2018 24th International Conference on Pattern Recognition (ICPR)
Beijing, China, August 20-24, 2018

Learning Evasion Strategy in Pursuit-Evasion by Deep Q-network

Jiagang Zhu*†, Wei Zou*‡, Zheng Zhu*†
*Institute of Automation, Chinese Academy of Sciences, Beijing, China
†University of Chinese Academy of Sciences, Beijing, China
‡TianJin Intelligent Tech.Institute of CASIA Co., Ltd
Email: zhujiagang2015@ia.ac.cn, wei.zou@ia.ac.cn, zhuzheng2014@ia.ac.cn

***Abstract*—This paper presents an approach for learning the evasion strategy for the evader in pursuit-evasion against the pursuers with Deep Q-network (DQN). To give the immediate reward to the agent, we handcraft a reward function, which considers both the evader escaping from being surrounded by the pursuers and keeping distance from the pursuers. This is a combination of the artificial potential field method with deep reinforcement learning. Our learned evasion strategy is verified by a series of experiments in three different game scenarios. The training stability and the value function are analyzed respectively. The three learned agents are compared with a random agent and a repulsive agent. We show the effectiveness of our method.**

I. Introduction

Deep reinforcement learning (DRL) [8] which combines deep learning and reinforcement learning has been developed rapidly in recent years. It has been applied to robotic motion control [18], video prediction [19] and complex games, such as chess [16] and Go [10], [28]. Deep Q-network (DQN) [1][2] is one of the most famous methods of deep reinforcement learning. It showed the ability of achieving human experts in several games of Atari 2600. Compared with the conventional reinforcement learning by handcrafted features for state representation, DQN is an end-to-end way of learning the action policy directly from the high-dimensional image data by using Convolutional Neural Network (CNN) [3].

In this research, we apply DQN to pursuit-evasion games. Specifically, we aim to strengthen the evasion strategy of the evader by DQN, which is rarely addressed in previous works of pursuit-evasion, where most of works focus on designing the pursuit strategies.

In pursuit-evasion, a number of pursuers attempt to chase another set of evaders. Researches on pursuit-evasion can be roughly divided into three categories: path planning [1], [2], multi-robot coordination [3]–[5] and game theory [6], [7]. In path planning, an efficient hunting trajectory is determined to catch the evader with the knowledge about the targets and the surroundings [1], [2]. In multi-robot coordination, the pursuers try to capture or enclose the evaders by forming a reasonable and efficient troop formation [3]–[5]. Researches on game theory between groups of robots focus on analyzing their hunting or evasion strategies and placing these two groups on the reciprocal position [6], [7].

A number of methods based on reinforcement learning are proposed for pursuit-evasion games, such as fuzzy reinforcement learning [22], [23], [26], [27], multi-agent reinforcement learning [14], [25], policy gradient [24], deep Q-network [14] [24]. The most closely related works to ours are [27] and [14]. [27] explored the use of a fuzzy Q-learning for the evader strategy learning in the game of guarding territory. [14] proposed multi-agent Deep Q-Network (MADQN), which represents the global state by a image-like tensor for arbitrary number of pursuer agents. The evaders in [14] move in the direction that takes them furthest from the closest pursuer. Different from above two works, we consider using Deep Q-network for the end-to-end learning of the evasion strategy of the single evader agent directly from high-dimensional visual information. Moreover, a reward function considering both the evader escaping from being surrounded by the pursuers and keeping distance from the pursuers is handcrafted to give the immediate reward to the agent. This is a combination of the artificial potential field method with deep reinforcement learning. The effectiveness of our learned evasion strategy is verified by a series of experiments.

This paper is organized as follows. Section II introduces the method of learning the evasion strategy in pursuit-evasion. Section III presents the game settings for the latter experiments. Section IV shows the learning results and gives detailed analysis. Conclusions are given in Section V.

II. Methods

In this section, Deep Q-network is firstly introduced. Then how to incorporate the artificial potential field method into the reward function is described.

A. Deep Q-network

Fig. 1 shows the architecture of the system. In reinforcement learning, the agent interacts with an environment E at each time-step t by observing a state s_t, selecting an action a_t, receiving a reward r_t and transitioning to a new state s_{t+1}. Deep Q-Networks (DQN) combines convolutional neural network (CNN) [15] with Q-learning. DQN solves the training instability of Q-learning in two ways. First, it uses experience replay which stores the experience $e_t = (s_t, a_t, r_t, s_{t+1})$ into a replay memory $M_t = \{e_1, ..., e_t\}$ at each time-step t during an agent's interaction with the environment. Second, it maintains two separate Q-networks: $Q(s, a; \theta)$ and $Q(s, a; \theta^-)$ with the current parameters θ and the old parameters θ^- respectively.

978-1-5386-3788-3/18/$31.00 ©2018 IEEE

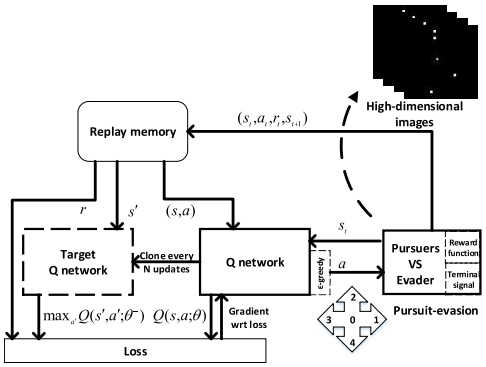

Fig. 1. Overall architecture of the system.

The current parameters θ are updated at every update iteration i to minimize the mean-squared Bellman error by optimizing the loss function,

$$L(\theta_i) = E[(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2] \quad (1)$$

For each sample $(s, a, r, s') \sim U(M)$ which is sampled uniformly from the replay memory M, the current parameters θ are updated at each update i by stochastic gradient descent. The model weights at each iteration are adjusted by performing (2)

$$\theta_{i+1} = \theta_i + \alpha \nabla_\theta L(\theta_i) \quad (2)$$

Finally, the Q-network selects one action at each time-step t by following ϵ-$greedy$ policy. A convolutional neural network is taken for the Q-network. The input to the CNN consists of an 84×84×4 image. The first hidden layer has 32 filters of 8×8 with stride 4 with the input image. The second hidden layer has 64 filters of 4×4 with stride 2. The third convolutional layer has 64 filters of 3×3 with stride 1. This is followed by the final hidden layer, which is a fully-connected linear layer with 512 nodes. Each hidden layer is followed by a rectifier nonlinearity [15]. The output layer is a fully-connected linear layer with a single output for each valid action. The number of valid action is 5.

B. Reward Function

Training with reinforcement learning requires designing an appropriate reward function. Hence for the evader agent in our pursuit-evasion game, a reward function is designed, which aims to improve the evasion strategy of the evader by giving reward to the evader agent at each time-step. In order to maximize the distance between the evader and all the pursuers, and the existing time of the evader in one game episode, a reward function is defined as follows

$$r_t = \begin{cases} 1 & if\ surounded\ and\ action\ is\ appropriate \\ 1 & if\ not\ surrounded\ and \\ & dist_{min} > D\ and\ thr_t < thr_{t-1} \\ -1 & if\ dist_{min} < D/2 \\ 0 & otherwise \end{cases} \quad (3)$$

where r_t is the immediate reward provided at time-step t. Suppose the evader and the i-th pursuer's position are (x_e, y_e) and (x_p^i, y_p^i) respectively, then the evader is regarded being surrounded if both $\min_{i=1,...,m} x_p^i < x_e < \max_{i=1,...,m} x_p^i$ and $\min_{i=1,...,m} y_p^i < y_e < \max_{i=1,...,m} y_p^i$ are true, where m is the number of pursuers.

The action's appropriateness in (3) is evaluated by the artificial potential field method. As shown in (4), $\vec{F}_{rep}(q_p^i)$ is the repulsive force generated from the i-th pursuer to the evader, where $\rho(q_p^i)$ is the Euclidean distance between the evader and the i-th pursuer, ρ_0 is the threshold distance that defines whether there exists repulsive force between the evader and pursuers. $\vec{n}_i = \left[\frac{x_e - x_p^i}{\rho(q_p^i)}, \frac{y_e - y_p^i}{\rho(q_p^i)}\right]^T$ is the direction of the repulsive force $\vec{F}_{rep}(q_p^i)$, where $i = 1, ..., m$. Similarly, $\vec{F}_{rep}(q_b^j)$ is the repulsive force exerted from the j-th boundary to the evader, where distance is the minimal Euclidean distance between the evader and each of boundaries, where $j = 1, ..., 4$, each corresponds to left, right, up, down boundary of the environment. As shown in (5), $\vec{F}_R$ is the sum over the repulsive forces on the evader exerted from m pursuers ($\sum_{i=1}^{m} \vec{F}_{rep}(q_p^i)$) and four boundaries ($\sum_{j=1}^{4} \vec{F}_{rep}(q_b^j)$)of the environment. $\beta_F = \arctan(F_y/F_x)$ is the direction of $\vec{F}_R$. Action is appropriate when the moving direction of the evader satisfies $\beta_e \in [\beta_F - \pi/4, \beta_F + \pi/4]$

$$\vec{F}_{rep}(q_p^i) = \begin{cases} \eta \left[\frac{1}{\rho(q_p^i)} - \frac{1}{\rho_0}\right] \frac{1}{\rho^2(q_p^i)} \vec{n}_i & \rho(q_p^i) \le \rho_0 \\ 0 & \rho(q_p^i) > \rho_0 \end{cases} \quad (4)$$

$$\vec{F}_R = \sum_{i=1}^{m} \vec{F}_{rep}(q_p^i) + \sum_{j=1}^{4} \vec{F}_{rep}(q_b^j) = [F_x, F_y]^T \quad (5)$$

In (3), $dist_{min} = \min_{i=1,...,m} \rho(q_p^i)$ is the minimal Euclidean distance between the evader and the pursuers at time-step t. The evader is captured when $dist_{min} < thres$. It means the distance between the evader and any one pursuer is less than a threshold, which is the end of one episode. D is the distance that defines whether the evader is safe or not and whether the agent should be rewarded or not. thr_t in (3) could be detailed into (6). It is the assessment of the threat to the evader exerted by all the pursuers and boundaries, which is the sum over $thrp_t$ and $thrb_t$. They are all denoted by Euclidean distance in Gaussian kernel function.

$$\begin{aligned} thrp_t &\propto \sum_{i=1}^{m} e^{-\frac{(x_e - x_p^i)^2 + (y_e - y_p^i)^2}{2\sigma^2}} \\ thrb_t &\propto e^{-\frac{(x_e - x_b^{left})^2}{2\sigma^2}} + e^{-\frac{(x_e - x_b^{right})^2}{2\sigma^2}} \\ &\quad + e^{-\frac{(y_e - y_b^{up})^2}{2\sigma^2}} + e^{-\frac{(y_e - y_b^{down})^2}{2\sigma^2}} \\ thr_t &= thrp_t + thrb_t \end{aligned} \quad (6)$$

where x_b^{left}, x_b^{right} and y_b^{up}, y_b^{down} are the horizontal coordinates of left, right boundary and the vertical coordinates of up, down boundary of square map respectively.

It is noticed that there are two cases when the agent receives a positive reward 1 from reward function (3). One case is the evader takes the appropriate action when it is surrounded by the pursuers. This intuition guides the evader into escaping from being enclosed. The other is that the current threat thr_t is smaller than the threat of last time-step threat thr_{t-1} when their minimal distance is larger than D, implying the evader should move further than the last time-step even if it is safe temporarily. This reward function is a union of escaping from being surrounded and keeping distance. It is handcrafted to give immediate reward r_t to the agent at each time-step t, thus improving the evasion strategy of the evader step by step.

III. Game Settings

A. Environment

As shown in Fig. 2(a), the environment of pursuit-evasion is represented by an image with width and length being w pixels and l pixels respectively. Each of the pursuers and the evader has a size of $n \times n$ pixels. For the easy learning of the network, the gray scale of the environment is set as $gray1 = 0$, while the pursuers are $gray2 = 200$, the evader is $gray3 = 100$. At each time-step t, the pixels representing the pursuers and the evader will sequentially move in the image according to their separate strategies. In Fig. 1, a tuple of consecutive F frames will be fed into the Q-network to derive the speed, direction, or other hidden complex characteristics.

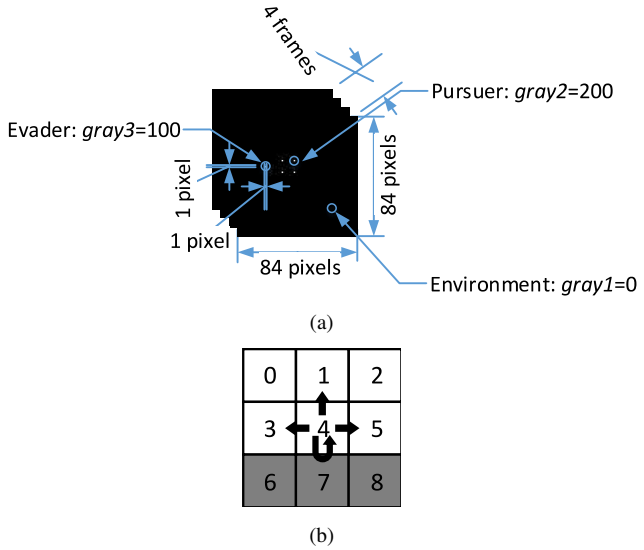

Fig. 2. Environment setting of pursuit-evasion. Best viewed in color.

The evader is defined to possess 5 legal actions: keeping still, moving left, moving right, moving up and moving down. All the pursuers can drive straight towards the evader with a constant speed of 2 pixels per second, while the evader is quicker with a speed of 3 pixels per second if it is not still. Fig. 2(b) shows an evader could choose one available action from *left, up, right and still* when it is in the cell 4 confronting the boundary cell 7.

B. Formation Strategies of Pursuers

Three formation strategies are designed for the pursuers in latter experiments. In *Formation1*, four pursuers form a square shape, in which each of the pursuers occupy one corner. During chasing, each of the four pursuers keep fixed position relative to other pursuers. In *Formation2*, each of the pursuers chase the evader by itself without communicating with other pursuers. *Formation3* is shown in Fig. 3 with its three typical states. In *S1* (left), the pursuers separate into two equal parts, the pursuers which are closer to the evader keep on chasing, while the other part firstly moves to the center of the environment. Then in *S2* (middle), the other part reaches the center and in *S3* (right) two parts chase the evader together. *Formation3* could make the surrounding scope much bigger than *Formation1* and *Formation2*.

Fig. 3. Three typical states of *Formation3*. Note that for illustration, the white cross star and white square denote the evader and the pursuer respectively. During training, each of them is one square pixel. Best viewed in color.

C. Game Scenarios

Three different and representative game settings $Gi, i = 1, 2, 3$ are carefully designed, considering the difficulty and generalization. The settings vary in the number of the pursuers, the formation strategy of the pursuers, the initial position of the pursuers and the evader, which are reported both in Fig. 4 and Table I.

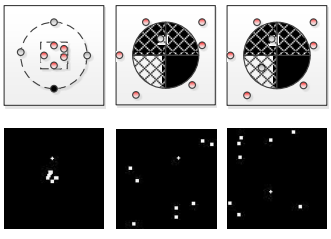

Fig. 4. Top row from left to right each represents *G1*, *G2*, *G3* respectively, in which the red point is pursuer and the gray point is evader. Bottom row are the real initial game states correspondingly. Best viewed in color.

In Fig. 4(a), the gray dots are the initial positions of the evader for training, while the black dots for testing. In Fig. 4(b) and 4(c), the grid map areas are the initial positions of the pursuers for training and black areas for testing. During testing period, our way is similar to [12], whose agent was ran from each of 100 starting points which were sampled from a

TABLE I
THREE DIFFERENT GAME SETTINGS FOR PURSUIT-EVASION GAMES.

	Initial position of evader	Initial position of pursuers	Number of pursuers	Formation strategy of pursuers
G1	random one from four of a circle	distributed in the center square	4-8	Formation1
G2	average positions of pursuers	distributed in the outside of circle	4-8	Formation2
G3	average positions of pursuers	distributed in the outside of circle	4-8	Formation3

human professional's game play. In latter experiments, these testing positions and areas are the held-out states for testing how well the evader agent generalizes.

IV. EXPERIMENTS

Experiments are implemented using Torch7 on a PC with an Intel Xeon(R) CPU E5-2620 (2.4 GHz), 47 GB RAM, single Nvidia GTX 1080 GPU (8GB). Each of the evader agents is trained in three different games respectively for 5 million steps about two days. Each agent is trained for 5 runs in the corresponding game. The behavior policy during training is ϵ-$greedy$ policy with ϵ annealed linearly from 1.0 to 0.1 over the first one million training steps, and fixed at 0.1 thereafter. In all experiments, we use the RMSProp algorithm [11] with minibatch size 32 and a replay memory of 1 million most recent frames. D is set to 4 pixels. Three *Learned agent i, i=1,...,3* for the final evaluation are obtained after the training is finished. Code and models will be available at *https://github.com/zhujiagang/pursuit-evasion-code*.

A. Training and Stability

During training, each of agents is periodically evaluated over the held-out set of states every 50,000 steps for 50,000 validation steps by running an ϵ-$greedy$ policy with ϵ=0.05. We use three evaluating metrics, which are the average of the action-value Q, the average total reward per episode and the average number of steps per episode respectively. The average of the action-value Q is the average over the maximum Q given the legal actions in validation states, $\frac{1}{n}\sum_{i=1}^{n}\max_{a} Q^*(s_i, a; \theta)$, where n is the number of validation steps, 50,000.

As shown in the leftmost curves of each subfigure of Fig. 5, the average of the action-value Q fluctuates at the beginning of training, even being negative. The reason is the insufficient training and *null op starts* [12] in which a random number of frames were skipped by repeatedly taking the null or do nothing action before giving control to the agent in order to ensure the variation in the initial conditions. The action-value Q provides an estimation of how much discounted reward the agent can obtain by following its policy from any given states. In the two rightmost curves of each subfigure, the average total reward per episode and the average number of steps per episode are quite noisy. It is due to the small changes

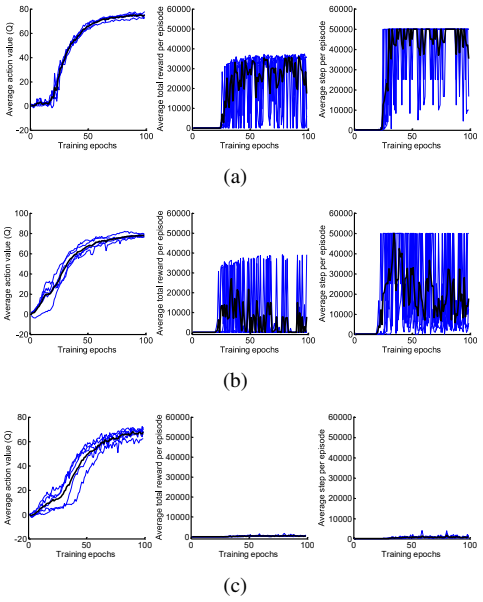

Fig. 5. Training curves of *G1*, *G2*, *G3*. The plots (black) are averaged over five runs (blue) for different seeds with fixed hyperparameters.

of weights of Q-network which defines behavior policy can lead to large changes in the distribution of states which the evader agent can visit.

B. Value Function Analysis

Four typical testing frames are extracted and shown in Fig. 6, while the corresponding values of the learned action-value function are listed in Table II.

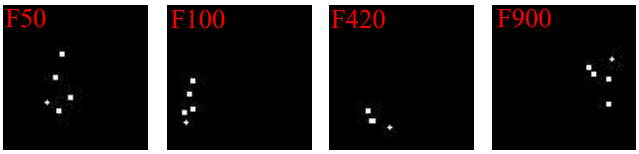

Fig. 6. Four testing frames in *G3*. From left to right, they are the 50th, 100th ,420th , 900th frame respectively in one testing experiment. Best viewed in color.

TABLE II
ACTION VALUE OF FIVE ACTIONS OF FOUR DIFFERENT STATES. NOTE THAT EACH OF HIGHEST VALUES (BOLDFACED) IN FOUR ROWS CORRESPONDS TO MOST APPROPRIATE ACTION THAT THE EVADER SHOULD TAKE IN NEXT STEP.

Frame	still	right	up	left	down
F50	54.63	53.02	55.11	**56.56**	53.95
F100	64.84	66.92	61.56	62.25	**67.05**
F420	81.61	**82.23**	81.29	80.92	81.87
F900	66.18	66.36	**67.04**	66.95	66.08

As shown in Fig. 6, the most promising action choice in *F50* for the evader is moving left, next choice is moving up. The action value of the left in Table II is the biggest, then follows action up, right is the smallest among the five actions.

As shown in Table II, all the action values of *F420* are larger than the other three states'. DQN predicts high state value when the evader is far away from the pursuers because it has learned that the evader will not be captured in a short time.

C. Comparison Among Agents

A random agent and a repulsive agent are devised for comparison in this subsection. The random agent chooses action uniformly from the 5 legal actions. While the repulsive agent follows the artificial potential field method, it selects actions according to $a_r = \arg\min_a |\theta_F - \theta_a|$, when $a = 1, 2, 3, 4$, $\theta_a = 0, \pi/2, \pi, 3\pi/2$ respectively. The three learned agents are compared with these two agents in two metrics, *surrounded steps* and *captured times*. *Surrounded steps* is the total step number when (4) is true in whole testing steps. *Captured times* is the total step number when (5) is true in whole testing steps. The testing for each agent in each game is run over the held-out set of states for 100,000 steps. The results are averaged over five runs.

The results are shown in Table III and IV respectively. All the agents tend to have increasing larger surrounded steps from *G1* to *G3* because of the increasing difficulties. *Learned agent 1, 2* have worse performance in all three games than the repulsive agent. The reason is that *Learned agent 1, 2* are trained in *G1*, *G2* respectively. They are games with the simple formation strategy. *Learned agent 3* which is trained in games with *Formation3* has better performance than the repulsive agent in all three games. Especially in *G3*, the best learned agent outperforms the repulsive agent by 18.342 in *surrounded steps* and 0.638 in *captured times* respectively, showing the advantage of combining the artificial potential field method with deep reinforcement learning. Because of more complex and representative samples for training, *Learned agent 3* becomes the best agent: it has the least surrounded steps and captured times in all games, which demonstrates that the learned agent could generalize well across different games. Experiments show that the learned agent for the evader can be more effective and robust than the repulsive agent method in different games.

TABLE III
COMPARISON OF THREE LEARNED AGENTS WITH A RANDOM AGENT AND A REPULSIVE AGENT IN *surrounded steps* (%) IN THREE GAMES IN 100,000 TESTING STEPS. THE SMALLEST VALUES IN EACH COLUMN ARE BOLDFACED.

Agent	G1	G2	G3
Random agent	0.030	99.755	99.804
Repulsive agent	**0**	0.077	20.287
Learned agent 1	**0**	5.376	23.090
Learned agent 2	**0**	0.131	51.974
Learned agent 3	**0**	**0.028**	**1.945**

D. Reward Function Analysis

To illustrate the importance of designing reward function, a different function without considering the evader being surrounded by the pursuers is also designed. That is

$$r_t = \begin{cases} 1 & if\ dist_{min} > D\ and\ thr_t < thr_{t-1} \\ -1 & if\ dist_{min} < D/2 \\ 0 & otherwise \end{cases} \quad (7)$$

The three agents with reward function (7) are trained from scratch and tested exactly the same as the former three learned agents with reward function (3). Each of agents is trained and tested in *G1*, *G2*, *G3* respectively. As shown in Fig. 7, each of the former agents *Learned agent i, i=1,...,3* considering the surrounded case in reward function has less *surrounded steps* and *captured times* than those not considering the surrounded case. Incorporating the artificial potential field into reward function helps the learned agent escape from being surrounded, thus suffering a lower risk of being captured.

TABLE IV
COMPARISON OF THREE LEARNED AGENT WITH A RANDOM AGENT AND A REPULSIVE AGENT IN *captured times* (%) IN THREE GAMES IN 100,000 TESTING STEPS. THE SMALLEST VALUES IN EACH COLUMN ARE BOLDFACED.

Agent	G1	G2	G3
Random agent	9.612	6.142	6.086
Repulsive agent	0.001	0.003	0.685
Learned agent 1	0.001	0.264	1.044
Learned agent 2	0.061	0.004	2.172
Learned agent 3	**0**	**0**	**0.047**

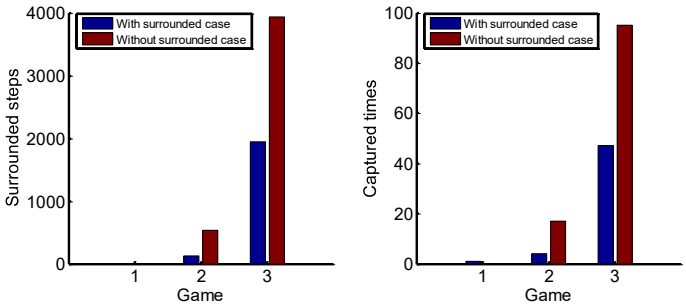

Fig. 7. Comparison between three former learned agents (dark blue) and three learned agents without considering the surrounded case (dark red) in reward function in *surrounded steps* and *captured times*. Each of agents is trained and tested in *G1*, *G2*, *G3* respectively. Best viewed in color.

E. Network Visualization

We use a visualization technique, t-SNE [13] to verify the representations learned by the DQN in the last hidden layer to game states experienced after running in *G3* for 5,000 steps, which is shown in Fig. 8. Note that this agent was trained with reward function (3). The two-dimensional points are colored according to the state value V, which is the maximum expected reward of a state, $\max_a Q^*(s, a; \theta)$ predicted by DQN. The state value ranges from the dark red (highest V) to the dark blue (lowest V). The red point has a larger state value than the yellow, green and blue points. It

means that the state of red point has less threats posed by the pursuers and boundaries. The nine frames corresponding to nine points which represent three different state value ranges, low (top left), middle (left), high (bottom) are selected and the numbers beside those frames are their index in 5,000 frames respectively. DQN predicts high state value when the evader is far away from the pursuers (bottom, *F3533*, *F4199*, *F3522*) because it has learned that the evader will not be captured in a short time. Top left frames *F303*, *F287*, *F77* are assigned lower state value because the evader is surrounded by the pursuers. The frames *F1146*, *F2331*, *F955* have moderate state value because the pursuers are taking separate formation but the evader still gets a possible road to escape.

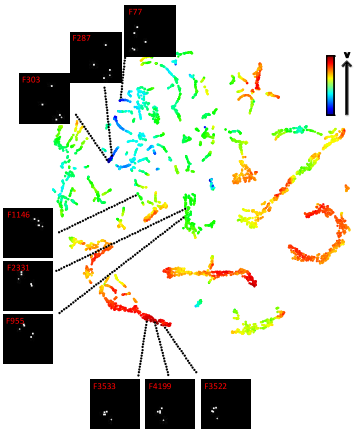

Fig. 8. Two-dimensional t-SNE embedding of the representations in the last hidden layer assigned by DQN to game states experienced after running in *G3* for 5,000 steps. Best viewed in color.

V. Conclusion

In this work we introduce an approach for learning the evasion strategy for the evader in pursuit-evasion against the pursuers with Deep Q-network. A reward function considering both the evader escaping from being surrounded by the pursuers and keeping distance from the pursuers is handcrafted to give the immediate reward to the agent to improve the evasion strategy step by step. In the three designed games, our learned agents achieve better performance in *surrounded steps* and *captured times* than a random agent and a repulsive agent, showing the advantage of combining the artificial potential field method with deep reinforcement learning. The future directions would include extending this work into multi-agent setting with multiple evaders and pursuers.

Acknowledgment

This work is supported in part by the National High Technology Research and Development Program of China under Grant No.2015AA042307, the National Natural Science Foundation of China under Grant No.61773374, and in part by Project of Development In Tianjin for Scientific Research Institutes Supported By Tianjin government under Grant No.16PTYJGX00050.

References

[1] S. Rodriguez et al, Toward realistic pursuit-evasion using a roadmap-based approach. In *2011 IEEE International Conference on Robotics and Automation*, pp. 1738–1745, 2011.

[2] L. Shen Hin, T. Furukawa, G. Dissanayake and H. F. Durrant-Whyte, A time-optimal control strategy for pursuit-evasion games problems. In *2004 IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3962–3967, 2004.

[3] B. Even, P. Alexey, G. Reza, Formation control of underactuated marine vehicles with communication constraints. *Control of Marine*, vol. 50, no. 3, pp. 455–461,2006.

[4] M. Ji and M. Egerstedt, Distributed Formation Control While Preserving Connectedness. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pp. 5962–5967, 2006.

[5] L. Barnes, M. Fields and K. Valavanis, Unmanned ground vehicle swarm formation control using potential fields. In *2007 Mediterranean Conference on Control and Automation*, pp. 1–8, 2007.

[6] H. Daniel et al, Muti-Robot control system for pursuit-evasion problem. *Journal of electrical engineering*, vol. 60, no. 3, pp. 143–148, 2009.

[7] H. Timothy , H. Geoffrey, V. Isler, Search and pursuit-evasion in mobile robotics. *Autonomous Robots*, vol. 31, no. 10, pp. 299–316, 2011.

[8] V. Mnih et al, Human-level control through deep reinforcement learning. *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[9] V. Mnih et al, Playing atari with deep reinforcement learning. *arXiv preprint*, arXiv:1312.5602, 2013.

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

[11] T. Tieleman, G. Hinton, Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4, 2012.

[12] A. Nair et al, Massively Parallel Methods for Deep Reinforcement Learning. *arXiv preprint*, arXiv:1507.04296, 2015.

[13] L. V. D. Maaten, G. Hinton, Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, vol. 9, no. 2, pp. 2579-C2605, 2008.

[14] M. Egorov, Multi-Agent Deep Reinforcement Learning. *http://cs231n.stanford.edu/reports/2016/pdfs/122_Report.pdf*. 2016.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[16] M. Lai, Giraffe: Using Deep Reinforcement Learning to Play Chess, *arXiv preprint*, arXiv:1509.01549, 2015.

[17] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, Prioritized Experience Replay, *arXiv preprint*, arXiv:1511.05952, 2015.

[18] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke, Towards Vision-Based Deep Reinforcement Learning for Robotic Motion Control, *arXiv preprint*, arXiv:1511.03791, 2015.

[19] J. Oh, X. Guo, H. Lee, R. Lewis, and S. Singh, Action-Conditional Video Prediction using Deep Networks in Atari Games, *arXiv preprint*, arXiv:1507.08750, 2015.

[20] V. Mnih et al, Asynchronous Methods for Deep Reinforcement Learning, *arXiv preprint*, arXiv:1602.01783, 2016.

[21] H. van Hasselt, A. Guez, and D. Silver, Deep reinforcement learning with double q-learning, *arXiv preprint*, arXiv:1509.06461, 2015.

[22] E. Camci, E. Kayacan. Game of Drones: UAV Pursuit-Evasion Game With Type-2 Fuzzy Logic Controllers Tuned by Reinforcement Learning. *IEEE International Conference on Fuzzy Systems*, pp. 618–625, 2016.

[23] M. D. Awheda, H. M. Schwartz. A Fuzzy Reinforcement Learning Algorithm Using a Predictor for Pursuit-Evasion Games. *Annual IEEE Systems Conference*, pp. 1–8, 2016.

[24] J. K. Gupta, M. Egorov, M. Kochenderfer. Cooperative Multi-agent Control Using Deep Reinforcement Learning. *Autonomous Agents and Multiagent Systems*, pp. 66–83, 2017.

[25] C. Undeger, F. Polat. Multi-agent real-time pursuit. *Autonomous Agents and Multi-Agent Systems*, vol. 21, no. 1, pp. 69–107, 2010.

[26] M. D. Awheda, H. M. Schwartz. A Decentralized Fuzzy Learning Algorithm for Pursuit-Evasion Differential Games with Superior Evaders. *Journal of Intelligent and Robotic Systems*, vol. 83, no. 1, pp. 35–53, 2016.

[27] H. Raslan, H. Schwartz, S. Givigi. A Learning Invader for the Guarding a Territory Game. *Journal of Intelligent and Robotic Systems*, vol. 83, no. 1, pp. 55–70, 2016.

[28] D. Silver et al, Mastering the game of Go without human knowledge. *Nature*, vol. 550, pp. 354–359, 2017.