it is clear that the distributed duelling DQN is far more efficient than the centralised DQN, especially in the early stage. This result fully demonstrates the benefits of distributed learning among agents, compared to the centralised learning methods when agents either communicate freely with a central controller and select their actions according to those indicated by the central controller, or have full observability of the environment to receive the joint-state-action information of all agents to control the learning process synchronously.

5.2.3 Cooperative capability of agents: In the pursuit-evasion problem, four predator agents need to work together to capture the prey agents. Evaluating cooperative capability is also one of the criteria to test the effectiveness of the learning algorithm. In this experiment, there are four predator agents and one prey agent in the 15 × 15 environment. We propose a way to judge the cooperative capability by using the percentage of the reward value contributed by each of the four predator agents to the total reward value by the whole system. If the performance of the four agents is almost the same, it indicates a better cooperative capability. The cooperative capability of the agents, P, can be calculated as follows:

$$P = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(R_i - \mu)} \tag{12}$$

where $R_i = (1/(T_2 - T_1))\sum_{t=T_1}^{t=T_2} r_i$ is the reward of agent i during the whole time, and $\mu = (1/N)\sum_{i=1}^{N} R_i$. A smaller P means the stronger cooperative capability between the agents.

It can be seen from Fig. 6 that the performance of the four agents is almost the same over the entire period, causing a cooperative capability $P = 2.7386$.

In order to reveal the cooperativeness during different learning stages, we divide the whole learning process of 2100 episodes into three different stages. The period of 1–700 episodes are the early stage, the period of 701–1400 episodes are in the middle stage, and the period of 1401–2100 are the later stage. Figs. 7 and 8 show the corresponding cooperative capabilities of the agents in the three stages. At the beginning of learning, the agents may not be able to cooperate over their behaviours, so the performance difference between the agents is still huge. In learning proceeds, they can learn to cooperate to complete the capture task, so the difference in

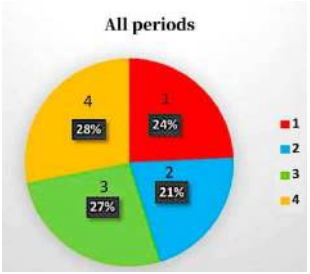

Fig. 6 *Percentage of the contribution of four predator agents*

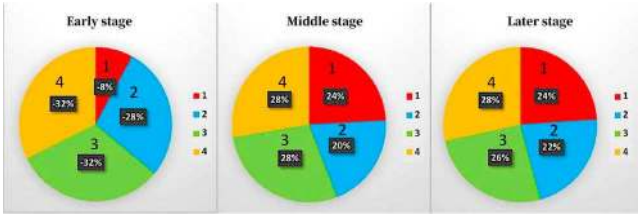

Fig. 7 *Percentage of the contribution of four predator agents during three different stages*

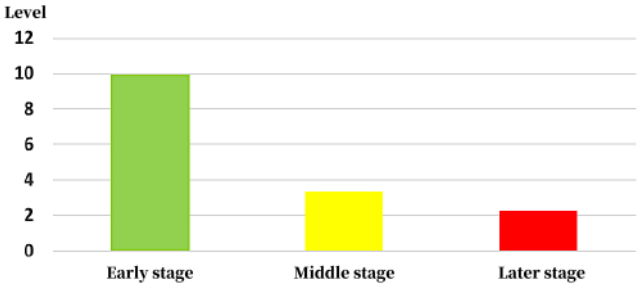

Fig. 8 *Cooperative ability of four predator agents during three different stages*

the performance between the agents in the later period will be much smaller, which indicates that their cooperative ability has been greatly improved.

5.2.4 Influence of random obstacles: Fig. 9 shows the learning efficiency in a 20×20 environment involving 40 randomly distributed obstacles. The agents show excellent learning efficiency with obstacles since the agents have more accurate information to help them judge the state and thus perform an action. Another reason can be speculated that obstacles can hinder the move of prey agents, which can facilitate the learning of predators accordingly.

5.2.5 Influence of ε descent methods: We also evaluated different ways of descending the exploration parameter ε in the learning process. Fig. 10 gives the result in the same 20×20 environment involving eight predator agents and two prey agents. The result shows that the learning efficiency of the agent with piecewise descent is higher than the exponential descent method. Therefore, a careful selection of exploration strategies is crucial in affecting the final performance of RL algorithms.

5.2.6 Comparison of different learning algorithms: Fig. 11 compares the learning performance using different deep RL algorithms in a 30×30 environment with eight predator agents and two prey agents. The result shows that the learning efficiency of distributed duelling DQN is comparable with many existing algorithms. Although the final reward of distributed duelling DQN is a bit lower than the double-DQN and naive DQN algorithms, the learning efficiency in the early stage is better. The A2C [30] and the DRQN algorithms all perform far worse than the distributed duelling DQN algorithm, which fully demonstrates the benefits of distributed learning of agents using only limited partial observations.

Fig. 9 *Influence of random obstacles*

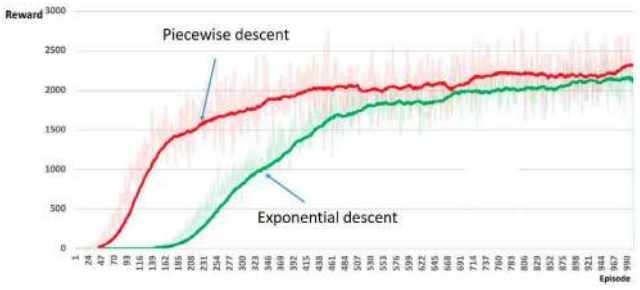

Fig. 10 *Influence of ε descent methods*

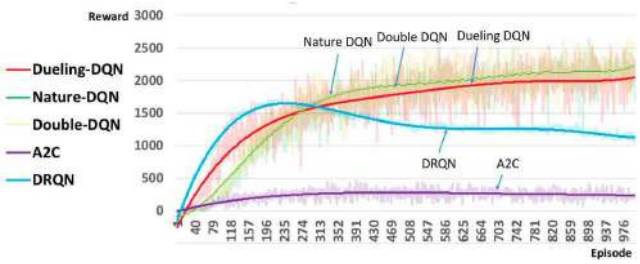

Fig. 11 *Different algorithms to solve the pursuit-evasion problems*

6 Conclusion

In this paper, we propose a distributed duelling-DQN algorithm and apply it to the multi-robot pursuit problem. Unlike most existing studies that apply fully centralised deep RL methods based on the CLDE scheme, the decentralised duelling-DQN algorithm models each agent as an individual deep RL agent that has its individual learning system (i.e. individual action-value function, individual leaning update process, and individual action output). To realise coordination among agents, the information of other environmental agents with limited observation range is used as input of the learning process. Various experimental settings such as different scales of environments, different deep RL algorithms including DQN, double-DQN, A2C, duelling-DQN, DRQN are used to verify the feasibility of the proposed algorithm. Experimental results show that both distributed and centralised approaches can ultimately solve the pursuit-evasion problem in different dimensions, but the learning efficiency and coordination performance of the proposed distributed approach are much better than the traditional centralised approach. In the future, we plan to evaluate the proposed algorithm in more complex game domains where the environment is changing rapidly; for example, the obstacles are changing continuously.

7 Acknowledgments

This work was supported by the Joint Key Program of National Natural Science Foundation of China and Liaoning Province under grant U1808206, Ministry of Military Equipment Development of China under grant no. 61403120203, the Dalian High-Level Talent Innovation Support Program under grant no. 2017RQ008, and the Dalian Science and Technology Innovation Fund under grant no. 2018J12GX046.

8 References

[1] Sutton, R.S., Barto, A.G.: '*Reinforcement learning: an Introduction*' (MIT Press, Cambridge, MA, USA, 2018)

[2] Jiang, Z., Xu, D., Liang, J.: 'A deep reinforcement learning framework for the financial portfolio management problem', arXiv preprint arXiv:1706.10059, 2017

[3] Silver, D., Huang, A., Maddison, C.J., *et al.*: 'Mastering the game of Go with deep neural networks and tree search', *Nature*, 2015, **529**, (7587), pp. 484–489

[4] Olfati-Saber, R., Fax, J.A, Murray, R.M.: 'Consensus and cooperation in networked multi-agent systems', *Proc. IEEE*, 2007, **95**, (1), pp. 215–233

[5] Busoniu, L., Babuska, R., De Schutter, B.: 'A comprehensive survey of multiagent reinforcement learning', *IEEE Trans. Syst. Man Cybern.-C, Appl. Rev.*, 2008, **38**, (2), pp. 156–172

[6] Mnih, V., Kavukcuoglu, K., Silver, D., *et al.*: 'Human-level control through deep reinforcement learning', *Nature*, 2015, **518**, (7540), p. 529

[7] Schulman, J., Levine, S., Abbeel, P., *et al.*: 'Trust region policy optimization'. ICML, Lille, France, 2015, pp. 1889–1897

[8] Lowe, R., Wu, Y., Tamar, A., *et al.*: 'Multi-agent actor-critic for mixed cooperative-competitive environments'. Advances in Neural Information Processing Systems, Long Beach, CA, USA, 2017, pp. 6379–6390

[9] Peng, P., Wen, Y., Yang, Y., *et al.*: 'Multiagent bidirectionally-coordinated nets: emergence of human-level coordination in learning to play StarCraft combat games', arXiv preprint arXiv:1703.10069, 2017

[10] Sunehag, P., Lever, G., Gruslys, A., *et al.*: 'Value-decomposition networks for cooperative multi-agent learning'. AAMAS, Sao Paulo, Brazil, 2017, pp. 2085–2087

[11] Leibo, J.Z., Zambaldi, V., Lanctot, M., *et al.*: 'Multi-agent reinforcement learning in sequential social dilemmas'. AAMAS, Sao Paulo, Brazil, 2017, pp. 464–473

[12] Isaacs, R., Philip, R.: 'Differential games: a mathematical theory with applications to warfare and pursuit', *Control Opt.*, 1966, **17**, (2), pp. 60–60

[13] Parsons, D.T.: 'Pursuit-evasion in a graph', *Theory and Applications of Graphs. Lecture Notes in Mathematics* (Springer, New York City, NY, USA, 1976), pp. 426–441

[14] Vidal, R., Shakernia, O., Kim, H.J., *et al.* 'Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation', *IEEE Trans. Robotics Autom.*, 2002, **18**, (5), pp. 662–669

[15] Camci, E., Kayacan, E.: 'Game of drones: UAV pursuit-evasion game with type-2 fuzzy logic controllers tuned by reinforcement learning'. IEEE Int. Conf. on Fuzzy Systems, Vancouver, Canada, 2016, pp. 618–625

[16] Kehagias, A., Hollinger, G., Singh, S.: 'A graph search algorithm for indoor pursuit/evasion', *Math. Comput. Model.*, 2009, **50**, (9), pp. 1305–1317

[17] Hollinger, G., Singh, S., Kehagias, A.: 'Improving the efficiency of clearing with multi-agent teams', *Int. J. Robotics Res.*, 2010, **29**, (8), pp. 1088–1105

[18] Grizou, J., Barrett, S., Stone, P., *et al.*: 'Collaboration in Ad hoc teamwork: ambiguous tasks, roles, and communication'. AAMAS Adaptive Learning Agents (ALA) Workshop, Singapore, 2016

[19] Su, Z.B., Lu, J.L., Tong, L.: 'Strategy of cooperative hunting by multiple Mobile robots', *J. Beijing Inst. Technol.*, 2004, **5**, (8), pp. 403–406

[20] Yu, C., Wang, X., Xu, X., *et al.*: 'Distributed multiagent coordinated learning for autonomous driving in highways based on dynamic coordination graphs', *IEEE Trans. Intell. Transp. Syst.*, 2020, **21**, (2), pp. 735–748, doi: 10.1109/TITS.2019.2893683

[21] Yu, C., Zhang, M., Ren, F., *et al.*: 'Emotional multiagent reinforcement learning in spatial social dilemmas', *IEEE Trans. Neural Netw. Learn. Syst.*, 2015, **26**, (12), pp. 3083–3096

[22] Yu, C., Zhang, M., Ren, F., *et al.*: 'Multiagent learning of coordination in loosely coupled multiagent systems', *IEEE Trans. Cybern.*, 2015, **45**, (12), pp. 2853–2867

[23] Yu, C., Zhang, M., Ren, F.: 'Collective learning for the emergence of social norms in networked multiagent systems', *IEEE Trans. Cybern.*, 2014, **44**, (12), pp. 2342–2355

[24] Tan, M.: 'Multi-agent reinforcement learning: independent vs. Cooperative agents'. Machine Learning Proc., Amherst, MA, USA, 1993, pp. 330–337

[25] Watkins Christopher, J.C.H., Dayan, P.: 'Q-learning', *Mach. Learn.*, 1992, **8**, (3–4), pp. 279–292

[26] Van Hasselt, H., Guez, A., Silver, D.: 'Deep reinforcement learning with double Q-learning'. AAAI, Austin, TX, USA, 2015

[27] Hausknecht, M., Stone, P.: 'Deep recurrent Q-learning for partially observable MDPs'. 2015 AAAI Fall Symp. Series, Arlington, VA, USA, 2015

[28] Wang, Z., Schaul, T., Hessel, M., *et al.*: 'Dueling network architectures for deep reinforcement learning'. Conf. on Int. Conf. on Machine Learning, New York City, NY, USA, 2016, pp. 1995–2003

[29] Zheng, L., Yang, J., Cai, H., *et al.*: 'MAgent: a many-agent reinforcement learning platform for artificial collective intelligence'. AAAI, New Orleans, LA, USA, 2018

[30] Grondman, I., Busoniu, L., Lopes, G.A., *et al.*: 'A survey of actor-critic reinforcement learning: standard and natural policy gradients', *IEEE Trans. Syst. Man Cybern., C*, 2012, **42**, (6), pp. 1291–1307