IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 22, 2019, accepted May 11, 2019, date of publication May 24, 2019, date of current version June 5, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2918703

Path Planning via an Improved DQN-Based Learning Policy

LIANGHENG LV, SUNJIE ZHANG, DERUI DING, AND YONGXIONG WANG
Department of Control Science and Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China

Corresponding author: Sunjie Zhang (zhang_sunjie@126.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61573246, Grant 61603255, and Grant 61873169, in part by the Australian Research Council Discovery Project under Grant DP160103567, and in part by the Shanghai Twilight Program of China under Grant 18CG52.

ABSTRACT The path planning technology is an important part of navigation, which is the core of robotics research. Reinforcement learning is a fashionable algorithm that learns from experience by mimicking the process of human learning skills. When learning new skills, the comprehensive and diverse experience help to refine the grasp of new skills which are called as the depth and the breadth of experience. According to the path planning, this paper proposes an improved learning policy based on the different demand of the experience's depth and breadth in different learning stages, where the deep Q-networks calculated Q-value adopts the dense network framework. In the initial stage of learning, an experience value evaluation network is created to increase the proportion of deep experience to understand the environmental rules more quickly. When the path wandering phenomenon happens, the exploration of wandering point and other points are taken into account to improve the breadth of the experience pool by using parallel exploration structure. In addition, the network structure is improved by referring to the dense connection method, so the learning and expressive abilities of the network are improved to some extent. Finally, the experimental results show that our model has a certain improvement in convergence speed, planning success rate, and path accuracy. Under the same experimental conditions, the method of this paper is compared with the conventional intensive learning method via deep Q-networks. The results show that the indicators of this method are significantly higher.

INDEX TERMS Machine learning algorithms, path planning, neural network.

I. INTRODUCTION

In the field of artificial intelligence, it is a well-known and important issue that how to find the best path from the start point to the goal in a given grid environment. For a long time, many researchers have spent a lot of effort in path planning, and have proposed many algorithms for processing path search and optimization. More representative heuristic algorithms such as A* algorithm [1], [2], simulated annealing algorithm [3], [4], artificial potential field method [5], [6], group intelligent algorithm such as particle swarm algorithm [7], [8] and ant colony algorithm [8], [9]. With the deepening of research, the planning speed and the accuracy of path planning continue to increase, but these traditional algorithms always have shortcomings such as low real-time performance and easy to fall into local best.

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

With the advent of the artificial intelligence era, the environment facing the path planning field is becoming more and more complex, which requires the path planning algorithm to have the ability to respond quickly to complex environmental changes and flexible learning capabilities. The expressive power of traditional algorithms encounters bottlenecks and it is difficult to model dynamic complex problems. It is imperative that something be done to change the path planning algorithm framework. Combining deep learning with reinforcement learning [10]–[12] is a good method of autonomous learning [32]–[34]. The concept of ''reinforcement learning'' first appeared in 1954. This learning method transforms the sequence decision problem into a Markov model [13], and establishes the mapping between the environment state and the state-action value function through the interaction between the agent and the environment, and then obtains the optimal state-action value function to obtain the optimal action sequence.

With the development of time, dynamic programming [14], Q-learning [15], SARSA [16] and other reinforcement algorithms have been proposed, but these tabular reinforcement learning methods have obvious limitations in the size of state space and action space. In 2013, Mnih et al. have proposed the first successful Deep Q-network (DQN) [17] framework that combines deep learning with reinforcement learning. In DQN, the experience pool structure is used to disrupt the sample order to solve the problem that the experience gained from reinforcement learning is related in time. It improves the stability of the combination and achieved impressive performance. Several improvements based on DQN have proposed in 2015 [18]–[25], including duel network structure [18], complex empirical sampling policy [20] and advantage function. The DQN framework has a certain improvement in learning speed and value function estimation accuracy. Q-learning is very suitable for solving some discrete motion sequence decision problems. In recent years, some researchers have used Q-learning on path planning, and the effect is excellent. However, as mentioned above, the tabular reinforcement learning algorithm has limited capacity, this leads to the model needing to train each map and the model has no generalization performance. DQN can solve these problems very well, so the DQN framework has great potential in path planning.

The policy which is chosen is a very important factor influencing learning outcomes. Many educators advocate a method of learning from typical experiences and then refining knowledge through learning diverse experiences. From the field of artificial intelligence, some special samples can really speed up the learning, and the comprehensiveness of the sample can reduce the over-fitting and improve the accuracy of the model. The network structure determines the learning efficiency and expressive ability of the network. In 2017, Huang proposed a dense connection network (Dense Network) [30], [31], which uses channel-level feature short-circuit connections, effectively improving the feature reuse rate and reducing the gradient disappearance. A small amount of parameters yields surprising results.

Unlike traditional path planning methods that require modeling of the various constraints of the problem, map image data are only used as input in our path planning method. In addition, taking into account the generalization ability of the model, the input maps in each round are different. Under the above conditions, we hope that the agent can find the shortest path without collision. In order to accomplish the above functions, the deep reinforcement learning framework is used to improve the common problems in path planning. From the aspects of policy and network architecture, the main contribution of this paper is listed as follows:

(1) An experience value evaluation network is built. At the beginning of the training, when the network is in the pseudo-minded stage, the network is used to help Q network to gain more depth experience and help the model to quickly learn the environmental rules.

(2) A parallel exploration structure is created in order to utilize every step well. When path wandering phenomenon occurs, the exploration of wandering point and other points are taken into account to improve the breadth of the experience pool and increase the accuracy.

(3) The dense connection is added to increase the utilization of network features. The convolutional layer is used as the transition layer to reduce the dimension, at the same time to retain more high-dimensional information and position sensitivity.

In general, we propose a rapid learning policy that changes the probability distribution of experience in the skill learning process, so that the model can obtain more needed experience at different stages of learning, which improves learning efficiency. In the path planning application, we only need the map image as input instead of modeling the map. For the image input, we improve the network structure and enhance the expressive ability of the network. The remainder of the paper is structured as follows. The traditional reinforcement learning and DQN are introduced in Section II. In Section III, we propose an efficient learning policy based on the different demand of the depth and the breadth of experience in different learning stages, and furthermore, build a value evaluation network to control the depth of experience and speed up the learning process. Data simulation process and simulation results are provided in Section IV to demonstrate the effectiveness of our model. Section V concludes the paper with some discussion on future research directions.

II. RELATED WORK

A. REINFORCEMENT LEARNING

Compared with the ''open loop'' machine learning method based on existing static data, the reinforcement learning is a kind of ''closed loop'' method that learns from the experience with the environment. By interacting with the environment, a basic reinforcement learning framework is considered to learn how to maximize the benefits of a sequence decision problem. The problem can be represented by the Markov decision process (MDP) [13], which can be composed of 5-tuple $\langle S, A, P, R, \gamma \rangle$, where S is the state set, A is the set of finite actions, R is a finite set of the expected reward r_t, $\gamma \in [0, 1]$ is a discount factor, and P stands for the state transition probability and a simplified form of the conditional probability $P_a(s_t, s_{t+1})$ that the action a_t in state s_t is performed to achieve state s_{t+1} in time $t + 1$.

The purpose of reinforcement learning is to ultimately find an optimal sequence of actions $\pi^* = \{a_1^*, a_2^*, \cdots, a_t^*, \cdots\}$ in a given environment to maximize the cumulative reward of the agent. For the given action policy π, the cumulative reward and state value functions are defined to quantify the value of the state:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (1)$$

In the formula, G_t is called cumulative reward, which represents the sum of the discounts of the reward from the time step t until the end of the action sequence. $\gamma \in [0, 1]$ is a

discount factor that determines the trade-off between short-term and long-term gains. $\gamma = 1$ means that the agent treats the rewards equally from different time step away from itself.

Since the sequence of actions may be different in the same state, the cumulative reward from a certain state is an expected value rather than a certain value. But we can define a state value function to quantify the expected value of the cumulative reward in a state under a given policy, as follow.

$$V_\pi(s_t) = \mathbb{E}_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s = s_t\right\} \tag{2}$$

where $V_\pi(s_t)$ represents the expected cumulative reward of the agent starting from the state s_t under the policy π. Furthermore, considering the executing action a_t, we can change the state value function (2) into the state-action value function to describe the cumulative reward, as below:

$$q(s_t, a_t) = \mathbb{E}_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s = s_t, a = a_t\right\} \tag{3}$$

Obviously, the algorithm of reinforcement learning needs to get the maximum value of the state-action value function $q^*(s_t, a_t) = \max_{a\in A} q(s_t, a)$ in a given state. In other words, it is easy to get the optimal policy a_t^*, that is, $\text{argmax}_{a\in A} q^*(s_t, a)$.

For the problem of policy evaluation, it is generally to find each $q(s_t, a_t)$ and the optimal policy finally by updating the Q table. Note that the state-action value function in (3) indicates that the entire complete sequence of actions is required to update the value of the state-action value function. This means a lot of computational burden, and such an algorithm is undoubtedly inefficient. According to the Bellman equation, $q(s_t, a_t)$ can be rewritten as the following form:

$$q(s_t, a_t) = \mathbb{E}_\pi\left\{r_{t+1} + \gamma q(s_{t+1}, a_{t+1})|s = s_t, a = a_t\right\} \tag{4}$$

The above equation indicates that the state-action value function can be written by the state-action value of next state and the reward of the specified action. Obviously, according to (4), the Q value of the state can be updated every step, and the efficiency of the algorithm are higher. The value iterative algorithm is based on this, directly updating the table to estimate the optimal state-behavior value function:

$$q^*(s_t, a_t) = \mathbb{E}_\pi\left\{r_{t+1} + \gamma \max_{a_{t+1}\in A} q(s_{t+1}, a_{t+1})|s = s_t, a = a_t\right\} \tag{5}$$

B. DEEP Q-NETWORK

The traditional reinforcement learning algorithm learns the optimal policy by establishing a Q table and updating the Q table, but the reinforcement learning method based on the Q table has an inevitable capacity limitation. When the state space and the action space are large or continuous, the algorithm needs to occupy a large amount of memory or even can't express the problem. In 2013, Mnih et al. proposed the first framework DQN, which combines deep learning with reinforcement learning, to solve the problem of capacity limitation and sample correlation. In 2015 [17], a double network structure was proposed to solve the correlation between the state-action value function and the update target. The main achievements are as follows: (1) Propose to use deep convolutional neural network $q(s, a; \theta)$ to represent $q(s, a)$, avoiding the problem that Q table capacity is limited and each state-action value function needs training; (2) Propose the experience replay(ER) structure, solving the problem of time-correlation of samples and improving the stability of training; (3) Set up a separate target network to handle temporal difference (TD) targets, estimating the state-action value and the TD target and updating the weight.

DQN is an algorithm based on Q-learning. Q-learning updates the value function by time difference formula:

$$q(s_t, a_t) = q(s_t, a_t) + \alpha[r_t + \gamma \max_{a_{t+1}\in A} q(s_{t+1}, a_{t+1}) - q(s_t, a_t)] \tag{6}$$

where $q(s_t, a_t)$ is the state-action value function at the current moment, $q(s_{t+1}, a_{t+1})$ is the state-action value function at the next moment, and α is the update step size. After using the deep convolutional neural network, change to update the weight θ of Q_v:

$$\theta_{t+1} = \theta_t + \alpha[r_t + \gamma \max_{a_{t+1}\in A} q(s_{t+1}, a_{t+1}; \theta^{TD}) - q(s_t, a_t; \theta_t)] \times \nabla q(s_t, a_t; \theta_t) \tag{7}$$

where $r_t + \gamma \max_{a_{t+1}\in A} q(s_{t+1}, a_{t+1}; \theta^{TD})$ is the TD target. The network update its weight to let $q(s_t, a_t; \theta_t)$ fit the TD target. A separate target network is employed to represent the TD target. Since θ_t used in the calculation of the gradient is different from θ^{TD} used in the calculation of the TD target, it solves the problem of unstable training due to the correlation between samples. The $r_t + \gamma \max_{a_{t+1}\in A} q(s_{t+1}, a_{t+1}; \theta^{TD}) - q(s_t, a_t; \theta_t)$ in the equation is the loss, the network is trained by minimizing the loss, and finally $q(s_t, a_t)$ is estimated. The algorithm uses the stochastic gradient descent method to train the network, θ_t is updated every training iteration, and θ_t is assigned to θ^{TD} after every C iterations. The DQN algorithm is as shown in Algorithm 1, where *pre_train_step*, *decline_step*, μ, *mini_batch* and *terminal* will be defined in Section III.

C. DENSE NETWORK

Convolutional neural network (CNN) has become the mainstream method in the field of computer vision. In general, the deeper network is, the higher nonlinearity and fitting accuracy are. However the gradient of network training is transmitted from the back to the front, so the gradient received by the front layer gradually becomes smaller or even the training is stagnant, if the number of network layers reaches a certain limit. Dr. He proposed the Residual Network (ResNet) [26]–[29] in 2015. Short-circuiting some layers front and back strengthens the information connection between the front and back layers in order to make training of deeper network possible. In 2017, Huang employed the idea of the residual network to connect the front layer with all the layers after it (Dense Network [30], [31]), enhanced

Algorithm 1 DQN.

Initialization Initialize replay memory space D to capacity N, Initialize the Q network Q with random weights θ_0, Initialize the weights θ^{TD} of target network Q_t with weights θ_0. Initialize $t = 0$.

for $t < t_{max}$ **do**
  if $t \neq 1$ **then** $s_t = s_{t+1}$
  else get the initial observation s_t
  end if
  if $t <$ *pre_train_step* **then**
    select a random action a_t
  else
    if $\mu < \epsilon$ **then** select a random a_t
    else select $a_t = \text{argmax}_{a \in A} q(s_t, a; \theta_t)$
    end if
  end if
  Store experience $ex_t = (s_t, a_t, r_t, s_{t+1})$
  if $t <$ *decline_step* **then**
    ϵ decreases by a certain percentage
  end if
  if $t \geq$ *pre_train_step* **then**
    Sample *mini_batch* in D and calculate y_i:

$$y_i = \begin{cases} r_i, & \text{if } terminal \\ r_i + \gamma \max_{a_{i+1} \in A} q_t(s_{i+1}, a_{i+1}; \theta^{TD}), & \text{otherwise} \end{cases}$$

    Calculate the loss $(q(s_i, a_i; \theta_t) - y_i)^2$
    Train and update Q network's weights θ_{t+1}
    Every C step copy θ_{t+1} to θ^{TD}
  end if
end for

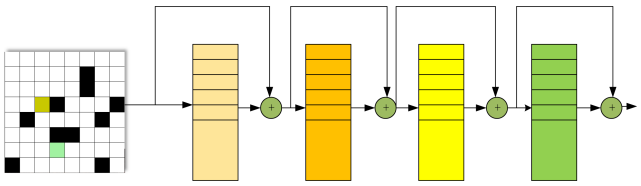

FIGURE 1. The structure of dense connection.

the feature reuse from the propagation and utilization of features, and achieved better performance with fewer parameters. As shown in Figure 1, an L-layer network ($L = 4$) is supposed. Each layer has a nonlinear conversion $H_l(\cdot)$, and the output of the $l-1$ layer is x_{l-1}.

Then, in the Dense Network, the output expression of the l-th layer is shown mathematically:

$$x_l = H_l([x_0, x_1, \cdots, x_{l-1}]) \tag{8}$$

where $[x_0, x_1, \cdots, x_{l-1}]$ is the splicing of the output feature map produced by the $0, 1, 2, \cdots, l-1$ layers in the dense block, which is a tensor. The advantage of this connection is that the model is more compact, information can be transferred to deeper layers, enhance the connection of features between the layers, and mitigate gradient disappearance. The premise of feature graph splicing is that the dimension of the feature graph is the same, so the Dense Network is divided into dense blocks and transition layers structure. The convolution with one step size is used in the dense blocks to keep the feature map size unchanged. The convolution uses multiple small convolution kernels to obtain higher nonlinearity, and the transition layer uses average or maximum pooling to reduce the dimension of the feature map.

III. PATH PLANNING BASED ON EFFICIENT LEARNING POLICY DQN COMBINED WITH DENSE NETWORK

In the context of path planning, this paper develops an improved in Policy and Network Deep Q-network (PN-DQN) model. We proposes some policies such as action experience value evaluation network, parallel greedy random exploration structure and merges the connection method of dense connection, aiming to improve the path planning speed and accuracy. This chapter will introduce our algorithm from the problem statement, the learning settings of the model (such as environment observation, action space, reward design, and policy setting) and network architecture, then carefully describe the algorithm flow and training process.

A. PROBLEM STATEMENT

The aim of our model is to find an optimal path from the start point to the end point in a randomly generated map without collision. It is considered that the agent moves in a four-connected environment consisting of passable and non-passable trellis. Given the start point s and the goal g which are connectable, the task of the agent is to find a feasible sequence of actions from s to g. It is also the policy $\pi(s, g)$.

Considering the general path planning problem and assuming that the map M, the start point s and the goal g are known, we use $E(M, s, g)$ to represent the environment. The traditional path planning algorithm deals with these problems by constraining conditions and problems. Mathematical modeling is transformed into searching optimization or solving energy optimization problem. When $E(M, s, g)$ changes, it needs to be solved again. It is also difficult to model $E(M, s, g)$. Our model mimics the process of human learning skills, which finds the optimal path by learning the rules of the environment. Even if a given obstacle or goal changes, the model does not need to be retrained because our rules are universal. Taking the picture of $E(M, s, g)$ as input, the agent selects the optimal policy to achieve the maximum benefit by observing its position, obstacle position and goal position. In order for the trained model to have good generalization performance, a new $E(M, s, g)$ will be randomly generated in each episode.

In the process of human learning new skills, typical cases tend to be more profound and clearer than the general experience. Furthermore, it is important for the refinement of skills to accumulate comprehensive and multi-faceted experience called as the depth and the breadth of experience. So we can improve the learning policy based on the depth and the

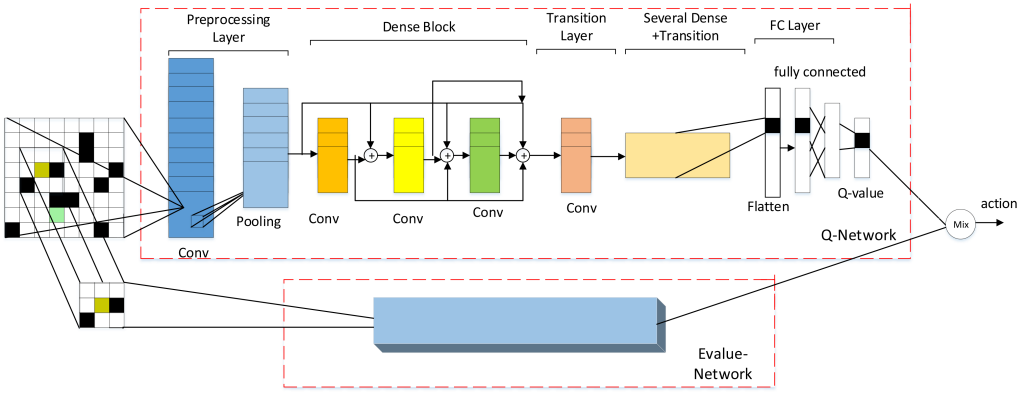

FIGURE 2. The structure of PN-DQN model.

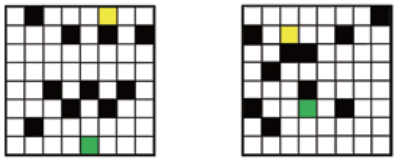

FIGURE 3. Two different observations.

breadth of experience in different stages of learning in two sides:

1. An action experience value evaluation network is built that increases the proportion of special experience (at obstacles or reaching the end) at the beginning of training. It is helpful for the model to learn environmental rules faster;

2. A parallel exploration structure is created. If path wandering phenomenon occurs in training, the learning policy will continue to explore the wandering point and take into account other points on the map, in order to obtain more diverse experience and help the model to master the skills in detail.

Combined with the efficient learning policy, we improve the network structure and get PN-DQN model. The main model of PN-DQN is shown in Figure 2. In this figure, we take the picture of $E(M, s, g)$ as input. The Q network above the figure is responsible for estimating the value of the state-action. The experience value evaluation network below the figure to take a part of evaluating the value of the experience of each action. For faster training speed and higher accuracy, the Q network is combined with the dense connection to improve the extraction and propagation of picture features. On the other hand, the value evaluation network uses a convolutional neural network with a simple structure. Ultimately, the model selects the actions performed by taking into account the output of the two networks.

B. LEARNING SETTINGS

1) ENVIRONMENT OBSERVATION

Figure 3 shows two different environment observations. An observation consists of background, obstacle, current point and goal. According to the image information of the environment, an RGB pixel matrix of 80∗80∗3 is formed, and then perform gray-scale processing on RGB image matrix to obtain 80∗80 gray matrix. In general, the gray matrix includes four types of image values. By preprocessing, four types of pixel values are rewritten into a matrix $[P_b, P_o, p_c, p_g]$ consisting of background pixel set P_b, obstacle pixel set P_o, current point pixel p_c and goal pixel p_g. The purpose of our preprocessing is to more accurately distinguish between different objects and get a more manageable observation matrix.

2) ACTION SPACE

There are two general kinds of action space in the mesh path planning task, which are four neighborhoods and eight neighborhoods. These definitions of action space can control the change of the current location. The experiments in this paper all use four neighborhoods, because the research goal is to get the best path not the motion plan. In the experiment, the speed will be ruled as that one unit distance can be moved per time step.

3) REWARD DESIGN

The reward is the only feedback that the model can get from the environment, and it is the learning orientation of the model. The reward determines the skills that the model learns and the efficiency of the model. Therefore, a good reward design should be concise and fully reflect the designer's desire to implement the model. In our tasks, reward design focuses on two aspects: reaching the goal and avoiding obstacles. Based on these requirements, the reward function is defined as a sparse form:

$$r_t = \begin{cases} r^{reach}, & \text{if } p_c = p_g \\ r^{crash}, & \text{if } p_c \in P_o \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

which divides the reward function into three parts according to the difference of the arrival point at the next moment. Combined with the optimal function Bellman equation (5), it can be seen that the value of the action to reach the goal

is r^{reach}, and the value of the action that hits the obstacle point is r^{crash}. We generally give r^{reach} a positive value to encourage the model to find the goal, give r^{crash} a negative values to punish collision behavior. The value of normal action decreases with the increase of the distance between the current point and the goal, and γ needs to take less than 1 to promote the agent to the goal in this experiment.

4) POLICY SETTING

DQN generally uses ϵ-greedy policy to balance the exploration and utilization of models, such as:

$$\pi(s_t) = \begin{cases} \text{argmax}_{a \in A} q(s_t, a), & \text{if } \mu \leq \epsilon \\ \tilde{a}, & \text{otherwise} \end{cases} \tag{10}$$

where μ is the random value generated from [0, 1] per round, ϵ is the exploration rate and $\tilde{a}$ is a random action.

On the basis of retaining certain randomness, combined with the application characteristics of grid path planning, we make the following improvements to the policy:

a: POLICY OF THE DEPTH OF EXPERIENCE

In order to gain more special experience in the early stage, we create an experience value evaluation network. The evaluation network only considers a rectangle with the current point as the center of the eight neighborhoods, and evaluates how valuable the experience of choosing a particular action is. The t-th loss of evaluation network E is defined as:

$$L_t(\theta_t) = \mathbb{E}_{s,a}\left\{((1 + |r_t|) - e(s_t, a_t; \theta_t^E))^2\right\} \tag{11}$$

where the value evaluation function $e(s_t, a_t; \theta_t^E)$ gradually progresses to $1 + |r_t|$ through training and θ^E is the weight of evaluation network E. Combined with (9), it can be estimated that the value of $e(s_t, a_t; \theta_t^E)$ will converge to:

$$e(s_t, a_t; \theta_t^E) = \begin{cases} 1 + |r^{reach}|, & \text{if } p_c = p_g \\ 1 + |r^{crash}|, & \text{if } p_c \in P_o \\ 1, & \text{otherwise} \end{cases} \tag{12}$$

The value evaluation network E completes the training in the pre-train stage before the train of network Q, and then help to select action:

$$a_t = \text{argmax}_{a \in A} q(s_t, a; \theta_t) e(s_t, a; \theta_t^E) \tag{13}$$

Remark 1: According to (12), the model selects an action based on the product of the value evaluation function and the state-behavior value estimation function. In the initial stage of Q network training, due to the maximum operation in (5), the state-action value function gives a positive difference estimate for each action. At this time, $e(s_t, a_t; \theta_t^E)$ encourages the model to choose the action of reach or crash in the next step, increasing the proportion of special experience. When $q(s_t, a_t; \theta_t)$ begins to correctly identify obstacles, $e(s_t, a_t; \theta_t^E)$ can suppress collisions, encourage agents to explore more locations, increase the diversity of experience, and learn skills carefully. Moreover, in the later exploration, when the model misjudges the obstacle $q(s_t, a_t; \theta_t) > 0$, it encourages to get the experience of the misjudges place.

b: POLICY OF THE BREADTH OF EXPERIENCE

In order to make better use of each step, the model creates a parallel structure for the path wandering phenomenon (left right left right, or up down up down) that appears during the training process, as shown in Figure 4. The model selects action to maximize $q(s_t, a; \theta_t)e(s_t, a; \theta_t^E)$ under normal conditions. When the path wandering phenomenon occurs, the parallel structure will be triggered. The parallel structure continues to explore the rest of the map with the greedy random policy, simultaneously continues to gain the experience of wandering point. The greedy policy randomly selects actions with a certain probability, or greedily chooses to move the current point closer to the goal without considering obstacles. We don't recommend direct forced out of the wandering point, because the phenomenon of wandering shows that the model lacks of that point's understanding, and the experience is very important. Wandering processing as shown in the right part of Figure 4, we extract the experience of the two steps before that points $ex_{t-1} = (s_{t-1}, a_{t-1}, r_{t-1}, s_t)$ and $ex_t = (s_t, a_t, r_t, s_{t+1})$, and the *current_step* in the current map. The model interacts with the environment through greedy random policy and also judges whether the network weights can jump out of the wandering point after update. If the model can identify that point, or if the number of steps reaches the preset maximum exploration steps for a single map, then the structure is ended. If not, it continue to add experience of wandering point. Our general idea is to take into account the wandering experience gained and the exploration of other locations on the map.

c: POLICY OF AVOIDING INCORRECT EVALUATION

In order to save time resource, we usually set a maximum number of steps as *max_step* that the agent can move in each episode. This assumption brings the problem of incorrect evaluation of the value function. In Algorithm 1:

$$y_i = \begin{cases} r_i, & \text{if } termial \\ r_i + \gamma \max_{a_{i+1} \in A} q_t(s_{i+1}, a_{i+1}; \theta^{TD}), & \text{otherwise} \end{cases} \tag{14}$$

where *termial* means $p_c = p_g$ or $p_c \in P_o$ or *current_step* = *max_step*. If there is no collision or reach the target, the value function should be $r_i + \gamma \max_{a_{i+1} \in A} q_t(s_{i+1}, a_{i+1}; \theta^{TD})$. However, when *current_step* is exactly equal to *max_step*, the model estimate function is r_i. This causes a large loss in the evaluation of the value function, which in turn leads to unstable training of the model. So in the experiment we abandoned the experience of *current_step* reaching *max_step*.

Remark 2: The *terminal* in the figure represents three cases: 1. *current_step* is equal to *max_step* 2. collision occurs 3. reaching the target point. The *terminal_w* in the figure indicates that the model successfully identifies the defect, that is, the action selected under the current weight

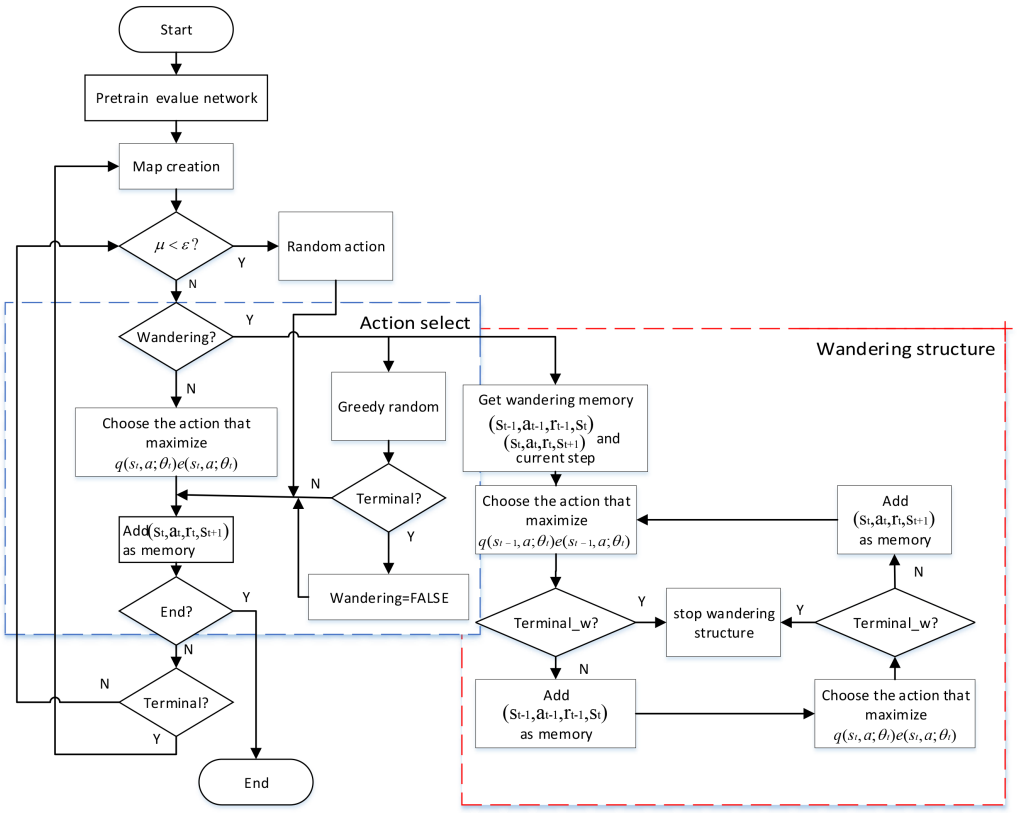

FIGURE 4. The algorithm flow charts of PN-DQN.

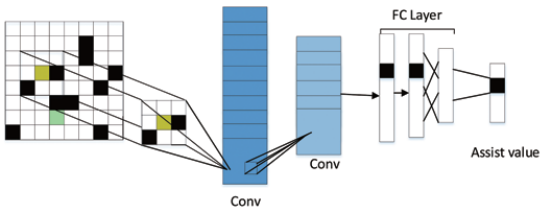

FIGURE 5. The structure of value evaluation network.

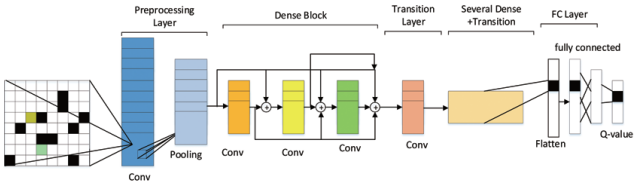

FIGURE 6. The structure of Q network.

is different from the action in the experience, and the *END* in the figure indicates that t reaches t_{max}.

C. MODEL AND NETWORK ARCHITECTURE

To successfully complete the navigation task, we propose a learning model PN-DQN which is suitable for the current task. The model is shown in Figure 2. The model consists of a value evaluation network E below the figure and the deep Q network Q above the figure and the target network Q_t with the same structure. The value evaluation network is shown in Figure 5.

The value evaluation network consists of convolution layers and fully connected layers. All convolution layers in this paper consist of convolution and batch normalization, extracting features, changing dimensions and reducing the possibility of over-fitting. The activation function uses ReLU to reduce the gradient disappearance and speed up the training. Using the same padding mode, the relationship of the convolution layer input feature map size W_{in}, output feature map size W_{out} and the stride S can be given by:

$$W_{out} = \frac{W_{in}}{S} \tag{15}$$

The Q network structure is shown in Figure 6. The network consists of a pre-processing layer, dense blocks and a fully-connected layer. The input of the network is the gray matrix of $80*80*4$. The first layer is the convolution layer, uses the ReLU function as activation function, the convolution kernel size is $8*8$, and the step size is 4, which reduces the image dimension, reduces subsequent calculations, and extracts features. Behind the convolution layer is an overlapping pooling layer of $2*2$, which maintains the feature map size and increases the generalization performance of the model to avoid over-fitting. Then there are three dense blocks and transition layers,the *growth rate* is 8, 16, and 16 respectively, and the *bottleneck* takes 2, which determines the output of the $3*3$ convolutional layer. The $1*1$ convolution layer has

TABLE 1. The parameters in Q network.

Layer	Input Size	Parameter
Input layer	80*80*4	
Conv1	20*20*32	8*8conv, stride4
Pooling	20*20*32	2*2avepooling, stride1
Dense block 1	20*20*80	$\begin{bmatrix} 1*1\text{conv} \\ 3*3\text{conv} \end{bmatrix} * 3$
Transition layer 1	10*10*40	$\begin{bmatrix} 1*1\text{conv, stride1} \\ 3*3\text{conv, stride2} \end{bmatrix}$
Dense block 2	10*10*136	$\begin{bmatrix} 1*1\text{conv} \\ 3*3\text{conv} \end{bmatrix} * 3$
Transition layer 2	5*5*68	$\begin{bmatrix} 1*1\text{conv, stride1} \\ 3*3\text{conv, stride2} \end{bmatrix}$
Dense block 3	5*5*164	$\begin{bmatrix} 1*1\text{conv} \\ 3*3\text{conv} \end{bmatrix} * 3$
Transition layer 3	5*5*82	1*1conv, stride1
FC layer	256*1	Fullconnected
Output layer	4*1	

the function of integrating features and reducing the amount of subsequent calculations. The number of output channels is *bottleneck* $*$ *growth rate*. Dense blocks use dense connections and use multiple small convolution kernels to improve feature propagation and reuse, as well as increase nonlinearity. The pooling layer is discarded in the transition layer, and the convolution layer is employed to reduce the dimension. The main purpose is to retain more high-dimensional features and location information. The transition layer has a ratio of input to output channels of 2 : 1, which compresses the features to make the network lighter. The third part is the fully connected layer, which integrates the features and outputs the state-action value for four actions. The specific parameters of the network are shown in Table 1.

To train the model, we calculate the loss and mean square error, and update the network parameters with (7). The algorithm flow and pseudo code are as follows:

Remark 3: As shown in Algorithm 2, we first initialize the environment, experience pool space, value network, target network, value evaluation network, etc. Set a *pre_train_step* to get some experience through random actions to store in the experience pool and complete the training of the value evaluation network in the pre-training stage. The training phase sets a ϵ that decreases as the number of steps increases, and uses the ϵ-greedy policy to determine whether the current step is random exploration or exploitation (13). If the path wandering phenomenon occurs, the exploitation will adopt the greedy random policy instead, and the parallel structure as shown in Figure 4 is taken into consideration, taking into account the exploration of other points and the acquisition of the wandering experience. It should be noted that the evaluation of the network weights in the form of θ_t^E is only to distinguish between moments and unify other weights. In fact, both θ_t^E and θ_t do not change with time t, and only during their training rounds will they update based on the calculated gradient. During training, draw *mini_batch* experiences from the experience pool per step and train the Q network. Finally, the parameters of the Q network Q are copied to the target network every C steps.

Algorithm 2 PN-DQN.

Initialization Initialize replay memory space D to capacity N, Initialize the Q network Q with random weights θ_0, Initialize the target network $Q_t's$ weights θ^{TD} with weights θ_0, Initialize the value evaluation network E with random weights θ^E. Initialize $t = 0$.

for $t < t_{max}$ **do**
  if $t \neq 1$ **then** $s_t = s_{t+1}$
  else get the initial observation s_t
  end if
  if $t <$ *pre_train_step* **then**
    Select a random action a_t
    if $t >$ *mini_batch* **then**
      Sample *mini_batch* in D and calculate $y_i = r_i$
      Calculate the loss $(e(s_i, a_i; \theta_t^E) - y_i)^2$
      Train and update E network's weights θ_{t+1}^E
    end if
  else
    if $\mu < \epsilon$ **then**
      select a random action a_t
    else
      select $a_t = \max_{a \in A} q(s_t, a; \theta_t) e(s_t, a; \theta_t^E)$
    end if
  end if
  Store experience $ex_t = (s_t, a_t, r_t, s_{t+1})$
  if $t <$ *decline_step* **then**
    ϵ decreases by a certain percentage
  end if
  if $t \geq$ *pre_train_step* **then**
    Sample *mini_batch* in D and calculate y_i:

$$y_i = \begin{cases} r_i, \\ \qquad \text{if } terminal \\ r_i + \gamma \max_{a_{i+1} \in A} q_t(s_{i+1}, a_{i+1}; \theta^{TD}), \\ \qquad \text{otherwise} \end{cases}$$

    Calculate the loss $(q(s_i, a_i; \theta_t) - y_i)^2$
    Train and update Q network's weights θ_{t+1}
    Every C step copy θ_{t+1} to θ^{TD}
  end if
end for

IV. TEST AND RESULT ANALYSIS

The evaluation of the model path planning ability is carried out in a grid environment. The goal of the model is to find

TABLE 2. The parameters in algorithm 2.

Parameter	Value	Meaning
$mini-batch$	32	Sample size each step
γ	0.9	Discount factor
N	150000	Capacity of ER buffer
C	5	Update frequency
pre_train_step	50000	The number of steps to pre-training and fill ER buffer
$decline\_step$	30000	The number of steps ϵ change from 1 to 0.01
ϵ	$[0.01, 1]$	Probability to select random action

the optimal sequence of actions from the starting point to the target point without collision. In order to verify that the model has the ability to adapt to different environments, map $E(M, s, g)$ is regenerated every episode. The model has only a limited number of steps *max_step* in each episode, and the episode ends early when the agent hits an obstacle or reaches the target. The experiment is divided into two groups with the size of $5 * 5$ and $8 * 8$. For each group PN-DQN compared by DQN, P-DQN (only change policy), N-DQN (only change network) is employed to show the advantage of our idea.

The policies and networks used in the experiments have been introduced in the previous chapter. Table 2 provide some of the basic parameters used in the experiments:

The size of Mini-batch is set to 32, and the parameters of the value network are copied to the target network every 5 steps. We set the value γ to 0.9 instead of the common 0.99. Since each round $E(M, s, g)$ is different, the minimum value is set to 0.01 to more intuitively observe the model performance.

Remark 4: Mini-batch determines the direction in which the gradient falls. Too small may cause over-fitting, while too large will slow down the convergence, and the memory size of the computer is also a limit. Since the reward function (9) stipulates that the return is 0 under normal conditions, we use the discount factor to distinguish the action different from their distance beyond the target point more clearly in the magnitude of the Q value. N should be appropriately larger to allow the model to learn more experience in the state to prevent overfitting and local optimization. A small C can keep the loss calculation real-time and effective, but too small will lead to training shocks.

To evaluate the performance of the model we define the following metrics:

1. Success rate: the ratio of the number of rounds that successfully find the target point to the total number of rounds;

2. Accuracy: the ratio of the shortest path steps to the number of steps used in a successful round;

3. Loss: the loss during training.

The premise that the model ultimately achieves good training results is that the value evaluation network can accurately identify different points. The training process of the $5*5$ map consists of 1450000 exploration steps and 50000 pre-training steps (no update network), and the maximum number of steps per episode is 15 steps. Similarly, the training process of the

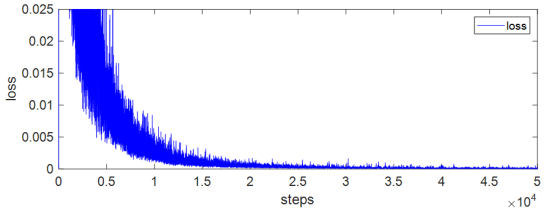

FIGURE 7. The loss of value evaluation network in the $5 * 5$ map.

$8 * 8$ map consists of 1850000 exploration steps and 150, 000 pre-training steps. The maximum number of steps per episode is 25 steps. After the pre-training step is completed, observe whether the empirical value network converges, and then train the Q network.

Figure 7 is the loss of the value evaluation network in 50, 000 steps in the $5 * 5$ map. It can be seen that the loss decreases with the increase of the number of training steps, which indicates that the parameters of the A network are optimized by the gradient descent method. The model converges around 25, 000 steps, and the value evaluation network can accurately identify different points.

Figure 8 is the success rate and the accuracy of the training process. The results show that as the training progresses, the models become familiar with the environment, and the success rate and the accuracy are increasing. Figure 8(a) and Figure 8(b) are the rates of change in success. From a policy point of view, P-DQN and PN-DQN learn faster than DQN and N-DQN because they get more deep experience in the early stage of training. After the stability, their success rate are higher than DQN (and N-DQN). From the perspective of network architecture, N-DQN and PN-DQN benefit from dense connection and efficient feature learning and utilization, so they have stronger learning ability and learning speed. In Figure 8(b), the P-DQN policy is more suitable for learning, so the success rate of the previous period is higher than that of the N-DQN. However, due to the limitation of the network architecture, the success rate is exceeded by the N-DQN after convergence. The PN-DQN model is excellent in both convergence speed and success rate.

Figure 8(c) and Figure 8(d) are accuracy, their trends are similar to success rates. It is worth noting that the accuracy is affected more by the policy than the success rate. It can be seen that the policy greatly improves the accuracy rate while slightly increasing the success rate of the model, showing the superiority of our policy. In general, both the improvement of the policy and the network structure have a certain improvement on the learning speed and the success rate and also the accuracy. The PN-DQN model has excellent performance in all these aspects.

Figure 9 shows the success rate and the accuracy of the test after the training of the model. The test has a total of 200000 steps. Figure 9(a) and Figure 9(b) are success rates. Through the comparison of the four models in each graph, it can be seen that both the policy and the network have improved the success rate of navigation. Comparing the two

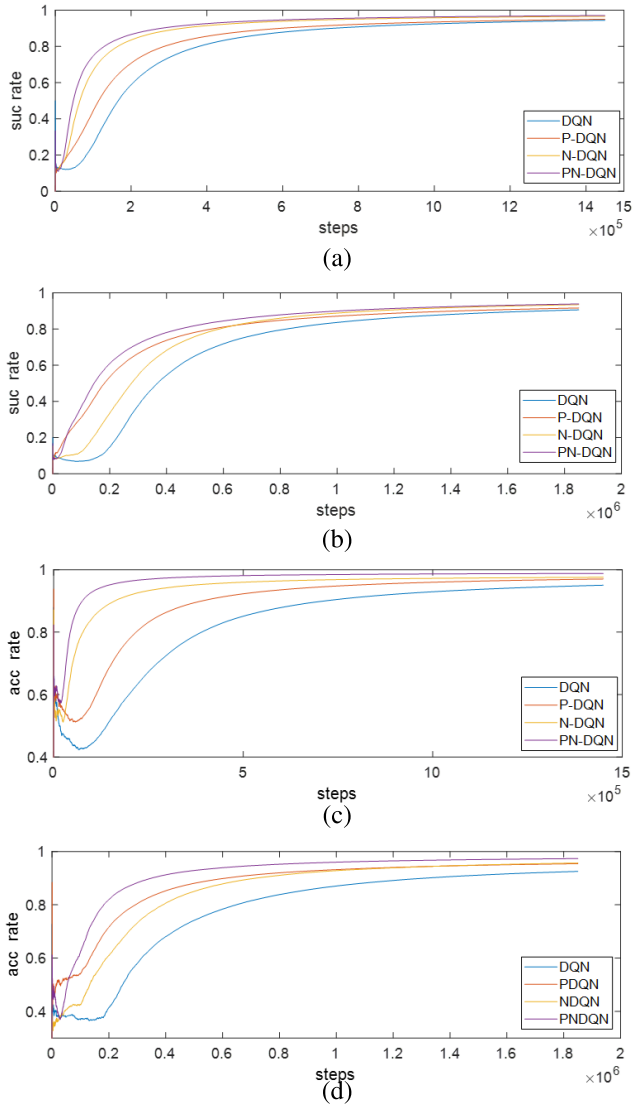

FIGURE 8. The success rate and the accuracy of training process. (a) The success rate in 5 ∗ 5 map. (b) The success rate in 8 ∗ 8 map. (c) The accuracy in 5 ∗ 5 map. (d) The accuracy in 8 ∗ 8 map.

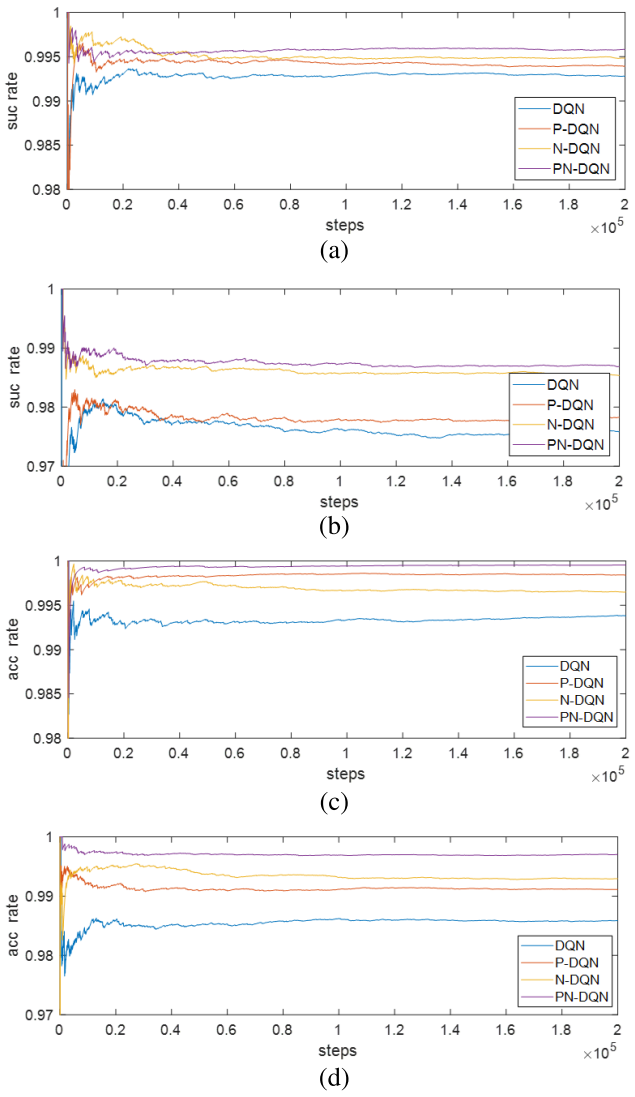

FIGURE 9. The success rate and the accuracy of testing process. (a) The success rate in 5 ∗ 5 map. (b) The success rate in 8 ∗ 8 map. (c) The accuracy in 5 ∗ 5 map. (d) The accuracy in 8 ∗ 8 map.

graphs Figure 9(a) and Figure 9(b), Figure 9(b) shows obvious stratification according to the network used by the model. The more complicated the environment is, the more obvious the learning advantage of dense connections is. In terms of accuracy, it can be seen from Figure 9(c) and Figure 9(d) that efficient learning policies have a great improvement in accuracy. In Figure 9(c), P-DQN even achieve better performance than N-DQN relies on policy advantages. The effect reflects the efficient learning policy is more conducive to the model to master the path planning skills. And Figure 9(d) once again shows us the learning advantage of dense connections. In Figure 9, the PN-DQN model is far ahead of the DQN model in both accuracy and success rate.

Deep experience can bring deeper impact (expressed in the learning of the appropriate loss function), speed up the learning speed of the early period, and breadth experience can improve the diversity of the sample and improve the learning accuracy. In general, the expressive ability of the network model determines the success rate and accuracy of the model, and our policy changes the distribution of the experience we have acquired, which is more suitable for skill learning, and has a higher success rate and accuracy, especially accuracy.

Figure 10 shows the loss during the training. Comparing P-DQN with DQN and PN-DQN with N-DQN respectively, the model with efficient learning policy drops faster under the same conditions, because our policy bring some of the more diversified samples that allows us to get steeper gradients in the early stage. This leads us to have more rounds to get more diverse experience, which make the training process more stable. Compared with N-DQN and DQN, the loss curves of PN-DQN and P-DQN are more stable and smaller, showing the excellent learning characteristics of our policy. In terms of

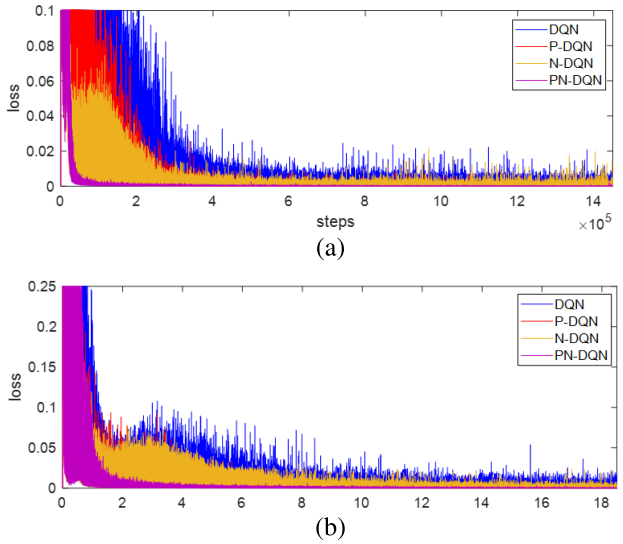

FIGURE 10. The loss of training process. (a) The loss in 5 ∗ 5 map. (b) The loss in 8 ∗ 8 map.

TABLE 3. Result.

	5*5		
	success rate	accuracy	loss
DQN	0.9929	0.9937	3.4507e-4
P-DQN	0.9940	0.9985	1.1275e-4
N-DQN	0.9949	0.9966	1.4953e-4
P-NDQN	**0.9958**	**0.9995**	**4.0800e-5**
	8*8		
	success rate	accuracy	loss
DQN	0.9757	0.9858	7.2958e-4
P-DQN	0.9775	0.9912	2.6321e-4
N-DQN	0.9857	0.9930	2.5677e-4
P-NDQN	**0.9870**	**0.9970**	**1.0741e-4**

network structure, the loss curve graphs of N-DQN and DQN are very similar, but due to the higher learning efficiency and higher fitting ability of our model, N-DQN has faster convergence speed and smaller loss. Overall, our PN-DQN model is more stable, faster, and more accurate than the traditional DQN model.

Table 3 is the performance

index of each model in this experiment, taking the average of the success rate and accuracy rate of the last 50, 000 steps in the test, and the loss is the average value of the last 50000 step error in the training. The probability of finding the path in our $5 * 5$ and $8 * 8$ variable map environment is 99.58% and 98.70%, the path accuracy is 99.95% and 99.70%, and the state-action values estimation errors are $4.0800e - 5$ and $1.0741e - 4$ respectively. All aspects have been improved.

V. CONCLUSIONS

This paper has proposed an efficient PN-DQN algorithm. It has been found that when people learn new skills, they like to learn typical cases to master the general framework of skills, and enrich their understanding of skills through diversified experience. In the experience pool, the proportion of deep experience and breadth experience has been decided to change depending on the type of experience required at the different stages of the learning model. In the grid path planning experiment, a value evaluation network has been built to control the depth of experience and speed up the learning process. For the path wandering phenomenon, a parallel structure has been created that it makes more efficient use of time steps and increases the breadth of experience. In addition, we have incorporated a dense connection to enhance the learning ability of the network model. In the end, simulation experiments have shown that our algorithm is much better than traditional DQN algorithms in terms of learning speed, path planning success rate and path accuracy. We believe that as long as the State space is discrete, our learning policy can speed up learning. our further research topics include the improvement of our algorithm and its application to obstacle avoidance and navigation of aircraft.

REFERENCES

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 37, pp. 28–29, 1972.

[2] R. C. Holte, M. B. Perez, R. M. Zimmer, and A. J. MacDonald, "Hierarchical A*: Searching abstraction hierarchies efficiently," in *Proc. 13th Nat. Conf. Artif. Intell.*, 1996, pp. 1–6.

[3] M. Steinbrunn, G. Moerkotte, and A. Kemper, "Heuristic and randomized optimization for the join ordering problem," *VLDB J.*, vol. 6, no. 3, pp. 191–208, 1997.

[4] P. Mishra and M. H. Eich, "Join processing in relational databases," *ACM Comput. Surv.*, vol. 24, no. 1, pp. 63–113, 1992.

[5] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots in cluttered environments," in *Proc. IEEE Int. Conf. Robot. Automat.*, Cincinnati, OH, USA, May 1990, pp. 572–577.

[6] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, St. Louis, MO, USA, Mar. 1985, pp. 500–505.

[7] J. Kennedy, "The particle swarm: Social adaptation of knowledge," in *Proc. IEEE Int. Conf. Evol. Comput. (ICEC)*, Indianapolis, IN, USA, Apr. 1997, pp. 303–308.

[8] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 26, no. 1, pp. 29–41, Feb. 1996.

[9] X. Wu, G. Wei, Y. Song, and X. Huang, "Improved ACO-based path planning with rollback and death strategies," *Syst. Sci. Control Eng.*, vol. 6, no. 1, pp. 102–107, 2018.

[10] C. Szepesvri, "Algorithms for reinforcement learning," in *Wiley Encyclopedia of Operations Research and Management Science*. 2011.

[11] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: An introduction," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 285–286, 2005.

[12] L. Cui, X. Wang, and Y. Zhang, "Reinforcement learning-based asymptotic cooperative tracking of a class multi-agent dynamic systems using neural networks," *Neurocomputing*, vol. 171, pp. 220–229, Jan. 2016.

[13] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1238–1274, 2013.

[14] L. R. Busoniu, R. Babuska, and B. D. Schutter, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Boca Raton, FL, USA: CRC Press, 2010.

[15] J. Ho, D. W. Engels, and S. E. Sarma, "HiQ: A hierarchical Q-learning algorithm to solve the reader collision problem," in *Proc. Int. Symp. Appl. Internet Workshops*, Phoenix, AZ, USA, Jan. 2006, p. 4 and 91.

[16] S.-L. Chen and Y.-M. Wei, "Least-squares SARSA (Lambda) algorithms for reinforcement learning," in *Proc. Int. Conf. Natural Comput.*, Jinan, China, Oct. 2008, pp. 632–636.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 581, no. 7549, pp. 529–533, 2015.

[18] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. 33nd Int. Conf. Mach. Learn.*, New York, NY, USA, 2016, pp. 1995–2003.
[19] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. Int. Conf. Learn. Represent.*, San Juan, Puerto Rico, 2015.
[20] T. Schaul, J. Quan, and I. Antonoglou, "Prioritized experience replay," in *Proc. Int. Conf. Learn. Represent.*, San Juan, Puerto Rico, 2015.
[21] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Proc. Neural Inf. Process. Syst. Conf.*, Barcelona, Spain, 2016, pp. 4026–4034.
[22] B. C. Stadie, S. Levine, and P. Abbeel, "Incentivizing exploration in reinforcement learning with deep predictive models," 2015, *arXiv:1507.00814*. [Online]. Available: https://arxiv.org/abs/1507.00814
[23] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn.*, New York, NY, USA, 2016, pp. 1928–1937.
[24] O. Anschel, N. Baram, and N. Shimkin, "Averaged-DQN: Variance reduction and stabilization for deep reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn.*, New York, NY, USA, 2016, pp. 176–185.
[25] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," 2015, *arXiv:1507.04888*. [Online]. Available: arXiv:1507.04888
[26] K. He, X. Zhang, S. Ren, and J. Su, "deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.
[27] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. Int. Conf. Comput. Vis.*, Santiago, Chile, Dec. 2015, pp. 1026–1034.
[28] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.*, Amsterdam, The Netherlands, 2016, pp. 630–645.
[29] G. Larsson, M. Maire, and G. Shakhnarovich, "FractalNet: Ultra-deep neural networks without residuals," 2016, *arXiv:1605.07648*. [Online]. Available: https://arxiv.org/abs/1605.07648
[30] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Honolulu, HI, USA, Jul. 2017, pp. 4700–4708.
[31] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, Lille, France, 2015, pp. 448–456.
[32] A. Hussein, E. Elyan, and M. M. Gaber, "Deep imitation learning for 3D navigation tasks," *Neural Comput. Appl.*, vol. 29, no. 7, pp. 389–404, 2018.
[33] J. Xu, Q. Liu, H. Guo, A. Kageza, S. AlQarni, and S. Wu, "Shared multi-task imitation learning for indoor self-navigation," in *Proc. IEEE Global Commun. Conf.*, Abu Dhabi, United Arab Emirates, Dec. 2018, pp. 1–7.
[34] M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto, "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 4423–4430, Oct. 2018.

LIANGHENG LV received the B.Sc. degree in automation from the University of Shanghai for Science and Technology, Shanghai, China, in 2017. He is currently pursuing the M.Sc. degree in control science and engineering. His research interests include the intelligence optimization algorithms, the machine learning algorithms, and neural networks.

SUNJIE ZHANG received the B.Sc. degree in applied mathematics and the Ph.D. degree in control theory and control engineering from Donghua University, Shanghai, China, in 2010 and 2015, respectively. He is currently an Associate Professor with the Department of Control Science and Engineering, University of Shanghai for Science and Technology, Shanghai. His research interests include nonlinear stochastic control and filtering, and wireless sensor networks. He is a reviewer of some international journals.

DERUI DING received the B.Sc. degree in industry engineering and the M.Sc. degree in detection technology and automation equipment from Anhui Polytechnic University, Wuhu, China, in 2004 and 2007, respectively, and the Ph.D. degree in control theory and control engineering from Donghua University, Shanghai, China, in 2014.

From 2007 to 2014, he was a Teaching Assistant and then a Lecturer with the Department of Mathematics, Anhui Polytechnic University. He is currently an Associate Professor with the Department of Control Science and Engineering, University of Shanghai for Science and Technology, Shanghai. In 2012, he was a Research Assistant with the Department of Mechanical Engineering, The University of Hong Kong, Hong Kong. From 2013 to 2014, he was a Visiting Scholar with the Department of Information Systems and Computing, Brunel University London, U.K. He has published around 15 papers in refereed international journals. He is a very active reviewer for many international journals. His research interests include nonlinear stochastic control and filtering, and multi-agent systems and sensor networks.

YONGXIONG WANG received the B.S. degree from Harbin Engineering University, in 1991, and the M.S. and Ph.D. degrees in control theory and control engineering from Shanghai Jiao Tong University, in 2006 and 2012, respectively.

He is currently a Professor with the Department of Control Science and Engineering, University of Shanghai for Science and Technology, Shanghai, China. From 2009 to 2010, he was a Visiting Scholar with the Department of Electrical and Biomedical Engineering, University of Nevada, Reno. He has published over 50 articles in the category of intelligent robots, computer vision, and intelligent control. His research interests include intelligent robots, computer vision, and machine learning.