IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received July 24, 2019, accepted July 31, 2019, date of publication August 5, 2019, date of current version September 4, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2933002

UAV Autonomous Target Search Based on Deep Reinforcement Learning in Complex Disaster Scene

CHUNXUE WU[1], (Member, IEEE), BOBO JU[1], YAN WU[2], XIAO LIN[3], (Member, IEEE), NAIXUE XIONG[4], (Senior Member, IEEE), GUANGQUAN XU[4], (Member, IEEE), HONGYAN LI[5,6], AND XUEFENG LIANG[7], (Member, IEEE)

[1]School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China
[2]School of Public and Environmental Affairs, Indiana University, Bloomington, CO 47405, USA
[3]Department of Computer Science, Shanghai Normal University, Shanghai 200234, China
[4]College of Intelligence and Computing, Tianjin University, Tianjin 300072, China
[5]School of Information and Communication Engineering, Hubei University of Economics, Wuhan 430205, China
[6]State Key Laboratory for Information Engineering in Surveying, Mapping, and Remote Sensing, Wuhan University, Wuhan 430072, China
[7]School of Information, Kyoto University, Kyoto CO 600-8586, Japan

Corresponding authors: Hongyan Li (hongyanli2000@126.com) and Xuefeng Liang (xliang@xidian.edu.cn)

This work was supported by the National Key Research and Development Program of China under Grant 2018YFC0810204.

ABSTRACT In recent years, artificial intelligence has played an increasingly important role in the field of automated control of drones. After AlphaGo used Intensive Learning to defeat the World Go Championship, intensive learning gained widespread attention. However, most of the existing reinforcement learning is applied in games with only two or three moving directions. This paper proves that deep reinforcement learning can be successfully applied to an ancient puzzle game Nokia Snake after further processing. A game with four directions of movement. Through deep intensive learning and training, the Snake (or self-learning Snake) learns to find the target path autonomously, and the average score on the Snake Game exceeds the average score on human level. This kind of Snake algorithm that can find the target path autonomously has broad prospects in the industrial field, such as: UAV oil and gas field inspection, Use drones to search for and rescue injured people after a complex disaster. As we all know, post-disaster relief requires careful staffing and material dispatch. There are many factors that need to be considered in the artificial planning of disaster relief. Therefore, we want to design a drone that can search and rescue personnel and dispatch materials. Current drones are quite mature in terms of automation control, but current drones require manual control. Therefore, the Snake algorithm proposed here to be able to find the target path autonomously is an attempt and key technology in the design of autonomous search and rescue personnel and material dispatching drones.

INDEX TERMS Deep reinforcement learning, Markov decision, Monte Carlo, Q-learning.

I. INTRODUCTION

Currently, drones have low levels of autonomy, and they don't have the autonomy to complete route self-planning, decision making, coordination, and mutual cooperation. However, as advanced autonomous unmanned systems, the UAV is destined to develop in the direction of high autonomy, low manual intervention, and high intelligence. Currently, most drones have reached the level of automatic control. However, these control actions are pre-programmed. In future UAS systems, people only need to assign tasks to drones as commanders instead of monitoring and controlling them in real time. AI is the key technology for the future of UAV systems to improve their independent performance [1]. The intelligence of search and rescue drones is mainly reflected in the path planning ability of autonomous flight and the ability to perform self-determination of tasks. Future drones should be able to autonomously plan flight paths based on their respective missions and corresponding constraints [2], [3]. When the constraints change, the drone will autonomously

The associate editor coordinating the review of this manuscript and approving it for publication was Zhanyu Ma.

adjust the flight path. The second intelligent trend of drones is the ability to understand and disassemble tasks. When faced with complex disaster search and rescue missions, they do not need people to assign tasks or make decisions, but to complete various tasks autonomously [4]. One of the characteristics of intelligent UAVs in the future is the ability to efficiently perform complex tasks through independent cooperation [5]. With the advancement of technology and technology, the future UAV system will become a truly advanced autonomous unmanned system. Multi-UAV work together is also a hot topic of current research. Some drone companies use cloud services to support multiple drones for aerial collaboration. The multi-UAV work together currently the most widely used area is Aerial acrobatics. There are still few examples of real use of multi-UAVs in disasters. Although cloud services to support multiple drones for aerial collaborative work are beyond the scope of this article, this is still a potential area of research.

What is the strategy? Backpropagation networks, convolutional neural networks, and recurrent neural networks are mostly in the process of completing classification problems—that is, the question of what type of label a sample is, or the input and output model of one sequence to another. This seems to be irrelevant to the function of a powerful humanoid robot in science fiction movies. Not only that, but even a AlphaGo with a simple function to play Go can't train in this way. If you want to make the robot (whether it is a humanoid robot or a non-humanoid) have learning functions, then you need to use this learning method [6].

Reinforcement learning presents some challenges from the perspective of deep learning. First, most successful deep learning applications to date require a large amount of hand-marked training data. On the other hand, reinforcement learning algorithms must be able to learn from frequently sparse, noisy, and delayed scalar reward signals. The delay between the action and the resulting reward (which may be thousands of steps) seems particularly daunting compared to the direct correlation between the input and the goal in supervised learning. Another problem is that most deep learning algorithms assume that data samples are independent, while in reinforcement learning, sequences of highly correlated states are often encountered. Furthermore, in reinforcement learning, the data distribution changes as the algorithm learns new behaviors, which may be problematic for deep learning methods that assume a fixed underlying distribution.

Learning to control Agents directly from high-dimensional sensory inputs such as vision and language is one of the long-term challenges of reinforcement learning. Most successful reinforcement learning applications that run on these areas rely on hand-crafted functionality as well as linear value functions or strategy representations. Obviously, the performance of such a system is highly dependent on the quality of the feature representation [7].

This paper demonstrates that convolutional neural networks can overcome these challenges and learn successful control strategies from raw video data in complex reinforcement learning environments [8], [9]. The variant training network using the Q-Learning algorithm uses random gradient descent to update the weights. In order to alleviate the problem of related data and non-stationary distribution, the empirical replay mechanism is used here to randomly sample the previous transitions, thereby smoothing the training distribution of many past behaviors.

The major contributions of this paper are summarized as follows: We apply the method of reinforcement learning to the process of simulating the autonomous exploration of the target by the drone in the game environment of the Snake Game. The snake body is used to represent the drone, and the hotspot is used to represent the target to be searched [10], [11]. Snake Game is a challenging and intensive learning platform that provides agents with high-dimensional visual input (resolution video, 30 frames) and a variety of interesting tasks that are difficult for human players to achieve. Our goal is to create a strategic agent that can successfully learn to play as many games as possible. The network does not provide any game-specific information or hand-designed visual features. Apart from video input, and without understanding the internal state of the simulator, rewards and terminal signals, and a range of possible actions, it is only learned from human players. In addition, the network architecture and all the hyperparameters used for training remain the same throughout the game. Figure 1 provides an example screenshot of a Snake for training.

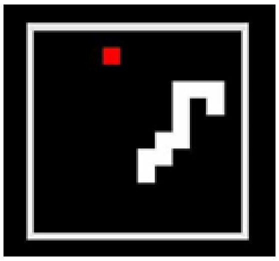


FIGURE 1. Snake game.

The rest of this paper is organized as follows: Chapter II introduces the techniques, designs to let the Snakes move autonomously and let the skills to speed up the training process. In Chapter III, we elaborate on our experimental plan. Chapter IV gives the relevant experimental process in detail, and analyzes the reliability of the experimental results. Finally, in Chapter V, we explain the conclusion and future work.

II. RELATED WORK

In terms of the intelligent implementation of simple 2D video games, the methods used by the predecessors include Sarsa, Contingengcy, Hneat, etc., but these have long been proven to be less powerful performance than reinforcement learning (2013 "Playing Atari with Deep Reinforcement Learning"). In fact, the use of deep learning and genetic algorithms to implement the smart snake algorithm has also been studied by the predecessors.[1] The general process is: use Pygame designed the Snake Game, which automatically generated training data in the back, and finally used GA+ANN to realize the training of Smart Snake. The method talked about the part of automating the production of training data. The predecessors don't think it's good method. The automation is not to let the game itself detect content to achieve automated snake. Instead, the angle is calculated according to a formula, and the process is finally realized. The subsequent GA algorithm also said that it is not the training data. He uses the GA algorithm in two parts of the program. The first is to use GA directly for training. The part of the fitness function is very similar to the method of generating rewards and punishments in reinforcement learning. The second place where the GA algorithm is used is the architecture used to select the neural network. There are also Q-learning implementations for the Snake games[2] (but it seems that his method is not very effective), its implementation principle is: use Q-Table saves some of the original state, through the reward mechanism, and according to the Q formula, the corresponding probability update. Finally, a table is obtained, which is the basis for the decision of the direction of the snake. In the process of subsequent training, it is necessary to continue to update this table. But Snake Game with Deep Learning and Q-learning method will take a lot of time to consider the new strategies and reward mechanisms, if Snake learning environment changed. In our theoretical research on ''during autonomous target search of drone based on deep reinforcement learning in complex disaster scenarios'', it is possible to encounter changing game experimental scenarios. In the DQN method we use, the reward mechanism is modularized using the Odor effect, and even if the game environment is changed, there are very few changes that need to be made. This is a very important reason why we use the DQN method to achieve the research content of this paper.

Q-learning is a very important off-policy learning method in reinforcement learning. It uses Q-Table to store the value of each state action pair. When the state and action space are high-dimensional or continuous, Q-Table is unrealistic. Therefore, the Q-Table update problem becomes a function fitting problem, the neural network is used to obtain the Q value of the state action, and the Q function is approximated to the optimal Q value by updating the parameter θ. This is the basic idea of DQN. In the original DQN, there is a case where the Q value is estimated to be high because it is an off-policy strategy. In order to decouple action selection from value estimation, Van Hasselt *et al.* proposed the Double-DQN method [12]. In Double-DQN, when calculating the actual Q value, the action selection is obtained by eval-net, and the value estimate is obtained by target-net. Double-DQN separates action selection from value estimation to avoid overestimation of value. In the traditional DQN experience pool, the selection of batch data for training is random, without considering the sample priority relationship. But in fact, the value of different samples is different, you need to give each sample a priority, and sample according to the priority of the sample. Schaul *et al.* proposed the Prioritized DQN method [13], which decomposes the Q value into state value and advantage functions to get more useful information. In the original DQN, the neural network directly outputs the Q value of each action. Wang, *et al.* proposed the Dueling DQN [14]. Dueling DQN The Q value of each action is determined by the state value and the advantage function. The dominant function can be simply understood as the difference between the value that can be obtained by an action and the average value that can be obtained for that state for a particular state. If the action taken takes a value greater than the average value, then the advantage function is positive and vice versa. This will give you more useful information. In DQN, the network outputs the expected value of the state-action value Q. This expectation actually ignores a lot of information. Bellemare *et al.* proposed the Distributional DQN [15]. In theory, modeling a deep reinforcement learning model from a distributed perspective can yield more useful information for better and more stable results. Increasing the exploration ability of Agent is a problem often encountered in reinforcement learning. A common method is to adopt the e-greedy strategy, that is, take the random action with the probability of e, and take the maximum value of the current value with the probability of 1-e action. Another common method is NoisyNet [16], proposed by Fortunato *et al.*, which increases the model's ability to explore by adding noise to the parameters. The Deep Reinforcement Learning Community has made some independent improvements to the DQN algorithm. However, it is unclear which of these extensions are complementary and can be effectively combined. DeepMind studied six extensions of the DQN algorithm and conducted empirical studies on their combination. On this basis, DeepMind proposed Rainbow [17]. And delivers state-of-the-art performance on the Atari 2600 benchmark, both in terms of data efficiency and ultimate performance.

A. MARKOV DECISION

All of the reinforcement learning training scenarios can be simplified as such models. Let's first look at which important objects are involved in the entire model.

The first is this subject, we can roughly understand it as a robot, and we just want to train its behavioral strategy. The second link is the environment and state, which is the ''situation'' of the subject at the time. Note that this situation is different in different occasions [18]. For example, for AlphaGo, the environment is the case of the chess board at that time. For the self-driving unmanned car, this environment is the data dimension of the current situation (the surrounding vehicle position, its own latitude and longitude, weather conditions), satellite map, vehicle speed and even tire pressure. In addition to this there are two important factors, Action and Reward.

[1] snake-game-with-deep-learning, please refer to https://theailearner.com/2018/04/19/snake-game-with-deep-learning/

[2] LearnSnake: Teaching an AI to play Snake using Reinforcement Learning (Q-Learning), please refer to https://italolelis.com/snake

ACTION can be understood as "action" or "behavior", which is the response or output that the robot has to make. For example, the position of the AlphaGo's chess piece, or a steering, throttle or brake command of the autopilot. Reward can be understood as "feedback" or "reward", but please note that "reward" may not always be a REWARD, but also a punishment. "Reward" as a positive value is a REWARD, just like a score. If it is a negative number, it means disciplinary or penalty. For example, after setting a piece in AlphaGo, the state of the checkerboard will change. If the checkerboard change is favorable at this time, it will get a positive value of REWARD. If the advantage is more obvious, the REWARD will be larger, and if the advantage is smaller, the REWARD will be smaller. On the contrary, if the checkerboard becomes more disadvantageous to itself, REWARD will take a negative value with a larger absolute value, and if there is a small disadvantage, a negative value with a smaller absolute value will be taken [19].

There are two more work to be done now. First, define these REWARDs and losses so that the REWARDs and losses generated in the environment can smoothly generate effective quantitative feedback to the subject. Second, let the subject quickly try to lower the cost at a lower cost to sum up the way REWARD works in different states [20].

So what kind of method can we use to get a better strategy? That is the hidden Markov chain is used to train, count the probability of state swapping and get the mathematical expectation of REWARD, and then find a path to get the maximum REWARD. We consider the task of the Agent interacting with the environment ε (in this case, the Snake Game) in a series of ACTIONs, observations, and REWARDs [21]. At each time step, the Agent selects an ACTION from the set of legitimate game ACTIONs $A = \{a_1, a_2 \ldots, a_k\}$. This will be passed to the simulator and its internal state and game score will be modified. In general, ε may be random. The Agent does not observe the internal state of the game; instead, it observes the image $x_t \in \mathbb{R}^d$ from the game, which represents the vector of the original pixel values of the current screen [22]. In addition, it receives a REWARD r_t indicating a change in the game score. Note that in general, the game score may depend on the entire previous ACTION and observation order; feedback on an ACTION can only be received after thousands of time steps [10], [23].

Since the AGENT only observes the image of the current screen, the task is partially observed and many game states are perceptually aliased, it is not possible to fully understand the current situation only from the current screen x_t. Therefore, we consider the sequence of ACTIONs and observations, $s_t = x_1, a_1, x_2, \ldots, a_{t-1}, x_t$, and learn the game strategies that depend on these sequences [24]. Assume that all sequences in the game are terminated with a limited number of time steps. This formalism produces a large and finite Markov decision process (MDP) in which each sequence is a unique state. Therefore, it is possible to simply use the complete sequence as a state representation of time as a standard reinforcement learning method for the MDP [10], [18], [23].

According to the state observed at each moment, an action is selected from the set of available actions to make a decision. The state of the next (future) of the system is random, and its state transition probability has Markov property. The decision maker makes new decisions based on the newly observed state, and repeats accordingly. Markov property refers to the nature of the probability that the future development of a stochastic process has nothing to do with the history before observation. Markov property can be simply described as no post-effect of state transition probability. The stochastic process with state transition probability and Markov property is the Markov process. The Markov decision process can be regarded as a special case of random countermeasures. In this random strategy, one of the countermeasures is unwilling. The Markov decision process can also be used as a Markov-type stochastic optimal control, and its decision variable is the control variable. In MDP, the optimal policy is a strategy that maximizes the probability weighted sum of future rewards. Therefore, the optimal policy consists of several actions that belong to a limited set of behaviors. In the fuzzy Markov decision process (FMDP), first, the value function is calculated as a regular MDP (i.e., has a finite set of actions); then, this policy is extracted by a fuzzy inference system. In other words, the value function is used as the input to the fuzzy inference system, and the policy is the output of the fuzzy inference system.

B. LOOP STORM EFFECT

In the process of intensive learning, the STATE is dealt with. In fact, many times the STATE is continuous, complex, and advanced. For example, if the picture is 128*128, then the number of STATEs is exponentially increasing, and the picture is continuous. Even if it is calculated at 30 frames per second, the speed of processing data cannot keep up with the speed of the game picture change. Therefore, resort to deep learning. Deep learning is very good at dealing with high-dimensional data and extracting patterns from it quickly. In image processing, a complete image is represented by an aggregate of pixels. At this time, the quality of the feature selection has a great influence on the classification or prediction results. Therefore, to choose a feature, how to choose a feature is very important to solve the actual problem. The artificial selection of features is a time-consuming and labor-intensive method that has no rules to follow in the face of a large number of unknown things. The choice of good or bad depends largely on experience and luck. Since the manual selection of features is not very good, can you let machine learning automatically learn some features? The answer is yes! Deep learning is used to do this. The deep learning alias is called Unsupervised Feature Learning, so the method of automatically learning features is collectively called deep learning.

The main skill used by DQN is experience replay, which is to save the rewards and status updates obtained each time with the environment for the update of the target Q value. Why do you need experience playback? We recall Q-Learning, which has a Q table to save the current results of all Q values, but DQN is not, then when doing the action value function update, you need other methods, this method is experience Playback. DQN has the ability to solve large-scale reinforcement learning problems because it approximates the value function. But DQN has a problem, that is, it does not necessarily guarantee the convergence of the Q network, that is, we may not be able to get the Q network parameters after convergence. This will result in a poorly trained model.

When the snake gains a constant value in the game, it will surround the constant value area by chance. And in the MDP, if the current sequence does not receive a penalty, and being repeated many times, this action will be strengthened and continued. This will result in a relatively bad result, and the AGENT will always go around this loop. We call it the ''loop storm effect.'' Figure 2 shows the trajectory that the AGENT may detour in the Loop Storm Effect. This method can be used in other places. A detailed application of the Loop Storm Effect in this experiment can be found in Appendix A.

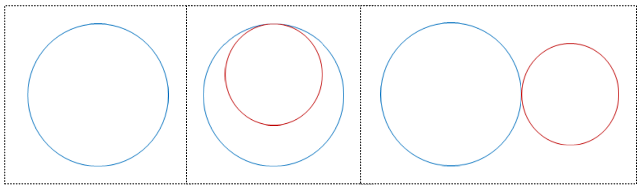


FIGURE 2. It shows the trajectory of a snake in the ''loop storm effect'', and the simplest loop trajectory is shown in Figure a. b and c show two more complex circular paths. More complex circular paths are repeatedly nested with path implementations in b and c.

C. ODOR EFFECT

Reinforcement learning is a process of continuous decision-making. The traditional supervised learning in machine learning is to give some annotation data, learn a good function, and make good decisions on unknown data. But sometimes, I don't know what the label is, that is, I don't know what is a ''good'' result at first, so RL is not a given label, but a reward function that determines what the current state gets. ''Good''or ''bad''), its mathematical essence is a Markov decision process. The ultimate goal is to optimize the overall reward function in the decision process. This process is a bit like supervised learning, except that the annotations are not prepared in advance, but are adjusted back and forth through a process and give so-called ''labeled data''. This process is reinforcement learning.

At each step, the environment gives the agent a reward, and the only goal of the agent is to maximize the total reward for long-term gain. The size of the reward reflects the quality of the event. The reward signal is the main basis for the change strategy. If the action selected by the strategy is low return, then in the future, the strategy may be changed to select other actions. Reinforcement learning is a simple framework for learning from interactions to achieve desired goals. Learners and decision makers are called agents, and interacting with agents is called the environment. The interaction continues, and the action environment selected by the agent reacts to the actions performed by the agent, causing the agent to be in another new environment. At the same time, the environment will pay off, and the agent tries to maximize these returns over time.

The ''nose''of the fly - the olfactory receptors are distributed on a pair of tentacles on the head. Each ''nose''has only one ''nostril''that communicates with the outside world and contains hundreds of olfactory nerve cells. If an odor enters the ''nostrils,''these nerves immediately turn the scent stimulus into a nerve impulse that is sent to the brain. The brain can distinguish substances with different odors according to the different nerve impulses produced by different odor substances. Therefore, the fly's tentacles are like a sensitive gas analyzer to guide the flies to find out where the food is. In the animal world, a considerable number of animals rely on the sense of smell to find food. Inspired by this, different REWARD values are set around the hot spot to simulate the smell of food. In theory, the blindness of the AGENT action can be reduced (the target odor distribution is shown in Figure 3). This summary is just an abstract introduction to the odor effect, which is to make his concept more concise and compact. This method can be used in other places. A detailed application of the odor effect in this experiment can be found in Appendix A.

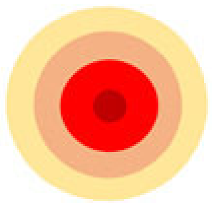


FIGURE 3. Target (hot spot) and its surrounding odor distribution, the closer to the center, the more intense the target smell, the greater the REWARD value.

III. THE PROPOSED AGENT SCHEME

A. MODEL

The Markov decision process is a decision process that is only related to the current state and has nothing to do with the previous state. As shown in Figure 4, the Markov decision process basically focuses on a situation in terms of thinking, that is, in a STATE (S_0, S_1, S_2, in the figure), how likely is it to select an ACTION? (a_0, a_1 in the figure), and how much a REWARD will be awarded for each STATE transition (with $+1$ and -1 on the curve turn arrow in the figure). The whole process is obtained from a large number of sample learning. This model is usually written as a four-tuple such as $\left(\mathrm{S,A,P}\left(s, s'\right), R\left(s, s'\right)\right)$. S represents the STATE, A represents the ACTION, P $\left(s, s'\right)$ represents the conversion probability between the two states s and s$'$, and R $\left(s, s'\right)$ represents the REWARD obtained by the conversion between the two states s and s$'$ [18].

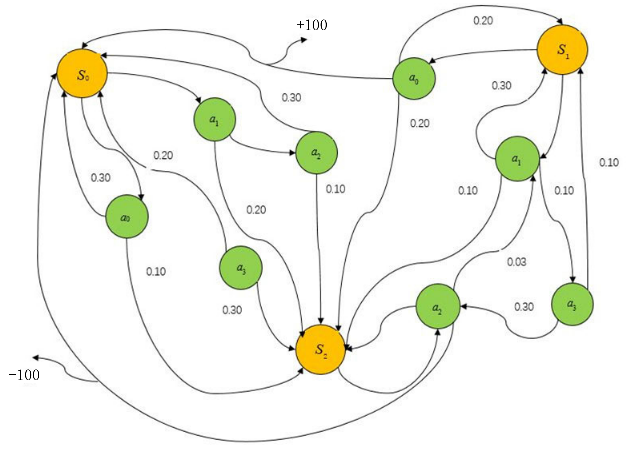


FIGURE 4. Markov decision process.

The goal of the AGENT is to interact with the simulator by selecting ACTIONs in a way that maximizes future REWARDs. Here, a standard assumption is made that the future REWARD is γ times per time step discount, and the future discount return of time t is defined as $R_t = \sum_{t'}^{T} \gamma^{t'-t} r_{t'}$, where T is the time step of the game termination. After seeing some sequence s and then taking some ACTIONs a, $Q^*(s,a) = \max_\pi \mathbb{Z}[R_t | s_t = s, a_t = a, \pi]$, the best action-value function $Q^*(s,a)$ is defined as the maximum expected return that can be achieved by following any strategy, where π is the ACTION (or distribution on the ACTION) The sequence of strategy maps [21].

The best ACTION value function obeys the Bellman equation. This is based on the intuition that if the best value $Q(s', a')$ of the sequence s' at the next time step is known for all possible ACTIONs a', then the best strategy is to select ACTION a' that maximizes the expected value of $r + \gamma Q^*(s', a')$,

$$Q^*(s,a) = \mathbb{Z}_{s' \sim \varepsilon}\left[r + \gamma \max_{a'} Q^*(s', a') \,|s, a\right] \tag{1}$$

The basic idea behind many reinforcement learning algorithms is to estimate the ACTION value function $Q_{i+1}(s,a) = \mathbb{Z}\left[r + \gamma \max_{a'} Q_i(s', a') \,|s, a\right]$ by using the Bellman equation as an iterative update. This value iterative algorithm converges to the optimal ACTION value function $Q_i \to Q^*(i \to \infty)$. In practice, this basic approach is completely impractical because the action-value function is estimated separately for each sequence without any generalization. Instead, a function approximator is typically used to estimate the ACTION value function $Q(s,a;\theta) \approx Q^*(s,a)$. In reinforcement learning, this is usually a linear function approximator, but sometimes a nonlinear function approximator, such as a neural network. We refer to the neural network function approximator with weight θ as the Q network. The Q network can be trained by minimizing the loss function sequence $L_i(\theta_i)$ that is changed at each iteration i,

$$L_i(\theta_i) = \mathbb{Z}_{s,a\sim\rho(.)}\left[(y_i - Q(s,a;\theta_i))^2\right] \tag{2}$$

where $y_i = \mathbb{Z}_{s'\sim\varepsilon}\left[r + \gamma \max_{a'} Q(s', a', \theta_{i-1}) \,|s, a\right]$ is the target i of the iteration, and $\rho(s,a)$ is the probability distribution of sequence s and ACTION a, which we call behavior distribution. When optimizing the loss function $L_i(\theta_i)$, the parameters from the previous iteration θ_{i-1} remain fixed. Note that the goal depends on the network weight; this is in contrast to the goal used to supervise learning, which is fixed before learning begins. Distinguish the loss function based on the weights that reach the following gradients [20],

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{Z}_{s,a\sim\rho(\cdot),s'\sim\varepsilon}\left[\left(\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}\right) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right] \tag{3}$$

Rather than calculating the expectation in the above gradient, the loss function is optimized by stochastic gradient descent, which is generally computationally advantageous. If the weights are updated after each time step and are expected to be replaced by a single sample from the ACTION distribution ρ and the game ε, then we get the familiar Q-Learning algorithm.

Note that this algorithm is modeless: it directly uses the samples from simulator ε to solve the reinforcement learning task without explicitly constructing the ε estimate. It is also non-strategic: it understands greedy policy $a = \max_a Q(s,a;\theta)$ and follows behavioral assignments to ensure full exploration of the living space. In practice, the behavioral distribution is usually chosen by the $\varepsilon - greedy$ policy that follows the greedy policy, with a probability of $1-\varepsilon$, and a random ACTION with a probability of ε.

During the training process, the trajectory needs to be continuously sampled. According to the

$$\pi(a_t|s_t) = \begin{cases} 1, & a_t = \arg\max_{a_t} Q(s_t, a_t) \\ 0, & otherwise \end{cases}$$

policy, the algorithm always selects the maximum action execution of the Q function. However, there is a problem here. During the training process, the current optimal Q value may be only a local minimum, so in the training sampling process, the policy needs to try other actions with a certain probability. The more commonly used exploration policy are:

① $\epsilon - \text{greedy}: \pi(a_t|s_t) = \begin{cases} 1-\varepsilon, & a_t = \arg\max_{a_t} Q_\phi(s_t, a_t) \\ \dfrac{\varepsilon}{|A|-1}, & otherwise \end{cases}$

② Boltzmann exploration: $\pi(a_t|s_t) \propto \exp\left(Q_\phi(s_t, a_t)\right)$

The exploration strategy (or policy gradients) used in this study is: ε-greedy policy.

DQN (Deep Q-Network) uses a deep neural network to fit an algorithm. The generalized expression is as follows [25].

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + a \times [r_{t+1} + \gamma MAXQ(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{4}$$

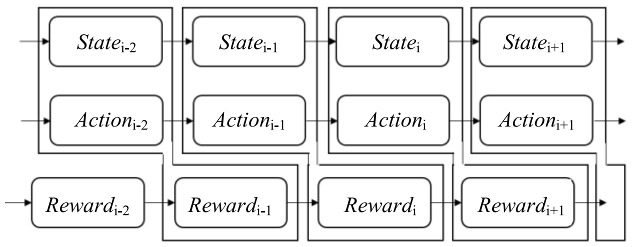


FIGURE 5. **DQN optimization of Markov decision process.**

First, three sequences of $State_i$, $Action_i$, and Reward$_i$ in the time series are obtained, and they are arranged in chronological order after receiving. Then follow the logic one by one from start to finish. At a time t, observe the s_i and a_i entered at this moment. If s_i and a_i do not have a corresponding REWARD value (that is, the first occurrence), then record this s_i and a_i directly [25], [26].

If you find that there is a larger value among all the possibilities of a_{t+1} in the case of s_{t+1}, that is, find the largest REWARD value in the case of s_{t+1} and multiply by a coefficient γ between 0 and 1, use this The value plus s_i and a_i should be obtained by rewarding the value r_{t+1} and subtracting the $Q(s_i, a_i)$ value from the previous step to update $Q(s_i, a_i)$. This logic is actually to avoid short-sightedness [26].

As shown in Figure 5, when iteratively calculating, under the next STATE(s_{t+1}), a ACTION (a_{t+1}) with more REWARDs will forward its return value to the previous STATE(s_t) and transfer part of the previous REWARD of the line to s_{t+1} through the ACTION(a_t).

Note that since the sequence of STATE and ACTION obtained in a large number of games is recorded, it is unlikely to form a separate chain like the one above. But even from the very beginning in the same STATE, the ACTION will be converted to a different STATE, and the new STATE has the same characteristics - it can also be converted to a different STATE due to different ACTIONs [27].

So the whole model is like a tree, a very wide and deep tree (as shown in Figure 6). A STATE is a node on such a tree [28]. When it does an ACTION, the state is transferred. It is equal to entering the tree from top to bottom. The number in the ellipse represents REWARD. When you get this tree, of course, you want to see which of the states below the ACTION, which one is the largest, which ACTION is chosen - this judgment logic is very natural. However, the REWARD value made by a transfer sample is actually very one-sided. To put it simply, in order to get a target and cause the next step to die, the meaning of this target does not exist - for example, the two branches on the left are similar. The last −100 represents a serious punishment. So the REWARD value or penalty value generated in the next step should go back up to the root along the tree, because to make the whole "Action Chain"look better, it is better to find a point near the root, and In order to correct such a mechanism introduced by this evaluation. Otherwise, if you use a method similar to greedy, you will follow a path like $State_1$, $State_2$, $Action_3$, and $State_4$, and this does not seem to be the best way to gain REWARD. You can see with the naked eye that the two leftmost paths are stronger than it. Note that the $State_3$, or $State_6$ type here is not necessarily the third or sixth occurrence state, which is just a label value representing a STATE.

$$Q(s_t, a_t) \leftarrow (1-a)\,Q(s_t, a_t) + a \times [r_{t+1} + \gamma MAXQ(s_{t+1}, a_{t+1})] \quad (5)$$

Note that there is a phenomenon here, if in this process, even if you can't get real-time REWARD - that is, if there are no methods to get an ACTION every time and then get an REWARD, in this case, at the end of the final game As

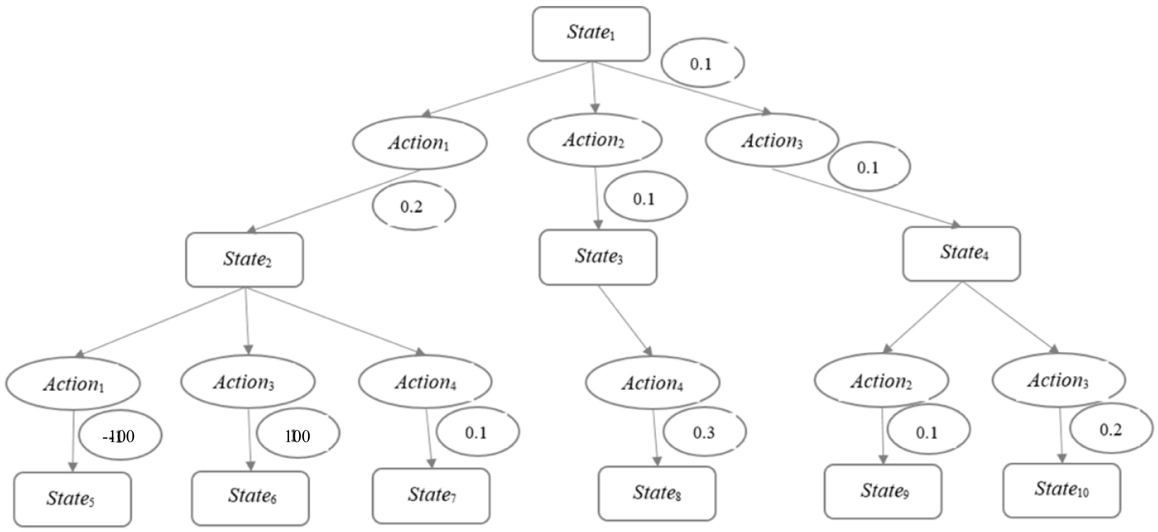


FIGURE 6. **Monte Carlo model.**

long as a large positive value of REWARD is placed in the last state as a REWARD, in theory, the entire tree can also pass the REWARD value up, and of course it can find the most suitable path. For example, the above figure can also be found by the Q-learning algorithm [29].

$$State_1 \rightarrow Action_1 \rightarrow State_2 \rightarrow Action_3 \rightarrow State_3 \rightarrow Action_5 \rightarrow State_3 \quad (6)$$

Such a path, backtracking will pass the REWARD value up, in the previous STATE under the learning of the Q-Learning algorithm will make those STATEs that can only lead to a larger reporting path look more "good", and ultimately The STATE that leads to greater disciplinary looks a bit "not good" [30]. By doing this, the robot can see which path is closer to the root of the tree and which path is easier to obtain in the long run. REWARD. This REWARD is 100 here, but it doesn't mean that you have to give such a disparity value to train. The other intermediate processes have already given 0. In fact, if this place obtained 1 can also be trained to get a result, because even if REWARD is bad, The zero point can also be compared and scaled in $MAX\ (s_{t+1}, a_{t+1})$.

Looking back at the entire training process, you will find that the process is also very clear. If the REWARD obtained at a certain step is relatively large, then it is inferred that the STATE and ACTION of the previous step are relatively good, thereby improving the REWARD evaluation of the previous step STATE. Under a large number of training samples, those often recurring states with high REWARD will be extensively verified and enhanced to learn some good paths. These paths consist of a series of STATEs and ACTIONs that form a complex decision making process. The entire section discussed above belongs to the category of dynamic programming, using the Monte Carlo model [31].

A conventional neural network defines a network structure and defines a loss function to describe the error. The result of the final iteration is to gradually reduce the loss function of the entire network to be small enough. The idea of DQN is to use a STATE vector as the input to the network, and the final output is an ACTION vector [32]. In this case, each time STATE passes through the network to obtain an ACTION output, the complex in-circle relationship in the network is used to fit this complex decision-making process [33]. But there is a strange place here, that is, as if we couldn't think of what should be used as a loss function, how to describe a so-called "residual" and let the "residual"move through the convex optimization in the direction of decreasing?

According to the Bellman equation of the famous mathematician Richard E. Bellman [34], there can be such a deduction:

$$Q_t^*\ (S_t, A_t) = E\left[r_{t+1} + \max_{a_{t+1}} Q_{t+1}^*\ (S_{t+1}, A_{t+1}, a_{t+1})\ |S_t, A_t\right] \quad (7)$$

That is, the convergence process of Q-Learning is a process of continuously iterating and moving the REWARD up part of the lower part of the entire tree [35].

It need minimize:

$$E\left[r_t + \max_{a_{t+1}} Q_{t+1}^*\ (S_{t+1}, A_{t+1}, a_{t+1}) - Q_t\ (S_t, A_t; \theta)\right]^2 \quad (8)$$

Or minimize:

$$E\left[Q_t^* - Q_t\ (S_t, A_t; \theta)\right]^2 \quad (9)$$

This θ is a meaning to the condition θ we use in statistics, and represents a parameter that describes the sensitive factors in the experimental environment. Simply put, every iteration calculation will occur that REWARD is worth spreading from "leaf"to "root", and this error is defined as the amount of propagation. Let the mapping of the network be constantly adjusted [36]. This mapping satisfies the mapping from one state to another and minimizes Loss [27]:

$$Loss\ (w) = E\left[r + \gamma \max Q\left(s', a', w\right) - Q\ (s, a, w)\right]^2 \quad (10)$$

Note that the expression for the entire network here is written as $Q\ (s, a, w)$.

Needless to say, w is to describe the parameters in the ownership of the entire network; s is STATE, output a best ACTION, should be such a structure. But this expression is not the same as the way we want it, we still need to make a transformation [6].

Since we need such a functional network, we construct a network with input STATE and output bit ACTION. The ACTION output is not in its original appearance, the output is in the form of $Q\ (s, a)$, and the output bit ACTION network is a multi-bit vector (see Figure 7). The dimension of this vector is n, n is the number of types of ACTION, each dimension is a Q value, or simply understood as a REWARD value. After the end of the network convergence, the n dimension vector will be output during the fitting process. Which dimension of the vector has the highest Q value, then which ACTION is selected as the result of the decision [26].

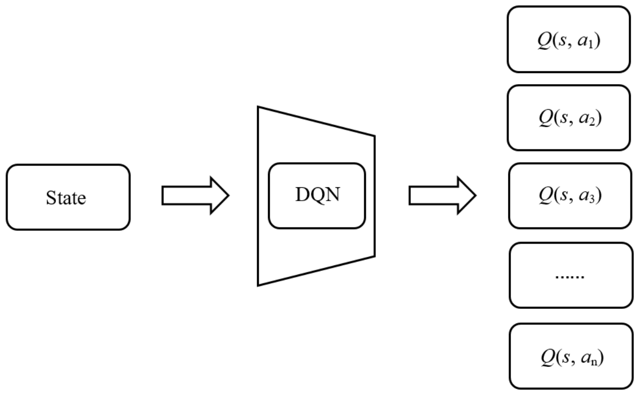


FIGURE 7. The process of Q function predicting the next ACTION in DQN.

B. PRETREATMENT

Direct use of raw picture frames (240 × 240 pixel images with 128 tones) may be computationally demanding, so we applied basic pre-processing steps aimed at reducing the input dimensions. The original frame is preprocessed by first converting the RGB representation to grayscale and downsampling it to an 80 × 80 image. The final input representation is obtained through an 80 × 80 area that roughly captures the playing area. The final clipping phase is needed because we are using a 2D convolution GPU implementation that requires square input. For the experiments in this paper, the function φ from Algorithm 1 applies the last 4 frames of the history in pre-processing and stacks them to produce the input of the Q function.

Algorithm 1 Deep Q-Network Algorithm

```
Initialize replay memory $D$ to size $N$
Initialize action-value function $Q$ with random weights
for episode $= 1, M$ do
    Initialize state $s_t$ and preprocessed sequenced $\phi_1 = \phi\{s_1\}$
    for $t = 1, T$ do
        With probability $\epsilon$ select random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe $r_t$ and $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
        Sample a minibatch of transitions $(s_j, a_j, r_j, s_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j & \text{for terminnal } s_{j+1} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta) & \text{for non-terminal } s_{j+1} \end{cases}$
        Perform a gradient step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to $\theta$
    end for
end for
```

There are several possible ways to parameterize Q using neural networks. Since Q maps historical ACTION pairs to scalar estimates of their Q values, historical ACTION has been used as input to neural networks by some previous methods. The main disadvantage of this type of architecture is that a separate forward transfer is required to calculate the Q value of each ACTION, resulting in a cost that is linearly proportional to the number of ACTIONs [37]. Instead we use an architecture where each possible ACTION has a separate output unit, and only the state representation is the input to the neural network. The predicted Q value of a single ACTION corresponding to the input state is output. The main advantage of this type of architecture is the ability to calculate the Q values of all possible operations in a given state, with only one forward pass through the network.

The exact architecture for the Snake Game is now described (the network architecture is shown in Figure 8 below. The convolutional network here can be replaced by other better solutions. ResNet has been widely proven to improve the extraction accuracy of visual features [38]. Reference [39] proposed a Channel Max Pooling Modified CNNs is also one of the most competitive visual feature extraction methods that can be modified and utilized in our

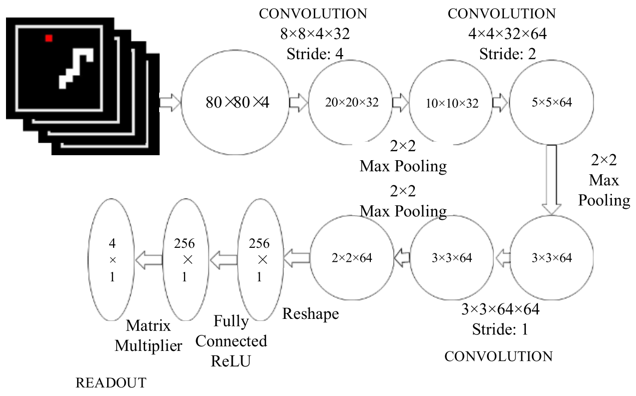


FIGURE 8. Neural network architecture for a snake game.

DQN solution.). The first layer of CNN convolves the input image with an 8 × 8 × 4 × 32 kernel of step size 4. The output of CNN is then passed through 2 × 2 max pooling. The second layer is convolved with a 4 × 4 × 32 × 64 kernel in steps of four. Then use 2 × 2 max pooling again. The third layer is convolved with a 3 × 3 × 64 × 64 kernel in steps of one. Then pass 2 × 2 max pooling again. The last hidden layer consists of 256 fully connected ReLU nodes. The output layer is a fully connected linear layer with one output for each valid operation. In the game we considered, the number of effective ACTIONs was 4. The final output layer has the same dimensions as the number of valid ACTIONs that can be performed in the game, where the 0th index always corresponds to nothing. Given the input state of each active ACTION, the value of this output layer represents the Q function. At each time step, the network uses the ε greedy policy to perform any ACTION corresponding to the highest Q value.

IV. PERFORMANCE ANALYSIS

A. TRAINING

This experiment uses the depth Q network (DQN) training of the above architecture (The training network model for multi-agent and multi-hotspots can be found in Appendix B. Only a single agent is described here. This is to avoid confusion in the description and to strengthen the contrast between the two process). Learning algorithms and hyperparameter settings show that our approach is powerful enough to handle a variety of games without including game-specific information. Although we evaluated our AGENT in real and unmodified games, we only made a change to the REWARD structure of the game during training. Since the proportion of the score varies from game to game, we fix all the positive REWARDs for finding the target to 100, all negative REWARDs are −100, and the REWARDs in the safe area are set according to the scent effect (ie the closer to the hotspot, the higher the REWARD, but the REWARD Will not exceed 100). This way will promote the performance of the AGENT, so that the AGENT can distinguish between different levels of REWARDs.

There are two ways to set the REWARD of the security zone in the game. The easiest way is to set the same REWARD in the security zone. When the AGENT hits the wall or your body, set a negative REWARD. When the AGENT eats the target, get a comparison. The big positive value is REWARD. In our experiments, different REWARDs were set in the security zone. Setting different REWARD values around the hot spot to simulate the smell of food. And it can reduce the blindness of the AGENT ACTION.

Determine whether a loop storm occurs by recording whether the first trajectory of the snake and the second trajectory are consistent. When a "loop storm"occurs, we let the game's own logic replace the reinforcement learning model to give the direction of the Snake's next ACTION. There was no the reason of "loop storm"in the experiment that caused AGENT to get any bad returns. Because of the randomness of the target's birth point, the AGENT's circular trajectory is necessary.

In the experiment, a total of 10 million frames were trained and the last one million frames of playback memory were used. First, the ownership weight matrix is randomly initialized which using a normal distribution with a standard deviation of 0.01, after that, the playback memory is set to a maximum size which has 100,000 experience. Training is initiated by randomly selecting ACTIONs in the first 100,000 time steps without updating the network weights. This allows the system to fill the playback memory before training begins. Noting this, unlike the initialization $\varepsilon = 1$, the linear annealing ε is from 0.1 to 0.0001 during next 10,000,000 frames. The reason for this setting is that the AGENT can select an ACTION every 0.03 seconds (or, FPS = 30) in the Snake game, high ε will make it excessive, so that the AGENT stays at the top of the 240×240 screen and finally crashes into the wall in a clumsy way. This situation makes the Q function converge relatively slowly, because it begins to take care of other conditions when ε is low. At each time step during the training time, the network samples a microcell of size 32 from the playback memory for training and performs a gradient step on the loss function described above using Adam optimization algorithm with a learning rate of 0.000001. After the annealing is completed, the network continues to train indefinitely, with ε fixed at 0.001. Figure 9 shows the training process. At the beginning of the training, the ACTION of the first 10,000 cycles is generated by the logic of the game itself. After extracting 4 frames of video images from the game simulator and converting them into 80×80 binary images, the STATEs vector corresponding to the ACTION in the game is evaluated. After 100,000 cycles of observation, the model generated by the DQN training began to predict the action of the AGENT in the game. The result of the model prediction always made AGENT get the REWARD trend to maximize.

AGENT's ACTIONs during 10 epochs of training are completely random, crashing around walls like headless flies. After training 20 epochs, AGENT has been able to find 3-4 hot spots in the game. After training for 30 epochs, AGENT becomes more and more intelligent, and can find

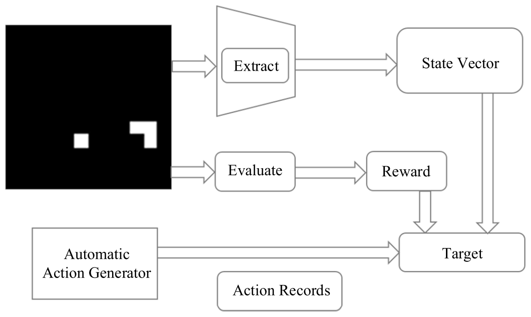


FIGURE 9. Snake training process.

about 8 hot spots. After training for 80 epochs, AGENT has been able to find at least 12 hot spots. When AGENT is trained to the 100 epochs, it can find at least 16 or more hotspots.

B. RESULTS AND ANALYSIS

In supervised learning, the performance of the model can be easily tracked during training by evaluating on training and validation sets. However, in reinforcement learning, accurately assessing the progress of the AGENT during training can be challenging. Since the evaluation indicator is the total REWARD collected by the AGENT in the game's average episode or game, it will be calculated periodically during the training. The average total REWARD indicator is often very noisy, as small changes in policy weights can lead to dramatic changes in the spatial distribution of policy access. The leftmost graph in Figure 10 shows the evolution of the average total REWARD during the Snake Game training. The average REWARD plot is really noisy, giving the impression that the learning algorithm has not made steady progress. Another more stable indicator is the estimated action value function Q of the policy, which estimates how many REWARDs the AGENT can get by following any policy for a given state. We collect a set of fixed states by running a random policy before the start of training and track the average of the maximum predicted Q of these states. The rightmost graph in Figure 15 shows that the average prediction Q is smoother than the average total REWARD obtained by the AGENT. Except for the relatively smooth improvement in predicting Q during training, no divergence was encountered in any of the experiments. This shows that, despite the lack of any theoretical convergence guarantee, the experimental method can train large-scale neural networks in a stable way using the reinforcement learning signal and the Adam optimization algorithm.

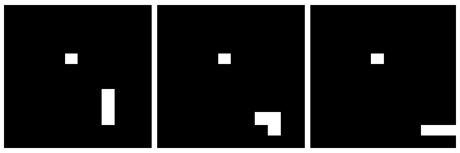


FIGURE 10. Performance of AGENT during 10 epochs.

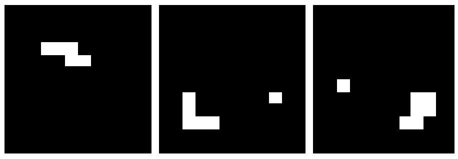


FIGURE 11. Performance of AGENT during 20 epochs.

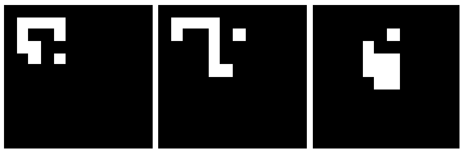


FIGURE 12. Performance of AGENT during 30 epochs.

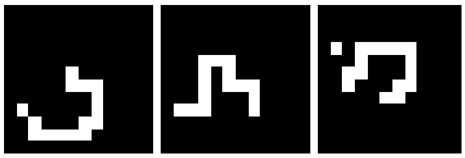


FIGURE 13. Performance of AGENT during 80 epochs.

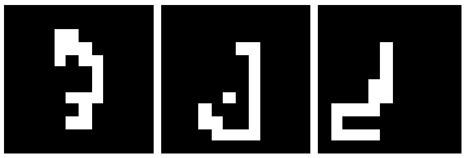


FIGURE 14. Performance of AGENT during 100 epochs.

During the experiment, we compared the methods of adding Breakout Loop Storm and Odor Storm respectively with did not add these methods, and proved that adding these methods is helpful to the correctness and rapid convergence of the training process, as shown in Figs 16, 17, It shows that although the number of occurrences of the loop phenomenon at the beginning of training is significantly higher in the average number per 100,000 training cycles, however, The use of Breakout Loop Storm technology in the training process makes the number of loop storms significantly reduced after the iteration of 6 million epochs. Compared to the Breakout Loop Storm technology, when the training reaches about 2 million epochs, the loop storm will last forever in no Breakout Loop Storm experiment. The Odor Effect was used in the training process making the Agent converge after 8 million Epochs. Compared with Odor Effect, there is still no convergence when training to about 10 million Epochs in no Odor Effect experiment.

We compare our results to the best performing methods in history. The method labeled Sarsa uses the Sarsa algorithm to learn linear strategies for several different feature sets of the AGENT, and we report the scores for the best performance feature set. Random events use the same basic method as Sarsa, but use the learning representation of the various parts of the screen under AGENT control to enhance the feature set. Note that both methods contain important prior knowledge about visual problems by using background subtraction and treating each of the 128 colors as separate channels. Since the Snake Game uses a different color for each type of object, treating each color as a separate channel can be similar to generating a separate binary map that encodes each object type. In contrast, our AGENT only accepts raw RGB screenshots as input and must learn to detect objects by itself.

We report the average scores of human game players and the policy of randomly selecting ACTIONs. Human performance is a median REWARD that is earned after each game. Report the average score under a fixed number of steps obtained by running a greedy policy. The first five rows of Table 1 show the average score for each game of the Snake Game. Although there is almost no prior knowledge of input, our method (labeled DQN) is much better than other learning methods.

We also compared the evolutionary policy search methods in the last three rows of Table 1. We report two sets of results

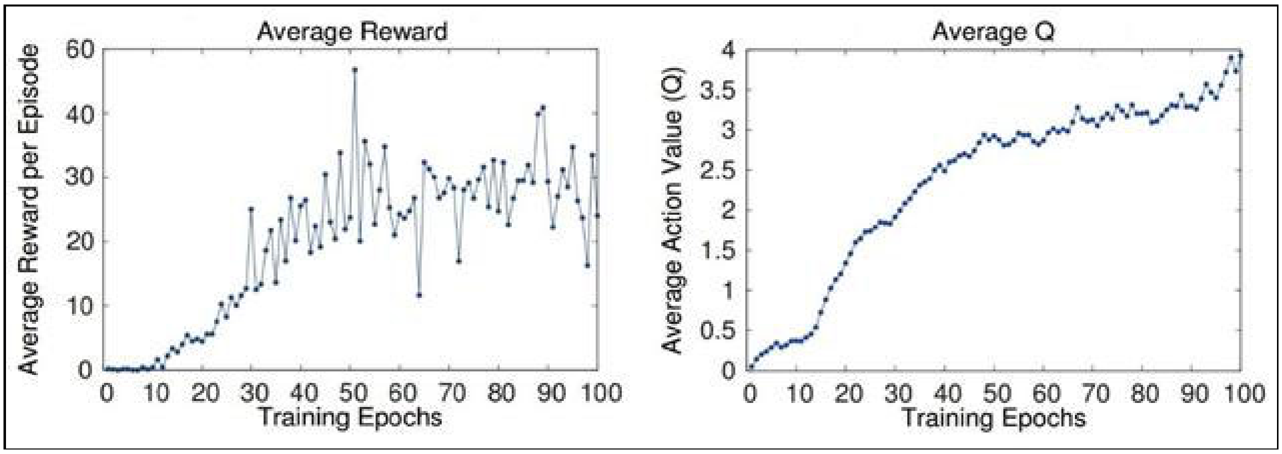


FIGURE 15. The graph on the left shows the average REWARD for each episode of the Snake during training. The statistics are calculated by the ε-greedy policy. The graph on the right shows the average maximum predicted ACTION value for a set of states on the Snake. An Epochs corresponds to 100,000 small batch weight updates or approximately 50 minutes of training time.

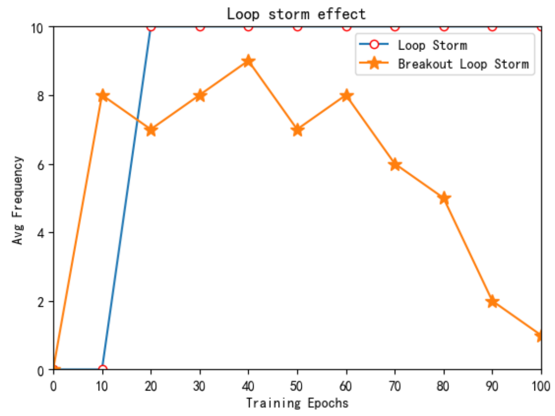


FIGURE 16. The use of Breakout Loop Storm technology in the training process makes the number of loop storms significantly reduced after the iteration of 6 million epochs. Compared to the Breakout Loop Storm technology, when the training reaches about 2 million epochs, the loop storm will last forever.

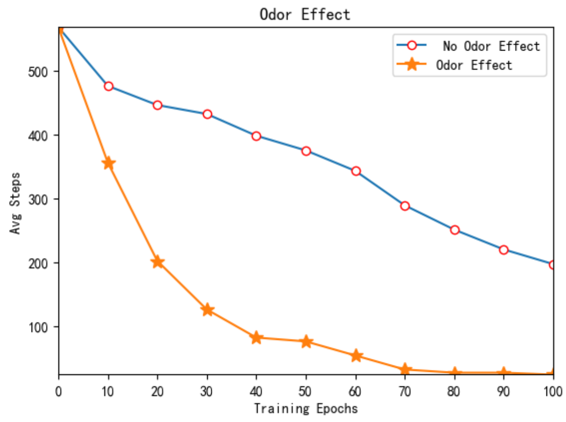


FIGURE 17. The Odor Effect was used in the training process making the Agent converge after 8 million Epochs. Compared with Odor Effect, there is still no convergence when training to about 10 million Epochs in no Odor Effect.

TABLE 1. The above table compares the average total REWARDs for various learning methods by running a fixed number of steps with's greedy policy with. The table below reports the results of the best episodes of HNeat and DQN singles. The deterministic policy generated by HNeat always gets the same score, while DQN uses's greedy policy.

Method	Average Score on Venomous Snake Game
Random	1
Sarsa	4
Contingency	6
DQN	**24**
Human	11
HNeat Best	13
HNeat Pixel	3
DQN Best	**58**

for this approach. The HNeat Best score reflects the results obtained using a hand-designed object detector algorithm that outputs the position and type of objects on the Snake game screen. The HNeat pixel score is obtained by using a special 8-color channel representation in the game, which represents the object label mapping at each channel. This approach relies heavily on finding a sequence of deterministic states that represent successful utilization. Strategies learned in this way are unlikely to be generalized to random perturbations. Therefore, the algorithm only evaluates on the highest score single event. Instead, our algorithm evaluates on the ε-greedy control sequence, so must be generalized wherever possible.

We have added some of our recent work in the actual scene which using the snake method. Although this is only a small range of tests, it is a good proof that the Snake method in this article is effective. These are two test trials that are closer to the target environment (Sports Ground Experiment (1) and Sports Ground Experiment (2). Note: In order to avoid confusion caused by cross description, we put Sports Ground Experiment (2) in Appendix B). In Sports Ground Experiment (1), Observer-Drone was lifted to a height of 80 meters to record the scene test situation of the sports ground of University of Shanghai for Science and Technology (This Observer-Drone also uses for special effect synthesis displaying the chessboard position which in computer. And when the return signal of the simulated GPS device is valid, Observer-Drone also assists in doing experimental analysis of data statistics). On the playground ground, the 12*12 checkerboard grid position is marked (each grid is 1.5m $\times$ 1.5m) which using White lines. The Target-Search-Drone (When performing the mission, the Target-Search-Drone flight altitude is 3 meters from the ground) is observed by Observer-Drone which send the video back to computer. Special effects processing on the video is added by computer to add 12*12 virtual blue checkerboard. In the experiment, it is necessary to ensure that the virtual blue checkerboard grid in the backhaul video of the Observer-Drone is aligned with the ground white checkerboard grid. As shown in Figure 18, the relative coordinates (0,0) are taken as the reference coordinates points in the upper left corner of the chessboard, and sequentially extended for (12,0) and (0,12) in the horizontal and vertical, and diagonally extended to (12,12). In the center of each cell on the chessboard, a simulated GPS signal return device is set by computer program (Here, we use computer to simulate the random generation of the GPS coordinate position in the chessboard, which is also can be regard as the relative coordinate of the central position of per-cell in the chessboard). The Target-Search-Drone automatically finds the analog GPS signal returner (the signal of each simulated GPS devices is set at random. When the drone reaches the cell where the analog GPS device is located, valid-signal analog GPS device is default found. Then, the current analog GPS device signal is disabled, but one of the return signal of the analog GPS device at other random locations is set valid, so that the drone continues to search for the next target). In the experiment of automatic Target-Search of single drones, we repeated 20 experiments (from the start of the operation to the range where the drone flew out of the chessboard is an experiment. Due to the limitations of the

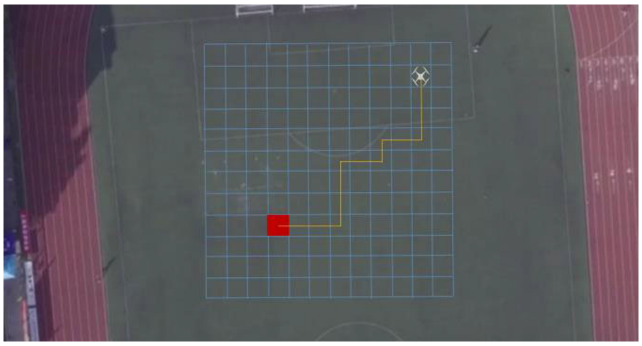


FIGURE 18. Single drone automatically finds objects. The figure is the scene of the Target-Search-Drone. The Target-Search-Drone is a white object in the chessboard. According to the coordinates returned by the Target-Search-Drone, special effects are used by computer to characterize the Target-Search-Drone. The Target-Search-Drone in the figure is not the actual size ratio of the on-site UAV. The analog GPS signal return device is the red dot position. (Note: Looking down DJI PHANTOM 4 PRO V2.0 from the height of 80 meters, in the case that the viewing angle covers all the 12*12 range of chessboards, it is almost difficult to see the Target-Search-Drone, so we are used the computer special effects enlarged version. The purpose of computer special effects and red dots is to let people better understand our experimental process).

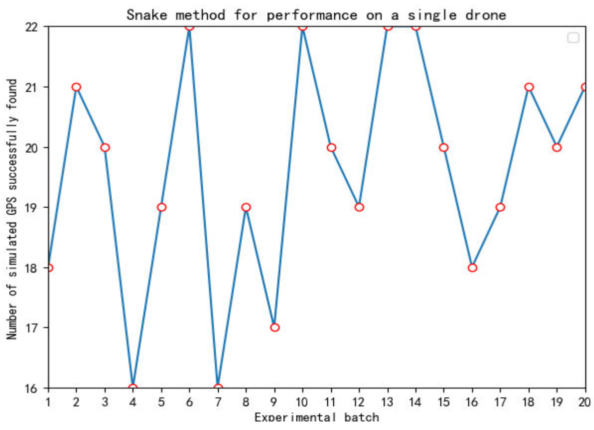


FIGURE 19. A single drone successfully found a quantitative analysis of a simulated GPS device in 20 experiments.

battery of the drone, our experiments are only carried out. 20 times, but this did not affect the Performance display of the algorithm), the actual performance of the Snake-Method is shown in Figure 19.

V. CONCLUSIONS AND FUTURE WORK

This paper demonstrates the ability of deep reinforcement learning to use the original pixel as input to grasp the difficulty control policy of the Snake game. It indirectly proves that the method can be applied to the multi-target search of the drone perfectly (Detailed information on multi-target search can be found in Appendix B). We also provide a variant of online Q-Learning that combines random small batch updates and experience replay memory to simplify the training of intensive learning deep networks. The final experimental results fully demonstrate that this method is effective for multi-target search of drones.

This method is used in the actual environment. This requires that the coordinates of the Agent and the coordinates of the hotspot in the simulation experiment be mapped to the coordinates of the drone's own coordinates and the coordinates of the target to be searched. We use a number of small drones DJI Phantom 4 Pro V2.0 (this drone is the first UAV designed specifically for artificial intelligence by DJI) as an agent. Achieve the target search work in a small space. In this process, we use a single PC with strong performance as our simulation cloud service platform. In the later research applications, we intend to replace the above equipment with a hybrid multi-rotor UAV (which can fly for up to 6 to 7 hours) and a professional cloud service work platform.

Although the Snake method works well in the actual Sports Ground Experiments (1) and (2) using the Snake method, its performance is not as good as its performance in 2D video games. This is because the drone has a certain deviation (1m∼3m) in the use of its own internal GPS module positioning fashion. Sometimes it is easy to fly to the internal of the adjacent grid due to the deviation, and this situation is not counted in found-by-UAVs-results of the data statistics of analog GPS signal device.

Further, the Snake algorithm that can independently find the target path can be applied to the following scenarios: ① UAV oil and gas field inspection. The plant area can be divided into a plurality of checkerboard areas, and a plurality of hot spot inspection work areas (high REWARD areas) can be randomly set, and the drones can realize the tracking inspection of these areas according to the set areas. ② Used in the search and rescue of personnel after the UAV complex disaster. For example, a city is divided into a relatively coarse chessboard area, and then the grid in each board (the jurisdiction in the city) can be divided into a sub-board area. When the drone detects the coordinates of the area of the person to be searched, the area can be set as a hot spot (high REWARD area) on the map. Use the drone's self-planning route to deliver relief supplies to multiple injured people (The method has a very detailed example of medical material distribution that can be found in Appendix C). ③ intelligent warehouse handling robots work together. Here, the smart warehouse handling robot is considered a snake and the cargo shelf is considered a hot spot. Imagine that there are many hot spots in the warehouse, and the length of the snake body will never increase. Each snake needs to find a hot spot that releases the same kind of identification signal (that is, the hotspot assigned to each snake can release a different scent to guide the corresponding snake to find itself). When the snakes hit the wall, the snakes collided, and the snakes ate the hot spots that were not their own, they were rewarded with a REWARD of -100. Only when the Snake eats the hotspot assigned to it, gets a REWARD of 100. Then experiment similar to the above, and finally, each smart storage robot can find its own most efficient working method.

It should be noted that although this paper has the most basic method of designing our system (We compared six classic DQN methods that can improve the environmental performance of snakes, in Appendix D). But there are still many places that can be improved or optimized. In the snake problem, the number of states that need to be searched is really huge, and although the Odor effect speeds up the training process throughout the training process, there are still some unnecessary search processes. For example, the loop storm path that the snake walks through. In the next chance, the snake may still fall into the same loop storm effect again. The alpha-beta pruning algorithm can remove some unnecessary searches. It can prevent the Snake from falling into the same loop storm that had passed before. At the same time, applying reinforcement learning to environments with sparse and unrecognized rewards is an ongoing challenge that needs to be promoted from limited feedback. Google AI solved this problem in a new method,[3] which can provide more accurate feedback for the agent, thus speeding up the training process. DeepMind's also mentions two methods[4] (One is Episodic Deep RL: Fast Learning through Episodic Memory, the other is Meta-RL: Speeding up Deep RL by Learning to Learn) to accelerate reinforcement learning and reduce the unnecessary searching process in training. In future research, we will continue to improve our snake method by referring to these optimization methods.

APPENDIX A

In the relevant work section, we present the odor effect and analyze the odor effect in the experimental analysis section and testified that odor effect can speed up the training process. In this section, we will introduce how we set up in a real simulation experiment. In order to avoid the chaos of the environment. We still use a single Agent (a Snake) to find a single hotspot (target) as an example. As shown in Figure 20 below, the reward value gradually decreases from the center hotspot. It is necessary to reasonably set the range of odor around the hot spot in the experiment. Because excessive or too little added odor will affect the final training process. Too little setting of the odor range is not obvious for accelerating the training process. Too much odor setting, perhaps speeding up training when training a single agent is obvious. But it is not good for multi-agent and multi-hots training. The odors between the hot spots interfere with each other, which will make the Agent blind and slow down the training process. There is no way to set it up to best. However, our experience in the experiment is "on the chessboard, there are mountains (light blue for the bottom of the mountain, dark blue for the mountainside, red spots for the top), and plains (black spots), the proportion of the plains should slightly more than mountain". However, even if it is a reasonable setting, in the environment of multi-agent and multi-hotspots, when several hotspots in the chessboard are close together, there will always be overlapping parts of

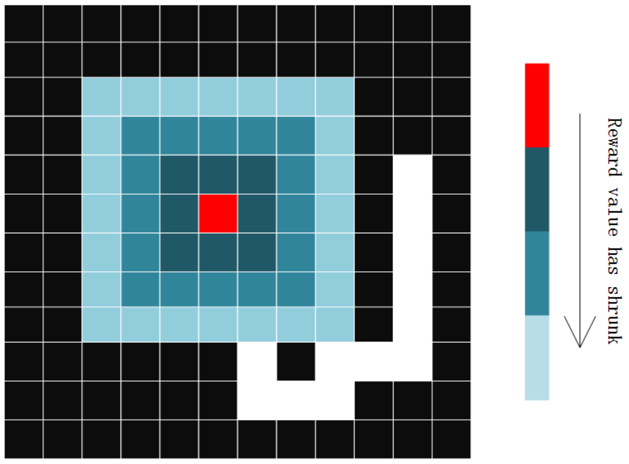


FIGURE 20. Odor effect. A gradient reduction reward mechanism is placed around the hotspot.

the ridge. At this time, in the overlapped sections, the higher reward value defaults submerged a lower reward value.

Similarly, we also introduced the loop storm effect in the relevant work section. So far, we have attributed the occurrence of the loop storm phenomenon to that the Agent has settled in this environment without punishment after receiving less rewards in the smooth zone and the mountainside. However, the Agent itself cannot jump out of the loop storm. This has been proven in many failed experiments. So we took a strategy to record the Agent trajectory. As shown in Figure 21, when the Agent walks the same trajectory for the second time, we replace the directional strategy given by the current training model with a random direction that is not on the current loop storm route. This measure effectively reduces the number of loop storms that occur during the

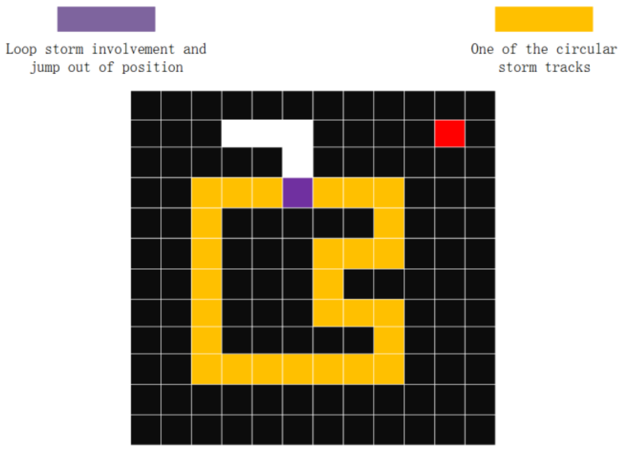

Loop storm involvement and jump out of position
One of the circular storm tracks

FIGURE 21. Loop Storm Effect. When entering the storm, use a dynamic array to record the coordinates of the first and second trajectories of the circular trajectory. When the first and second trajectories are the same, that is, there are two identical maximum strings in the dynamic array, we think there is a loop storm. At this point, the direction of the strategy given by the training model is replaced by a random direction that is not on the current storm path.

[3]Learning to Generalize from Sparse and Underspecified Reward, please refer to Google AI blog https://ai.googleblog.com/2019/02/learning-to-generalize-from-sparse-and.html?m= 1

[4]Reinforcement Learning, Fast and Slow, please refer to https://www.cell.com/trends/cognitive-sciences/fulltext/S1364-6613(19)30061-0

training period. When we train for 100 cycles, the loop storm phenomenon is almost non-existent.

APPENDIX B

First, in our experiment, the drone and the target to be searched were abstracted as coordinate points, and then the snakes were used to find food to behalf the process of UAV flight controlling strategy and the process of finding the target.

As shown in Figure 22, the white spots represent the snakes, and the red spots represent the foods to be sought (we call them hot spots in the article). The right image in Figure 1 is the gridded representation of the left image. That is, the chessboard we pointed out in the conclusion part of the paper, each black small grid is composed of corresponding coordinates. When the coordinates of the snake (the drone) moved to the coordinates of the food (hotspot) are equal, we believe that the target was found, the network model of multi-agent multi-hotspot environment training is shown in Figure 23, and Figure 26 shows the average reward for each episode in the training. It is well known that drones

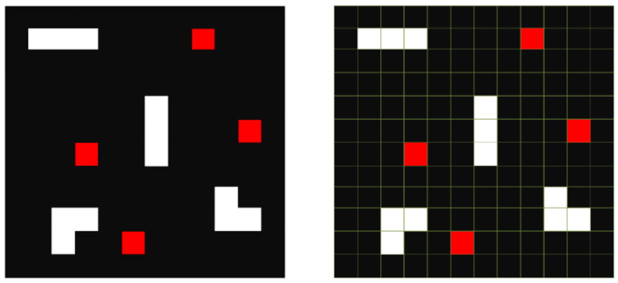


FIGURE 22. Visualization of the simulation model in the paper. In the simulation experiment, we use three white points to behalf the Agent, which can allow us to clearly distinguish the moving direction of the Agent in the experiment. But only the head vertex records the coordinate position of the Agent. Note: Four hotspots are not allowed to appear in the same coordinate position. Here, reward1, reward2, reward3, and reward4 represent the rewards obtained by Agent1, Agent2, Agent3, and Agent4, respectively. The reward received by the network model is REWARD_SUM = reward1 + reward2 + reward3 + reward4.

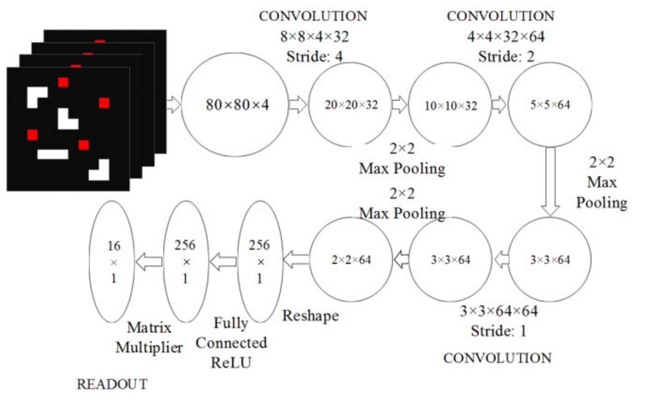


FIGURE 23. Neural network architecture for four snake game. Different from the previous single agent network structure, the network structure outputs 16 directions, and each four of the 16 directions constitutes a group of policies to control the moving direction of one of the four agents.

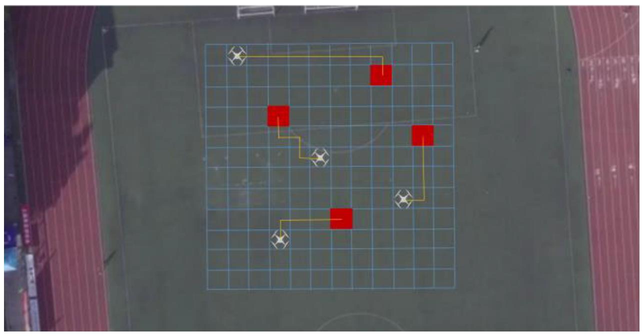


FIGURE 24. Multi-UAV automatically finds objects. The figure is the scene of the Multi-Target-Search-Drone. The Target-Search-Drone is a white object in the chessboard. According to the coordinates returned by the Target-Search-Drone, special effects are used by computer to characterize the Target-Search-Drone. The Target-Search-Drone in the figure is not the actual size ratio of the on-site UAV. The analog GPS signal return device is the red dot position. (Note: Looking down DJI PHANTOM 4 PRO V2.0 from the height of 80 meters, in the case that the viewing angle covers all the 12*12 range of chessboards, it is almost difficult to see the Target-Search-Drone, so we are used the computer special effects enlarged version. The purpose of computer special effects and red dots is to let people better understand our experimental process).

are capable of precise flight control and their internal GPS modules play a very important role.

As show in Figure 24, in the Sports Ground Experiment (2) of multi-UAV object-seeking, we used 5 drones, one of which was used for the Observer-Drone as the same with Sports Ground Experiment (1). And the other four were used to perform the Target-Search. Different from the Sports Ground Experiment (1), at this time, the computer program needs to make the analog GPS signal return device of 4 different positions valid at the same moment. The actual performance of the Sports Ground Experiment (2) is shown in Figure 25(a) and Figure 25(b) below.

APPENDIX C

Here we describe a process for using multiple drones to provide medical supplies to multiple mobile clinics. To illustrate the application scenario of the Snake algorithm. As shown in Figure 27. First, the disaster relief area is set up, and then the four mobile clinics send back their location. The material deployment center uses four drones to provide them with continuing medical supplies (such as anti-inflammatory drugs and plasma, etc.). The types and quantities of medical supplies carried on each drone are equal. When one of these materials is exhausted, it will automatically return to the material dispatching center to replenish it. Figure 28 shows the setting of the reward mechanism for the main area where the mobile clinic is located and its two sub-areas. The setting of the cascading area here is based on the following considerations: If the entire disaster area is regarded as a chessboard, it is relative to the unmanned The machine and the target it is looking for are too big, and maybe even the best server training it to convergence is quite difficult. Figure 29 shows the reward mechanism setting in the smoothing area (the black spot part of the right picture of Figure 22 or the white

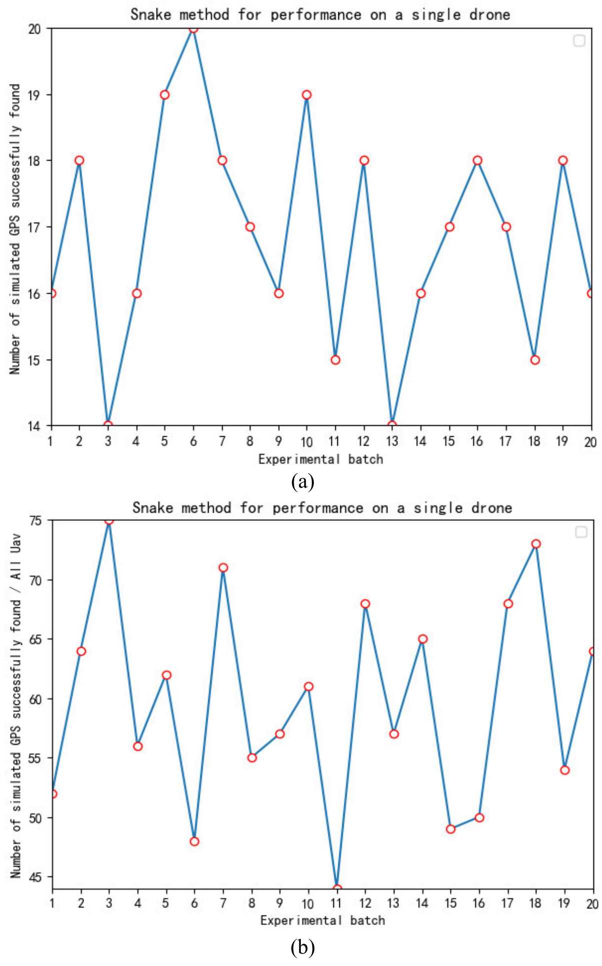


FIGURE 25. **(a) The number of found simulated GPS devices from the one of four drones which first escaped from the chessboard, in each experiment. (b) Four drones in 20 experiments, the sum of each time successfully found simulated GPS devices.**

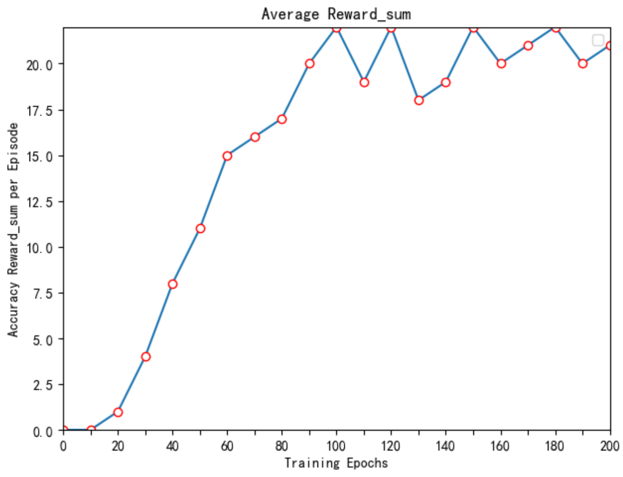


FIGURE 26. **It shows the average reward for each episode in the training. The calculated statistics are the ε-greedy policy.**

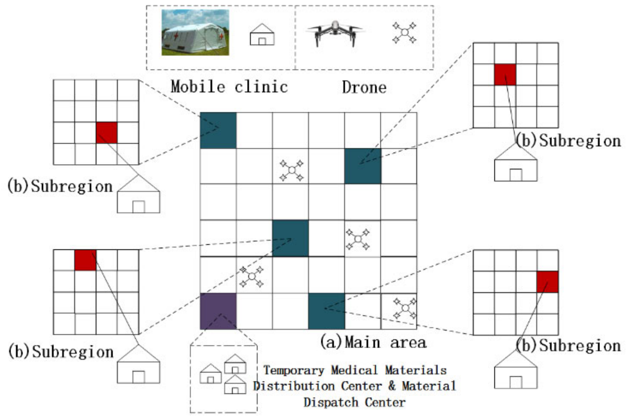
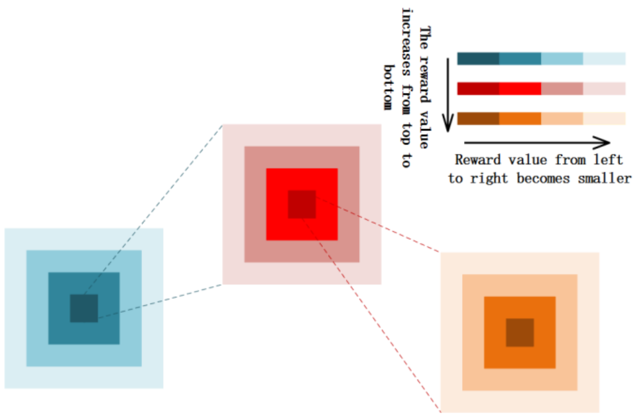


FIGURE 27. **Expands to multi-UAV search targets through the Snake algorithm. Here, our mobile clinic (the temporary medical team in the disaster area can send back its coordinates to the material dispatching center) can be built in any place where the disaster area needs. When a local rescue is completed, it can be removed and moved to the next place where assistance is needed. (a) The main area may contain (b) sub-areas and (b) sub-areas may contain (c) sub-areas and the like. Note: All the chessboards here have a uniform size of 12 × 12, although the image content is not show in this way for the sake of clarity. In the policy control of the main area, we use the multi-agent multi-hotspot model, and in the cascaded target area we use a single agent single hotspot model.**



The reward value increases from top to bottom
Reward value from left to right becomes smaller

FIGURE 28. **The reward mechanism part1 for the target area. This is a cascading reward mechanism.**

part of the main area of Figure 27). The smoothing area may be too large for the drone, so a gradient reward mechanism has been added to allow the drone to find the target more quickly (find the mean of the reward mechanism in the four-way neighbor smoothing area, and then calculate the pair with the highest difference between the top-bottom or the left-right, then choose the max difference as the direction of the gradient reward mechanism).

In the training, the single agent has no penalty mechanism in the actual application, and the multi-agent training environment only has a corresponding punishment mechanism when the collision occurs between the agents. We did not set a penalty mechanism at the checkerboard boundary to allow the Agent to escape from one chessboard to another chessboard (which is only moving from one square to another square relative to the upper chessboard area). When the Agent is inside the chessboard, the model control strategy corresponding to the chessboard plays a role. When the Agent escapes from a chessboard to an adjacent chessboard, the model control

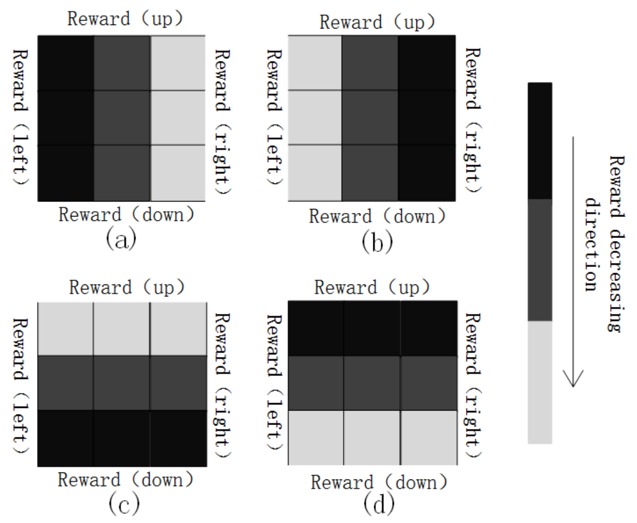


FIGURE 29. The reward mechanism part2 for the smooth zone. This plan of smooth zone is not necessary in the experiment, but it can speed up the training process. This method is used in the boundary between the edge of the scent and the smooth zone.

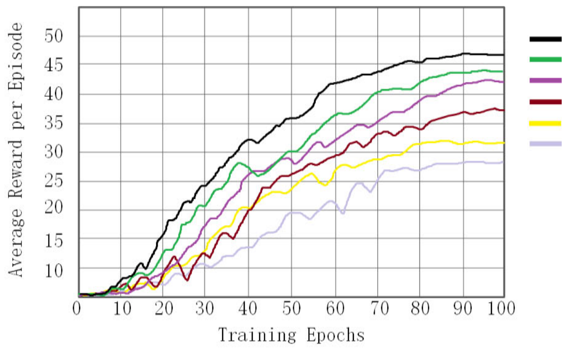


FIGURE 30. Analysis of the performance improvement of Single-Snake environment by six classical DQN methods.

strategy of the upper chessboard plays a role. Here our Agent control strategy is not controlled by a single network model. And by a number of cascaded models together play a role. In the horizontal direction, the control strategy of the adjacent chessboard will only work when the Agent crosses from the current chessboard to the adjacent chessboard.

APPENDIX D

In the study of this project, we paid more attention to how to make the snake environment more suitable for controlling the drone to carry out some auxiliary types of disaster relief. This is an area that has not been studied yet by others. We applied the most basic DQN method and In the process, it proved its effectiveness in the real environment. However, DQN has indeed produced quite a few variants in recent years. DeepMind integrates several of the most classic methods of the reinforcement learning community and proposes the Rainbow DQN approach, which combines some of the advantages of other approaches. Proved to be the best method in 57 Atari environments. Similarly, we analyzed the

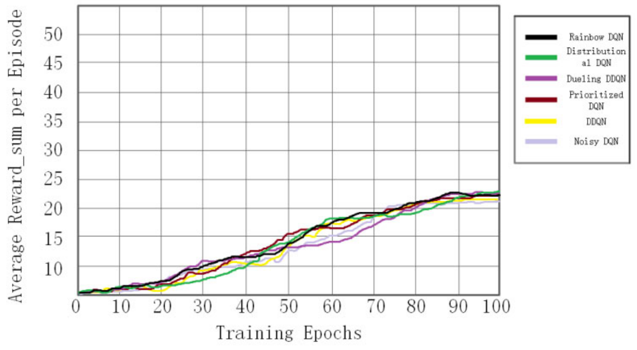


FIGURE 31. Analysis of the performance improvement of multi-snake environment by six classical DQN methods.

Rainbow DQN and the other five classic DQN methods in the snake environment of this study (see Figure 30 and 31). Proving that Rainbow can nearly double the scores of each snake in Single-Snake, but the performance is much smaller improved in multi-snake environment. This is estimated to be related to the crowded space of the multi-snake environment.

ACKNOWLEDGMENT

The authors would like to appreciate all anonymous reviewers for their insightful comments and constructive suggestions to polish this article in high quality. *(Chunxue Wu and Bobo Ju are co-first authors.)*

REFERENCES

[1] A. K. Yadav and P. Gaur, "AI-based adaptive control and design of autopilot system for nonlinear UAV," *Sadhana*, vol. 39, no. 4, pp. 765–783, Aug. 2014.

[2] H. Jiang and Y. Liang, "Online path planning of autonomous UAVs for bearing-only standoff multi-target following in threat environment," *IEEE Access*, vol. 6, pp. 22531–22544, 2018.

[3] S. Hayat, E. Yanmaz, T. X. Brown, and C. Bettstetter, "Multi-objective UAV path planning for search and rescue," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May/Jun. 2017, pp. 5569–5574.

[4] H. Zhang, D. Gao, Y. Bai, and H. Wang, "Collaborative tasks planning research for multi-UAV based on ant colony algorithm," *J. Beijing Univ. Civil Eng. Archit.*, 2017.

[5] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, and J. Aru, "Multiagent cooperation and competition with deep reinforcement learning," *PLoS ONE*, vol. 12, Apr. 2017, Art. no. e0172395.

[6] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke, "Towards vision-based deep reinforcement learning for robotic motion control," Nov. 2015, *arXiv:1511.03791*. [Online]. Available: https://arxiv.org/abs/1511.03791

[7] H. Cheng, Z. Su, N. Xiong, and Y. Xiao, "Energy-efficient node scheduling algorithms for wireless sensor networks using Markov random field model," *Inf. Sci.*, vol. 329, pp. 461–477, Feb. 2016.

[8] X. Xue, C. Wu, Z. Sun, Y. Wu, and N. N. Xiong, "Vegetation greening for winter oblique photography using cycle-consistence adversarial networks," *Symmetry*, vol. 10, no. 7, p. 294, 2018.

[9] S. Xi, C. Wu, and L. Jiang, "Super resolution reconstruction algorithm of video image based on deep self encoding learning," *Multimedia Tools Appl.*, vol. 78, no. 4, pp. 4545–4562, 2019.

[10] X. Jiang, Z. Fang, N. N. Xiong, Y. Gao, B. Huang, J. Zhang, L. Yu, and P. Harrington, "Data fusion-based multi-object tracking for unconstrained visual sensor networks," *IEEE Access*, vol. 6, pp. 13716–13728, 2018.

[11] J. Zhang, J. Geng, J. Wan, Y. Zhang, M. Li, J. Wang, and N. N. Xiong, "An automatically learning and discovering human fishing behaviors scheme for CPSCN," *IEEE Access*, vol. 6, pp. 19844–19858, 2018.

[12] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016.

[13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," Nov. 2015, *arXiv:1511.05952*. [Online]. Available: https://arxiv.org/abs/1511.05952

[14] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2015, *arXiv:1511.06581*. [Online]. Available: https://arxiv.org/abs/1511.06581

[15] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," 2017, *arXiv:1707.06887*. [Online]. Available: https://arxiv.org/abs/1707.06887

[16] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, "Noisy networks for exploration," 2017, *arXiv:1706.10295*. [Online]. Available: https://arxiv.org/abs/1706.10295

[17] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," 2017, *arXiv:1710.02298*. [Online]. Available: https://arxiv.org/abs/1710.02298

[18] L. Xia, J. Xu, Y. Lan, J. Guo, W. Zeng, and X. Cheng, "Adapting Markov decision process for search result diversification," in *Proc. Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 2017, pp. 535–544.

[19] J. Buongiorno and M. Zhou, "Multicriteria forest decisionmaking under risk with goal-programming Markov decision process models," *Forest Sci.*, vol. 63, no. 5, pp. 474–484, 2017.

[20] G. Neu, A. Jonsson, and V. Gómez, "A unified view of entropy-regularized Markov decision processes," 2017, *arXiv:1705.07798*. [Online]. Available: https://arxiv.org/abs/1705.07798

[21] R. Berthon, M. Randour, and J.-F. Raskin, "Threshold constraints with guarantees for parity objectives in Markov decision processes," 2017, *arXiv:1702.05472*. [Online]. Available: https://arxiv.org/abs/1702.05472

[22] E. M. Hahn, V. Hashemi, H. Hermanns, M. Lahijanian, and A. Turrini, "Multi-objective robust strategy synthesis for interval Markov decision processes," in *Proc. Int. Conf. Quant. Eval. Syst.*, 2017, pp. 207–223.

[23] X. Yu, X. Zhou, and Y. Zhang, "Collision-free trajectory generation and tracking for UAVs using Markov decision process in a cluttered environment," *J. Intell. Robotic Syst.*, vol. 93, nos. 1–2, pp. 17–32, 2019.

[24] K. Lee, S. Choi, and S. Oh, "Sparse Markov decision processes with causal sparse tsallis entropy regularization for reinforcement learning," *IEEE Robot. Automat. Lett.*, vol. 3, no. 3, pp. 1466–1473, Jul. 2018.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," Dec. 2013, *arXiv:1312.5602*. [Online]. Available: https://arxiv.org/abs/1312.5602

[26] G. Lample and D. S. Chaplot, "Playing FPS Games with deep reinforcement learning,"2016.

[27] A. Das, S. Kottur, J. M. F. Moura, S. Lee, and D. Batra, "Learning cooperative visual dialog agents with deep reinforcement learning," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2970–2979.

[28] M. Ishihara, T. Miyazaki, C. Y. Chu, T. Harada, and R. Thawonmas, "Applying and improving Monte-Carlo tree search in a fighting game AI," presented at the Proc. 13th Int. Conf. Adv. Comput. Entertainment Technol., Osaka, Japan, 2016.

[29] B. O'Donoghue, R. Munos, K. Kavukcuoglu, and V. Mnih, "Combining policy gradient and Q-learning," 2017, *arXiv:1611.01626*. [Online]. Available: https://arxiv.org/abs/1611.01626

[30] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2829–2838.

[31] D. J. N. J. Soemers, C. F. Sironi, T. Schuster, and M. H. M. Winands, "Enhancements for real-time Monte-Carlo tree search in general video game playing," in *Proc. IEEE Conf. Comput. Intell. Games (CIG)*, Sep. 2017, pp. 1–8.

[32] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2018, pp. 1–8.

[33] M. M. Zhang, N. Atanasov, and K. Daniilidis, "Active end-effector pose selection for tactile object recognition through Monte Carlo tree search," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 3258–3265.

[34] H. Pham and X. Wei, "Bellman equation and viscosity solutions for mean-field stochastic control problem," *ESAIM, Control, Optim. Calculus Variat.*, vol. 24, no. 1, pp. 437–461, 2018.

[35] X. Guo, S. Singh, R. Lewis, and H. Lee, "Deep learning for reward design to improve Monte Carlo tree search in ATARI games," 2016, *arXiv:1604.07095*. [Online]. Available: https://arxiv.org/abs/1604.07095

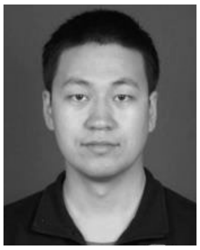
[36] J. Wang, W. Wang, R. Wang, and W. Gao, "Beyond Monte Carlo tree search: Playing go with deep alternative neural network and long-term evaluation," 2017, *arXiv:1706.04052*. [Online]. Available: https://arxiv.org/abs/1706.04052

[37] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 3357–3364.

[38] K. Zhang, N. Liu, X. Yuan, X. Guo, C. Gao, and Z. Zhao, "Fine-grained age estimation in the wild with attention LSTM networks," 2018, *arXiv:1805.10445*. [Online]. Available: https://arxiv.org/abs/1805.10445

[39] Z. Ma, D. Chang, J. Xie, Y. Ding, S. Wen, X. Li, Z. Si, and J. Guo, "Fine-grained vehicle classification with channel max pooling modified CNNs," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3224–3233, Apr. 2019.

CHUNXUE WU received the Ph.D. degree in control theory and control engineering from the China University of Mining and Technology, Beijing, China, in 2006. He is currently a Professor with the Computer Science and Engineering and Software Engineering Division, School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai, China. His research interests include, wireless sensor networks, distributed and embedded systems, wireless and mobile systems, and networked control systems.



BOBO JU is currently pursuing the master's degree in computer science and technology with the University of Shanghai for Science and Technology, Shanghai, China. He is engaged in big data and artificial intelligence research under the guidance of Prof. C. Wu.

YAN WU received the Ph.D. degree from Southern Illinois University Carbondale, with concentrations in environmental chemistry and ecotoxicology. He is currently a Postdoctoral Associate with the School of Public and Environmental Affairs, Indiana University Bloomington. His research involves elucidations of environmental fate of contaminants using chemical and computational techniques, predictions of their associated effects on wildlife and public health, and data processing and analysis in environmental related fields.

XIAO LIN received the Ph.D. degree in computer science from Shanghai Jiao Tong University, Shanghai, China. She engaged in Postdoctoral Research at the University of Shanghai for Science and Technology, Shanghai, China. She is currently an Associate Professor with the Department of Computer Science, Shanghai Normal University, Shanghai, China. Her research interests include image processing and computer vision.

NAIXUE XIONG received the Ph.D. degrees from Wuhan University (about sensor system engineering), and the Japan Advanced Institute of Science and Technology (about dependable sensor networks), respectively.

He was with Northeastern State University, Georgia State University, Wentworth Technology Institution, and Colorado Technical University (Full Professor about five years) about ten years. He is currently a Professor with the College of Intelligence and Computing, Tianjin University, China. He published over 300 international journal papers and over 100 international conference papers. Some of his works were published in the IEEE JSAC, the IEEE or ACM transactions, ACM Sigcomm Workshop, the IEEE INFOCOM, ICDCS, and IPDPS. His research interests include cloud computing, security and dependability, parallel and distributed computing, networks, and optimization theory.

Dr. Xiong is a Senior Member of the IEEE Computer Society. He received the Best Paper Award in the 10th IEEE International Conference on High Performance Computing and Communications (HPCC-08) and the Best Student Paper Award in the 28th North American Fuzzy Information Processing Society Annual Conference (NAFIPS2009). He has been a General Chair, Program Chair, Publicity Chair, PC Member, and OC Member of over 100 international conferences, and as a Reviewer of about 100 international journals, including the IEEE JSAC, the IEEE SMC (Park: A/B/C), the IEEE TRANSACTIONS ON COMMUNICATIONS, the IEEE TRANSACTIONS ON MOBILE COMPUTING, and the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. He is the Chair of Trusted Cloud Computing Task Force, the IEEE Computational Intelligence Society (CIS), and the Industry System Applications Technical Committee. He is serving as an Editor-in-Chief, an Associate Editor, or an Editor Member of over ten international journals (including an Associate Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN & CYBERNETICS: SYSTEMS and *Information Science*, an Editor-in-Chief of *Journal of Internet Technology* (JIT), and an Editor-in-Chief of *Journal of Parallel & Cloud Computing* (PCC)), and a Guest Editor of over ten international journals, including *Sensor* Journal, WINET, and MONET.

GUANGQUAN XU received the Ph.D. degree from the Tianjin Key Laboratory of Advanced Networking (TANK), College of Intelligence and Computing, Tianjin University, China, in 2008, where he is currently a Full Professor. He is a member of the CCF. His research interests include cyber security and trust management.

HONGYAN LI received the master's degree in computer science from Central China Normal University, China, 2005, and the Ph.D. degree in computer architecture from the Huazhong University of Science and Technology, China, in 2016. She is currently an Associate Professor with the Hubei University of Economics. Since 2017, she has been a Postdoctoral Researcher with the State Key Laboratory of Information Engineering in Surveying, Mapping, and Remote Sensing, Wuhan University. Her research interests include computer vision, big data, and machine learning.

XUEFENG LIANG is currently a Professor with the School of Information, Kyoto University, Japan. His research interests include computer vision, visual cognition and perception (psychology), and machine learning. He is a member of The American Association for the Advancement of Science (AAAS) and the Japan Chinese Scholars Artificial Intelligence Association. He is an Associate Editor of ICTACT *Journal on Image and Video Processing* (IJIVP) and *International Journal of Information Technology, Communications and Convergence* (IJITCC). He had served as a Guest Editor of the Special issue of *Sensors Mobile Sensor Computing: Theory and Applications* and Special issue of *Signal Processing Image Communication* (Elsevier) and The Deep Learning in Computational Photography.