Multi-Virtual-Agent Reinforcement Learning for a Stochastic Predator-Prey Grid Environment

Yanbin Lin, Zhen Ni and Xiangnan Zhong
Electrical Engineering and Computer Science
Florida Atlantic University
Boca Raton, FL, USA, 33431
{liny2020, zhenni, xzhong}@fau.edu

2022 International Joint Conference on Neural Networks (IJCNN) | 978-1-7281-8671-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/IJCNN55064.2022.9891898

***Abstract*—Generalization problem of reinforcement learning is crucial especially for dynamic environments. Conventional reinforcement learning methods solve the problems with some ideal assumptions and are difficult to be applied in dynamic environments directly. In this paper, we propose a new multi-virtual-agent reinforcement learning (MVARL) approach for a predator-prey grid game. The designed method can find the optimal solution even when the predator moves. Specifically, we design virtual agents to interact with simulated changing environments in parallel instead of using actual agents. Moreover, a global agent learns information from these virtual agents and interacts with the actual environment at the same time. This method can not only effectively improve the generalization performance of reinforcement learning in dynamic environments, but also reduce the overall computational cost. Two simulation studies are considered in this paper to validate the effectiveness of the designed method. We also compare the results with the conventional reinforcement learning methods. The results indicate that our proposed method can improve the robustness of reinforcement learning method and contribute to the generalization to certain extent.**

***Index Terms*—Reinforcement learning, multiple virtual agents, generalization problem, dynamic environment, and parallel learning.**

I. Introduction

In the last decades, reinforcement learning (RL) has attracted increasing interests in the field of machine learning. Unlike supervised learning and unsupervised learning, reinforcement learning method is aimed at maximizing the sum of the rewards an agent can receive in a long run [1]. Reinforcement learning has proven its value in a series of artificial domains. However, many of the research advances in RL are often hard to be applied in dynamic systems due to a series of assumptions that are rarely satisfied in practice [2].

One of the largest challenges for applying reinforcement learning methods in dynamic environments is the generalization problem. Most researches analyzed in specific environments with ideal assumptions. Tasks may be partially observable without considering abrupt changes or stochastic variations in a dynamic environment. For example, both Pacman game [3] and Hunter & Prey game [4] are based on the agents that can sense the position of enemies, while some enemies are unpredictable or insensible in the reality. Real-world tasks can be more complex and diversified. For renewable energy, high variability of solar production, especially the ramp event, poses serious challenges to traditional power system [5]. The ability of agents to predict the renewable energy accurately maintains the reliability and stability of the power system, especially under extreme weather conditions.

Recently, some approaches have been developed to improve the robustness of reinforcement learning. Iyengar proposed a robust formulation for discrete time dynamic programming (DP) [6]. In [7], the authors involved an explicitly training agent on various perturbations with an average learning error and used a training policy that can determine online environments [8]. Beyond that, several works [9], [10], [11] analyzed the dynamics of policies learned by multiple self-interested independent learning agents using its deep Q-network. Although combining with deep learning can somehow settle the generalization problem of RL, deep neural network needs more memory and computations [12]. Besides, as [13] illustrated, deterministic optimization approaches may not ensure the optimal point in most real-life problems containing uncertain parameters [14]. Therefore, stochastic optimization is prescribed involving uncertainties and probabilities to deal with this problem.

RL has been successfully used in many fields and the predator-prey model is one of the most popular models in this field. J.Schrum compared the performances of several algorithms applied in a simple predator-prey grid world in terms of how well the predator and the prey compete with each other [15]. But it analyzed this predator-prey problem by using partial state representations with full awareness of predators' locations in all directions. Another related predator-prey model used reinforcement learning algorithms to improve the existing individual-based ecosystem [16]. For our experiments, we concentrate on how to improve the prey's ability to escape when the predator occurs randomly with some certain probabilities. This means that the knowledge learned by the prey can be adapted to a changeable environment.

In this paper, we work on the generalization problem of reinforcement learning methods in dynamic predator-prey grid environments. Specifically, we propose a new algorithm called multi-virtual-agent reinforcement learning (MVARL) method. It is aimed at associating parallel learning with probability in changing environments. The contributions of this paper are provided as follows. First, our algorithm requires less computational cost. Different from traditional multi-agent systems

which require instantaneous communication information for collaboration, our virtual agents can accelerate the learning process by sharing information at periodic points during the learning process. Second, instead of using multiple agents in the same environment, we use virtual agents to avoid non-stationary of sub-environments. In existing works of multi-agent reinforcement learning (MARL), agents are solving different parts of a task or working in an environment that is altered by the actions of other agents [17] [18]. The non-stationary of the environment is not generated by an arbitrary stochastic process, but rather by other agents. Third, our algorithm offers up the generalization problem of RL in dynamic environments. Experiments show that our multi-virtual-agent reinforcement learning algorithm has better performance than other tabular reinforcement learning methods.

The remainder of this paper is organized as follows. Explanations about comparative methods of reinforcement learning including four kinds of traditional methods are given in Section II. Section III states the main idea and structure of our multi-virtual-agent reinforcement learning algorithm. Specific experiments and results of two cases are shown in Section IV. Finally, conclusions and the future research's possible applicable guidance are provided in Section V.

II. Background and Methods

A Markov decision process (MDP) can be expressed by a tuple: $\{S, A, P, R, \gamma, s_0\}$, where S is a set of states s, A is a set of possible actions a, P is the transition probability which can be represented as $P : S \times A \times S \to \mathbb{R}$, R is the reward function $R(s, a)$, γ is the discounted rate and s_0 is the initial state where the agent will depart from. Mapping from current state and action pair (s, a) to next state s' is probabilistic determined by $P(s, a, s')$. As we mentioned before, the goal for a reinforcement learning agent is to maximize the cumulative reward. The total reward is realized giving: $G = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)$, where γ should be subjected to $0 < \gamma < 1$. In our experiments, the discounted rate $\gamma = 0.9$. Besides, ε-greedy method is applied in order to keep trade-off between the exploration and the exploitation.

A. Q-learning

One of the popular reinforcement learning algorithms is Q-learning [19], which is an action selection method by using state/action value function $Q(s, a)$ and state value function $V(s)$ to learn the decision policy π. Bellman's optimal principle suggests that an optimal policy can be built for the tail sub-problem [20].

$$\begin{aligned} Q(s, a) &= R(s, a) + \gamma \sum_{s'} P(s, a, s') V(s') \\ V(s) &= max_a(Q(s, a)) \end{aligned} \tag{1}$$

Therefore, agent can update its Q-value corresponding to this formula: $Q(s, a) = R(s, a) + \gamma max_{a'} Q(s', a')$. Once we introduce the learning rate α to determine Q-values' updating speed, the formula will become

$$Q(s, a) = (1-\alpha)Q(s, a)+\alpha(R(s, a)+\gamma max_{a'} Q(s', a')) \tag{2}$$

B. Double Q-learning

Although the formulation of Q-learning works well in majorities of cases, a Q-learning agent has a potential to put itself at risk when an exploratory move compels it to receive a low reward. To solve this problem, Hasselt applied a double estimator to Q-learning to construct double Q-learning method [21].

Double Q-learning method performs well in some settings in which Q-learning performs poorly due to its overestimation. And it's proved that double Q-learning is not less data-efficient than Q-learning. Double Q-learning stores two Q-functions (Q_A and Q_B) updated with these formulas as below

$$\begin{aligned} &a^* = argmax_a Q^A(s', a) \\ &Q^A(s, a) \leftarrow Q^A(s, a) + \alpha(R(s, a) + \gamma Q^B(s', a^*) - Q^A(s, a)) \\ &b^* = argmax_a Q^B(s', a) \\ &Q^B(s, a) \leftarrow Q^B(s, a) + \alpha(R(s, a) + \gamma Q^A(s', b^*) - Q^B(s, a)) \end{aligned} \tag{3}$$

where a^*, b^* are the optimal actions of Q^A and Q^B respectively.

Double Q-learning is likely to eliminate the limitation caused by maximization bias [1]. Although double Q-learning avoids the flaw of the overestimation bias that Q-learning does, it might underestimate the action values at times. Double Q-learning is suitable for the cases with random value reward, like the example shown in Sutton's textbook.

C. Q-learning with Experienced Replay

Pieters and Wiering proposed several adaptions of Q-learning for a dynamic environment called Q-learning with experienced replay method, where experience tuples are reused based on time or based on the obtained reward [22]. The main idea of this method is to accumulate tuple $E = \{s, a, r, s'\}$ to construct the database M. After some trials Z, the agent needs to randomly choose K tuples of E from M to update Q-function as the equation below. This process is represented by L function as below

$$\begin{aligned} M &\leftarrow update(M, E, N) \\ Q &\leftarrow L(Q, M, K, \gamma, \alpha) \end{aligned} \tag{4}$$

where K is the number of tuples and N is number of experiences in the database M.

There are two kinds of methods to update database M in order to solve the capacity problem. One is to update based on time, the other is to update based on reward. In the comparison experiments, we use the updating rule based on time. If size(M) $< N$, database M will be merged with E. If size(M) $> N$, we will remove the earliest element from itself and then merge it with E.

D. Monte Carlo Method

Unlike dynamic programming, the Monte Carlo (MC) method does not need a model of the environment. Hence, it's possible to achieve optimal behavior without any prior knowledge. Its goal is to learn $v_\pi(s)$ given some numbers of episodes under π. Monte Carlo method samples and averages

returns observed for each state-action pair after visiting to s [23]. Policy improvement is done by making the policy greedy with respect to the current value function.

In our experiment, we use Monte Carlo method by using ε-soft policy and the probability of the greedy action is determined by (5)

$$P(s,a) = 1 - \varepsilon + \frac{\varepsilon}{\mathcal{A}(s)} \tag{5}$$

where $\mathcal{A}(s)$ is the sum of all actions' values.

III. Multi-Virtual-Agent Reinforcement Learning

Our algorithm is developed in a parallel learning framework, where several virtual agents are learning simultaneously in different virtual sub-environments. Meanwhile, a global agent interacts with the real changing environment. Each virtual agent i has a Q-table Q^i and action set A^i to control its optimal policy. The virtual environments compose the global environment with probability. Namely, if the probability of virtual environment i is p^i, the global environment will be determined by this relation: Global Environment $\Rightarrow \sum_{i=1}^{n} p^i \times$ Virtual Environment^i. The sum of all p^i should be equal to 1. The framework of multi-virtual-agent reinforcement learning is shown as Fig. 1 with three main modules. Multiple virtual agents interact with virtual environments in module I and module II. Then each virtual agents integrate with global agent in module III with the experience they learned. After this iteration, global agent interacts with global environment in module III, then sends the updated parameters back to those virtual agents in module II.

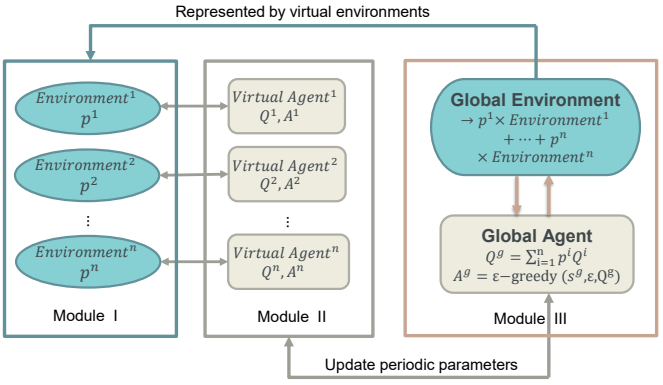


Fig. 1. The structure of MVARL algorithm. Module I and module II stand for the multiple virtual environments and virtual agents respectively. Module III is the actual operation module which includes the global agent and the global environment. Global environment is composed of several virtual environments with probabilities which totally sum up to 1. Global agent updates its Q-values according to those virtual agents' Q-values with probabilities and take its action using ε-greedy algorithm. In the meanwhile, global agent will send the updated information to virtual agents in module II after its iteration.

For our algorithm, the update of i^{th} virtual agent's Q-table Q^i in each episode is defined by Equation (6).

$$\begin{aligned} Q^i(s^i,a^i) = (1-\alpha)Q^i(s^i,a^i) + \\ \alpha(R^i(s^i,a^i) + \gamma max_{a'} Q^i(s',a')) \end{aligned} \tag{6}$$

where $s^i, a^i, Q^i(s^i,a^i)$ and $R^i(s^i,a^i)$ represent the i^{th} virtual agent's state, action, Q-value and reward of pair (s^i,a^i). s' is the next state and a' is the next action of i^{th} virtual agent.

Q^g is the Q-table of global agent which is integrated with virtual agents' Q-table after each episode. In regard to the update algorithm of global agent's Q-table, there are two kinds of approaches to integrate information. One is to update with statistical probability expressed as Equation (7), the other is to update averagely expressed as Equation (8).

$$Q^g(s,a) = \sum_{i=1}^{n} p^i Q^i(s,a) | \sum_{i=1}^{n} p^i = 1 \tag{7}$$

$$Q^g(s,a) = \sum_{i=1}^{n} \frac{1}{n} Q^i(s,a) \tag{8}$$

After the iteration of global agent, the virtual agents' Q-values are also updated according to the global agent.

$$Q^i(s,a) = Q^g(s,a), \ \forall \ i = 1, \cdots, n \tag{9}$$

IV. Experiments

A. Task Description

The task used in our work is a **predator-prey game**, which is shown in Fig. 2. It's a 6×9 grid world with several walls. The biggest difference from traditional grid world is that a predator and a prey are introduced to the environment and the position of the predator is changing according to specific probability.

A prey will start each episode at the S point and a predator will appear at Area 1, Area 2, and Area 3 or will not appear with a statistical probability P, where $P = [p^1, p^2, p^3, p^4] \ | \ \sum_{i=1}^{4} p^i = 1$. p^i means the probability that a predator will appear at Area i $(\forall i = 1, 2, 3)$, and p^4 means the probability that a predator will not appear.

The predator in each area has two possible positions and each time a predator will only appear at one of them randomly. Only the prey can occupy the goal space G and the prey has no access to wall spaces. If the goal space is occupied by a prey, it means the prey successfully escapes and the terminal state is a **win state** with a positive reward 100. Otherwise, if a predator eats a prey by occupying the same space as the prey or the prey can't escape within required steps, it means the prey loses the game and the terminal state is a **lose state** with a negative reward -100. The rewards of other steps are 0. The maximum step (Max-Step) for a prey to escape is set as 600. The goal for a prey is to start from point S, avoiding walls and the predator in the way of finding an optimal way to reach the point G and escape. It's apparent that Area 1, Area 2, and Area 3 are very crucial for a prey to escape from this grid world.

B. Evaluation Indicators

To evaluate the performance of algorithm, we introduce two indices: **win probability** (probability of the win state) and **step**. Win probability represents the probability of the prey to escape successfully which is defined by Equation (10).

Oppositely, **lose probability** means the probability that the prey will lose the game. Step stands for how many steps the prey needs to take in one episode which is defined by Equation (11). The step-to-goal means the actual steps the prey needs to take in the win states. It's worth mentioning that if the terminal state is a lose state, the step will be set as the Max-Step which equals to 600. Accordingly, we define an episode as the prey ends at a win state or a lose state, which means the prey escape successfully or unsuccessfully.

$$\text{Average win probability} = \frac{\sum_{j_1=1}^{N_{trials}} \sum_{j_2=1}^{N_{episodes}} \text{flag}}{\sum_{j_1=1}^{N_{trials}} \sum_{j_2=1}^{N_{episodes}} 1} \tag{10}$$

$$\text{flag} = \begin{cases} 1, & \text{if the prey successes} \\ 0, & \text{if the prey fails} \end{cases}$$

$$\text{Average step per episode} = \frac{\sum_{j_1=1}^{N_{trials}} \text{Step}}{N_{trials}} \tag{11}$$

$$\text{Step} = \begin{cases} \text{step-to-goal}, & \text{if the prey successes} \\ \text{Max-Step}, & \text{if the prey fails} \end{cases}$$

where N_{trials} is the number of trials, $N_{episodes}$ is the number of episodes, and j_1, j_2 are the indexes of trials and episodes separately.

C. *Case 1*

According to our algorithm, we initially choose $\alpha = 1, \gamma = 0.9$, $N_{trials} = 50$, and $N_{episodes} = 5000$. In order to eliminate the effect of random results, we implement 50 trials and average the results.

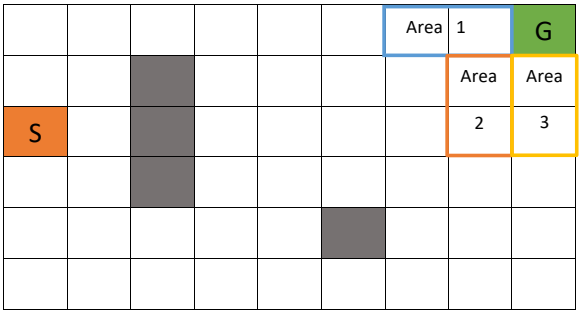
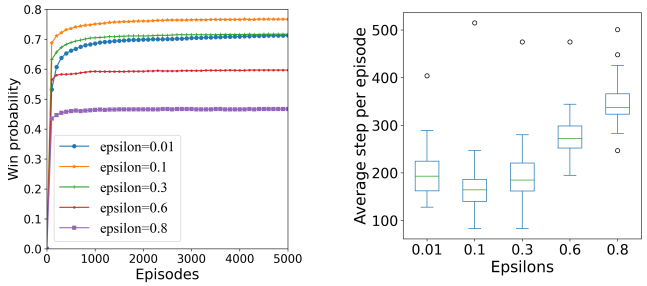
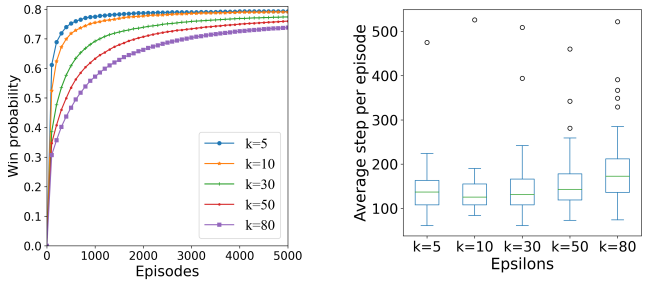


Fig. 2. Case 1: predator-prey game with size of 6×9. Area 1, Area 2, and Area 3 stand for the areas that a predator is possible to occur. S is the start space and G is the goal space. The grey ones are wall spaces.

Case 1 is executed in a 6×9 predator-prey game with a predator appearing possibility $P = [0.4, 0.3, 0.2, 0.1]$. The environment of case 1 is shown in Fig.2. In order to choose optimal parameter of ε-greedy, we have two kinds of experiments. One is to choose ε fixed, the other is to choose ε variable according to the number of episodes which means $\varepsilon = \frac{\varepsilon}{\tau}$ where τ is the declining rate. For the first choice, our experiment chooses $\varepsilon = 0.01/0.1/0.3/0.6/0.8$. For the second choice, our experiment chooses initial $\varepsilon = 1, \tau = 1.1, \varepsilon$ declines every k episodes, where k varies from 5 to 80.



(a) Average win probability of fixed epsilons. (b) Average step of fixed epsilons.



(c) Average win probability of variable epsilons. (d) Average step of variable epsilons.

Fig. 3. Average win probability and step of case P=[0.4,0.3,0.2,0.1]. (a) and (b) are fixed epsilons, where $\varepsilon = 0.01/0.1/0.3/0.6/0.8$. (c) and (d) are variable epsilons and we chose $\varepsilon_0 = 1$, $\tau = 1.1, \varepsilon = \frac{\varepsilon}{\tau}$, ε declines every k episodes as variable epsilons, where k varies from 5 to 80.

According to our algorithm structure, we use four virtual agents to simulate four kinds of sub-environments separately. Each virtual environment i has p^i probability to occur in the global environment. For each episode, virtual agents will end at the win state or the lose state. Then virtual agents share information to the global agent. The global agent integrates these information with probabilities and then sends information back to virtual agents.

As the Fig. 3(a) and Fig. 3(c) demonstrated, variable ε has a better win probability than fixed ones. Especially, when $\varepsilon = \frac{\varepsilon}{1.1}, k = 5$, the result of average win probability has an optimal performance nearly reach 80 % and the result of average step is also the smallest one. Although when $\varepsilon = \frac{\varepsilon}{1.1}, k = 10$, the average step is the most concentrated according to Fig.3(b) and Fig.3(d), we choose $\varepsilon = \frac{\varepsilon}{1.1}, k = 5$ as parameters of ε for overall consideration.

Reasons for optimal win probability can't reach 100% is because Area1, Area2, and Area3 are critical and difficult spaces for a prey to overcome. They are important spaces for a prey to go to the goal space to escape. So, it's hard for a prey to avoid all situations of the predator because the predator changes with probability. The best solution for a prey is to choose minimum probability way to the lose state, which is equal to 20% in the case of $P = [0.4, 0.3, 0.2, 0.1]$. It's obvious proved by the heatmap of global agent's Q-table, which is shown in Fig. 4. The Q-table has 6 rows and 9 cols with 54

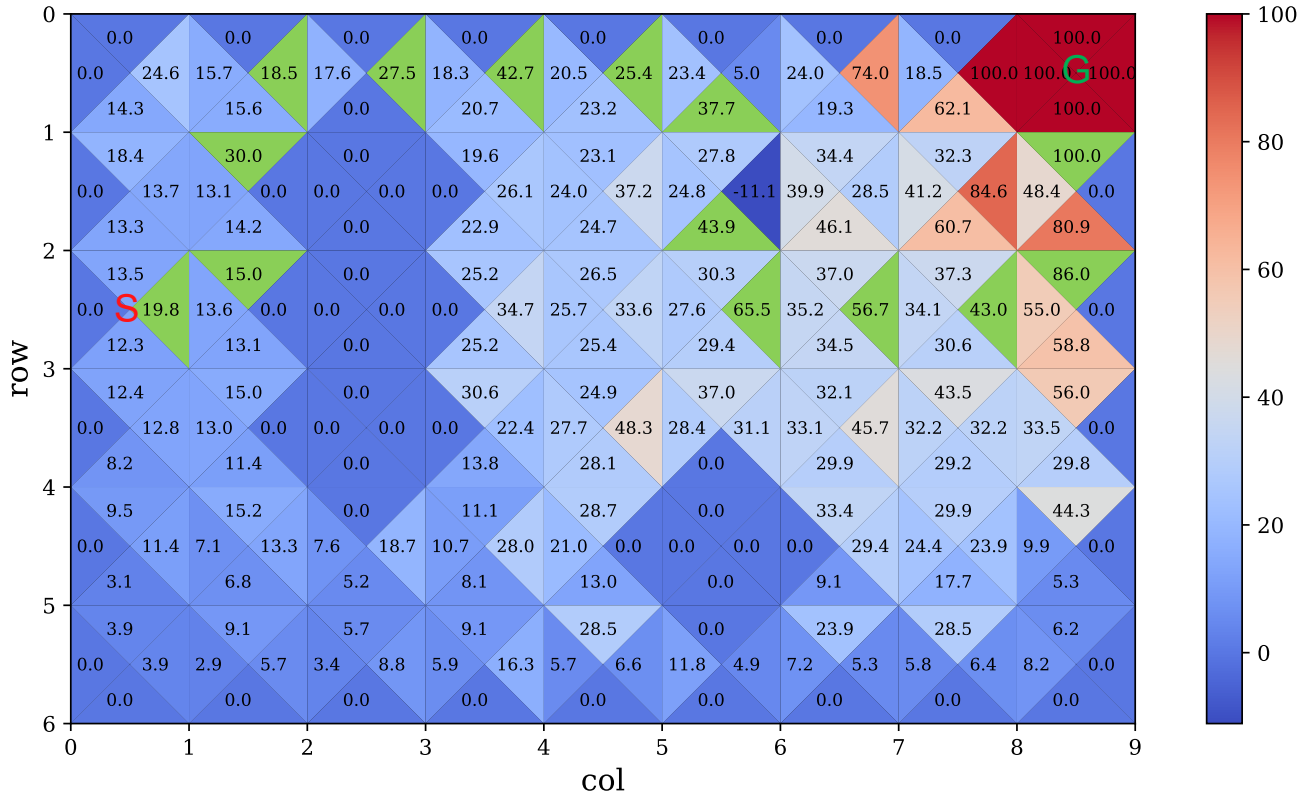


Fig. 4. Case 1: heatmap of Q table when $P = [0.4, 0.3, 0.2, 0.1]$. This heatmap represents Q-table of case 1: a 6×9 predator-prey game. The Q-table has $6 \times 9 = 54$ spaces and each space has four sub-spaces. Every four sub-spaces stand for four actions' Q-values in this space, which are up, down, left, and right. Global agent will choose the action with the biggest Q-value (highlighted in green color) as its actual action from four directions in each space. Therefore, it will find a way to go through Area3 which has the minimum probability to lose the game.

spaces. Each space is separated into four sub-spaces standing for four actions' Q-values. Global agent will choose the action with the biggest Q-value (highlighted in green color) as its actual action from four directions (up, down, left, and right) in each space.

The best route for global agent in case 1 is to find the way with the minimum lose probability, which is the way to go through Area3 with a lose probability of 20%. That explains why the best performance of our algorithm in the case is 80%. Likewise, when the probability is changed to $P = [0.1, 0.3, 0.4, 0.2]$, this conclusion can also be applied. The best route in this case is to go through Area1 with a win probability 90%. Cases with different probabilities satisfy this rule as well.

We also change the distribution of P to see the generalization performance of our two kinds of updating algorithms. By using the Equation (8), the update of global Q-table uses average probability of sub-environments, while the actual environment's distribution is various. The results illustrate that using average probability to update global agent's policy can achieve performance as good as the Equation (7) which updates Q-table with statistical probability. The results of two updating algorithms with different P are shown in Fig.5.

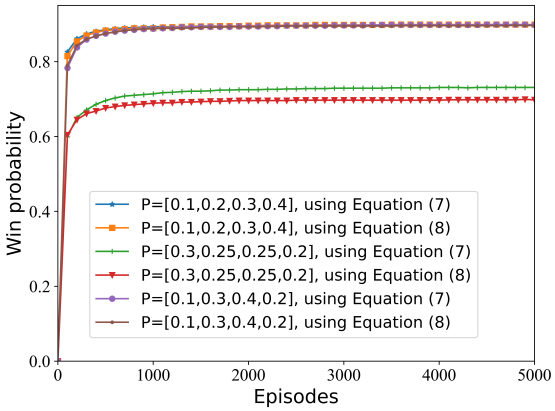


Fig. 5. Case 1: average win probability of different probabilities P with two updating algorithms according to Equation (7) and Equation (8). Two kinds of updating algorithms can have almost similar results.

Comparison Experiments. We used random walk as our blank experiment. This means the prey acts without any algorithm, randomly choosing its action from four directions.

As we mentioned before, we choose four kinds of classical reinforcement learning algorithms to compare our algorithm's performance, which include Q-learning, double Q-learning, Q-

TABLE I

RESULTS OF AVERAGE WIN PROBABILITY AND STEP FOR CASE 1.

Indices \ Methods	MVARL	Random Walk	Q-learning	Double Q	Q-ER	MC
Win Probability	**79.3%**	30.6%	57.3%	61.3%	60.2%	62.6%
Average Step	**136**	480	284	311	259	249

learning with experience replay and Monte Carlo methods. To be fair, we implement several experiments to choose best parameters for each comparative method.

- Q-learning: best $\varepsilon = 0.3$
- Double Q-learning: best $\varepsilon = 0.2$
- Q-learning with experience replay: best $\varepsilon = 0.3, K = 20, T = 20, N = 40$
- Monte Carlo method: best $\varepsilon = 0.25$

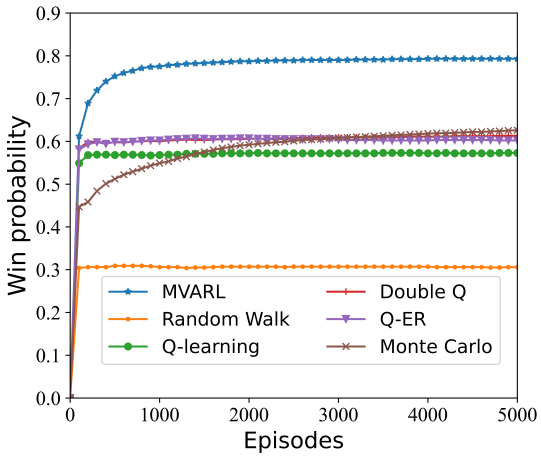


Fig. 6. Case 1: average win probability of different algorithms. Y-axis means the probability of the prey that can escape from the grid world under this case. The higher the win probability is, the better the algorithm's performance is. MVARL represents our proposed multi-virtual-agent reinforcement learning algorithm with best parameters $\varepsilon = \frac{\varepsilon}{1.1}, k = 5$. Double Q means double Q-learning method and Q-ER stands for Q-learning with experience replay method. Four comparative methods are implemented with best parameters which are all listed in Sec.IV.

The performance for experiments of case 1 is listed as the Table I shown and the curve of average win probability and the box-plot of average step are plotted in Fig.6 and Fig.7. Obviously, our algorithm MVARL has the most outstanding performance both in terms of average win probability and average step. The win probability of our MVARL algorithm is 79.3%. It outperforms 26.6% than the optimal comparative reinforcement learning method (Monte Carlo method) and is 2.6 times bigger than blank experiment. Besides, the step of Monte Carlo method is 1.83 times bigger than our proposed algorithm, where MVARL only needs 136 steps averagely to reach the goal space.

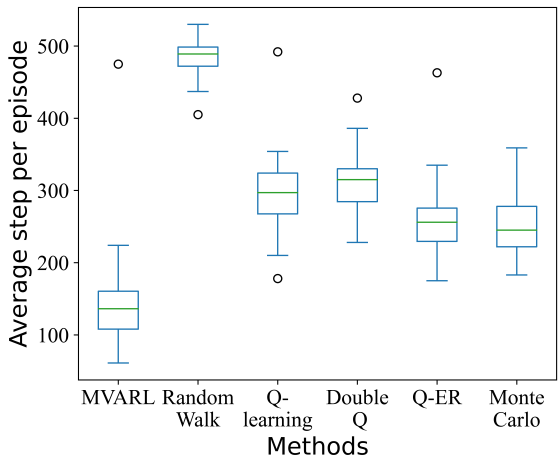


Fig. 7. Case 1: average step of different algorithms. Y-axis means how many steps the prey needs to escape form the grid world under this case averagely. And if the terminal state is the lose state, the step will be set as 600. Obviously, the lower the step is, the better the algorithm's performance is.

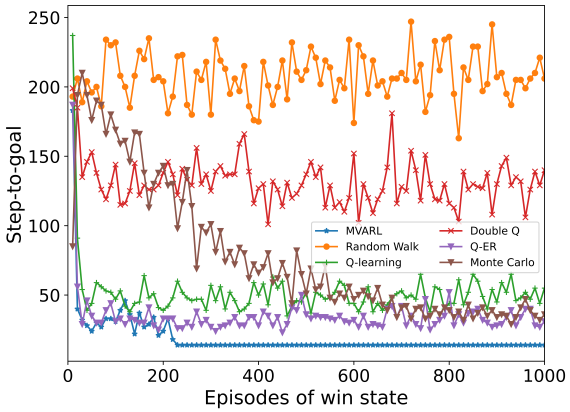


Fig. 8. Case 1: average step-to-goal of the first 1000 win states of different algorithms. We represent the actual steps the prey needs to take to get to the goal position under win states.

The reason for the average step is a big number is because the steps are averaged in each episode for different trials. So, average step will be larger than actual steps in the win state due to the step is set as 600 in the lose state.

To reflect our results more intuitively, we also plot average step-to-goal curve of the first 1000 win states in 50 trials, which is shown in Fig.8. It's obvious that our multi-virtual-agent reinforcement learning algorithm converges fastest and its average step-to-goal is shortest with a value of 14.

D. Case 2

In order to have more generalized performance, we expand the grid world's size and the appearing scope of a predator. The experiment of case 2 is executed in a 20×20

TABLE II

RESULTS OF AVERAGE WIN PROBABILITY AND STEP FOR CASE 2.

Indices \ Methods	MVARL	Random Walk	Q-learning	Double Q	Q-ER	MC
Win Probability	**84.8%**	5.8%	66.2%	22.4%	70.0%	72.6%
Average Step	**116**	588	252	543	213	211

predator-prey game with a predator appearing possibility $P = [0.4, 0.3, 0.2, 0.1]$. The predator in each area has three possible positions. The environment is shown in Fig.9.

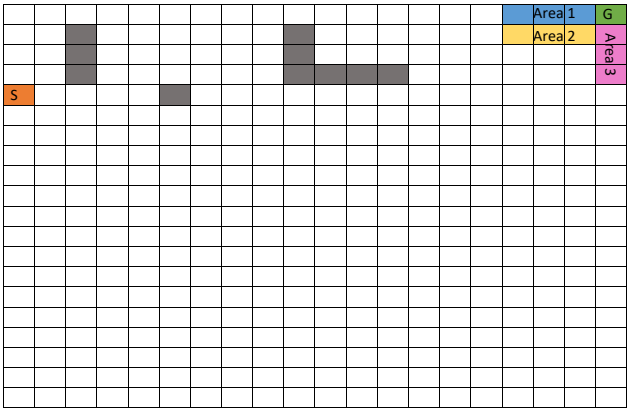


Fig. 9. Case 2: predator-prey game with size of 20×20. The size of grid world and the areas of the predator become bigger. This means the predator in each area has three possible positions to appear.

The performance for experiments of case 2 is listed in the Table II. The average win probability curve, avearage step's box-plot, and average step-to-goal curve of the first 1000 win states are plotted in Fig.10, Fig.11 and Fig.12. Similarly, we can conclude that our MVARL algorithm converges fastest with a step-to-goal attaining to 26 after convergence. Our MVARL algorithm achieves best performance in terms of both average win probability and step in case 2.

It's proved that when the size of grid world become bigger, our algorithm still outperforms other comparative methods. The average win probability is a little more than 80%. This is because the predator has more spaces to move around. We also noticed that the performance of double Q-learning method falls down a lot when the size of the grid world is expanded. This may be because double Q-learning method is based on the assumption of estimating $Q^A \approx Q^B \approx Q^*$. But for our case with a changing predator, the bigger the size of grid world is, the less possible for this hypothesis to become true. The update of two Q-tables leads the learning process to fall into a loop. Therefore, double Q-learning method will have worse results as the state space becomes bigger in this case.

V. CONCLUSIONS

In this paper, we propose an algorithm called multi-virtual-agent reinforcement learning method using several virtual

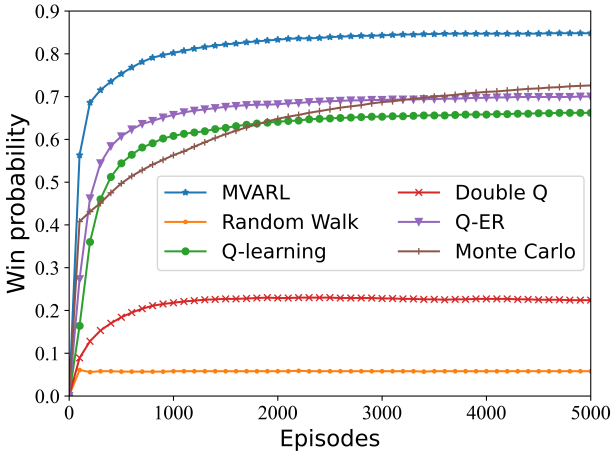


Fig. 10. Case 2: average win probability of different algorithms.

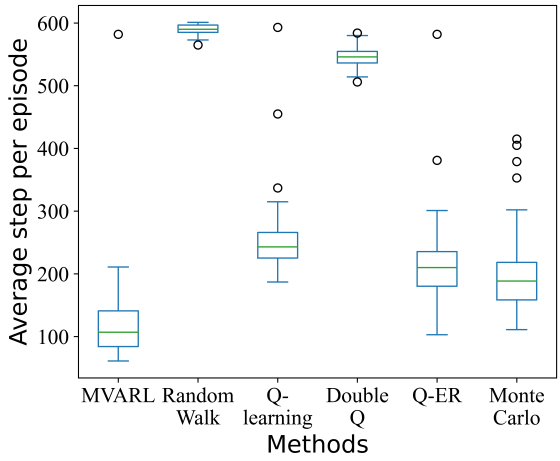


Fig. 11. Case 2: average step of different algorithms for case 2.

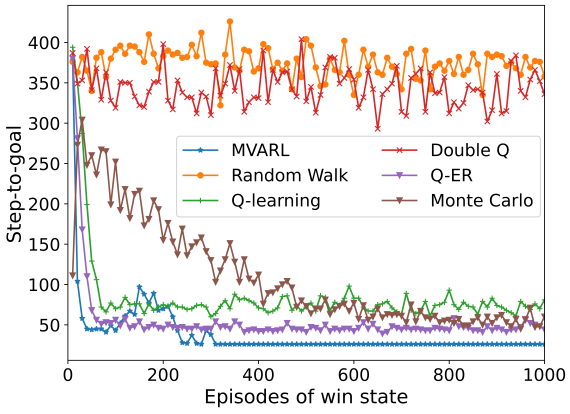


Fig. 12. Case 2: average step-to-goal of the first 1000 win states of different algorithms.

agents to simulate sub-environment of dynamic environment in parallel. The most important contribution of our work is to improve the robust of reinforcement learning algorithm and make it more efficiently to be applied in dynamic environments. Our experiments show that MVARL has better performance than other tabular learning methods, such as Q-learning, double Q-learning, Q-learning with experience replay and Monte Carlo methods both in terms of average win probability and average step indices. We proved that our conclusions are still valid when the size of the grid world in predator-prey game becomes bigger.

ACKNOWLEDGMENT

This work is partially supported by National Science Foundation under grants # 2047064, 2047010, 1947419, and 1947418.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester, "Challenges of real-world reinforcement learning: definitions, benchmarks and analysis," *Machine Learning*, vol. 110, no. 9, pp. 2419–2468, 2021.

[3] L. Bom, R. Henken, and M. Wiering, "Reinforcement learning to train ms. pac-man using higher-order action-relative inputs," in *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, 2013, pp. 156–163, doi: 10.1109/ADPRL.2013.6615002.

[4] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[5] Y. Lin, D. Duan, X. Hong, X. Cheng, L. Yang, and S. Cui, "Very-short-term solar forecasting with long short-term memory (lstm) network," in *2020 Asia Energy and Electrical Engineering Symposium (AEEES)*. IEEE, 2020, pp. 963–967, doi:10.1109/AEEES48850.2020.9121512.

[6] G. N. Iyengar, "Robust dynamic programming," *Mathematics of Operations Research*, vol. 30, no. 2, pp. 257–280, 2005.

[7] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810, doi:10.1109/ICRA.2018.8460528.

[8] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 06–11 Aug 2017, pp. 1126–1135. [Online]. Available: https://proceedings.mlr.press/v70/finn17a.html

[9] Z. Ni, X. Zhong, and H. He, "A boundedness theoretical analysis for gradp design: A case study on maze navigation," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–8, doi:10.1109/IJCNN.2015.7280475.

[10] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel, "Multi-agent reinforcement learning in sequential social dilemmas," *arXiv preprint arXiv:1702.03037*, 2017.

[11] H. Zafar, Z. Utkovski, M. Kasparick, and S. Stanczak, "Transfer learning in multi-agent reinforcement learning with double q-networks for distributed resource sharing in v2x communication," *arXiv preprint arXiv:2107.06195*, 2021.

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[13] A. Das, Z. Ni, X. Zhong, and D. Wu, "Experimental validation of approximate dynamic programming based optimization and convergence on microgrid applications," in *2020 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE, 2020, pp. 1–5, doi:10.1109/PESGM41954.2020.9281629.

[14] Y. Lin, D. Duan, X. Hong, X. Han, X. Cheng, L. Yang, and S. Cui, "Transfer learning on the feature extractions of sky images for solar power production," in *2019 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE, 2019, pp. 1–5, doi:10.1109/PESGM40551.2019.8973423.

[15] J. Schrum, *Competition between reinforcement learning methods in a predator-prey grid world*. Citeseer, 2008.

[16] X. Wang, J. Cheng, and L. Wang, "A reinforcement learning-based predator-prey model," *Ecological Complexity*, vol. 42, p. 100815, 2020.

[17] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2961–2970.

[18] X. Wang and D. Klabjan, "Competitive multi-agent inverse reinforcement learning with sub-optimal demonstrations," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5143–5151.

[19] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[20] Z. Ni, H. He, J. Wen, and X. Xu, "Goal representation heuristic dynamic programming on maze navigation," *IEEE transactions on neural networks and learning systems*, vol. 24, no. 12, pp. 2038–2050, 2013.

[21] H. Hasselt, "Double q-learning," *Advances in neural information processing systems*, vol. 23, pp. 2613–2621, 2010.

[22] M. Pieters and M. A. Wiering, "Q-learning with experience replay in a dynamic environment," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2016, pp. 1–8, doi:10.1109/SSCI.2016.7849368.

[23] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Machine learning*, vol. 22, no. 1, pp. 123–158, 1996.