remote sensing

Article

Radar-Jamming Decision-Making Based on Improved Q-Learning and FPGA Hardware Implementation

Shujian Zheng, Chudi Zhang, Jun Hu * and Shiyou Xu

School of Electronic and Communication Engineering, Sun Yat-sen University, Shenzhen 518107, China; zhengshj23@mail2.sysu.edu.cn (S.Z.); zhangchd@mail2.sysu.edu.cn (C.Z.); xushy36@mail.sysu.edu.cn (S.X.)
* Correspondence: hujun25@mail.sysu.edu.cn

Abstract: In contemporary warfare, radar countermeasures have become multifunctional and intelligent, rendering the conventional jamming method and platform unsuitable for the modern radar countermeasures battlefield due to their limited efficiency. Reinforcement learning has been proven to be a practical solution for cognitive jamming decision-making in the cognitive electronic warfare. In this paper, we proposed a radar-jamming decision-making algorithm based on an improved Q-Learning algorithm. This improved Q-Learning algorithm ameliorated the problem of overestimating the Q-value that exists in the Q-Learning algorithm by introducing a second Q-table. At the same time, we performed a comprehensive design and implementation based on the classical Q-Learning algorithm, deploying it to a Field Programmable Gate Array (FPGA) hardware. We decomposed the implementation of the reinforcement learning algorithm into individual steps and described each step using a hardware description language. Then, the reinforcement learning algorithm can be computed on FPGA by linking the logic modules with valid signals. Experiments show that the proposed Q-Learning algorithm obtains considerable improvement in performance over the classical Q-Learning algorithm. Additionally, they confirm that the FPGA hardware can achieve great efficiency improvement on the radar-jamming decision-making algorithm implementation.

Keywords: radar-jamming decision-making; reinforcement learning; Field Programmable Gate Array

Citation: Zheng, S.; Zhang, C.; Hu, J.; Xu, S. Radar-Jamming Decision-Making Based on Improved Q-Learning and FPGA Hardware Implementation. *Remote Sens.* **2024**, *16*, 1190. https://doi.org/10.3390/rs16071190

Academic Editors: Dusan Gleich, Nebojsa Doncov and Venceslav Kafedziski

Received: 8 February 2024
Revised: 14 March 2024
Accepted: 27 March 2024
Published: 28 March 2024

1. Introduction

In contemporary warfare, electronic warfare has been integrated into the multidimensional theatre of combat that encompasses sea, land, air, and other domains [1,2]. It has become a crucial factor for military operations. Radar jamming, an integral part of electronic warfare, is the attack on enemy radar through the use of radar intelligence [3]. The goal of radar-jamming research is to achieve jamming or destruction of the enemy radar for identification of target information through the method of active jamming, which involves generating different methods of jamming signals to influence the enemy radar receiver. Selecting the most suitable radar-jamming method is a critical aspect of radar-jamming decision-making. Modern warfare's radar confrontation capabilities have become multifunctional and intelligent, and this puts traditional jamming algorithms at a disadvantage [4].

With the increasing popularity of artificial intelligence, reinforcement learning [5] was developed to enable intelligent systems to make choices. It aims to address the challenge of sequential decision-making and has been applied in robot control, game theory, smart urban planning, recommendation systems, and so on. An intelligent agent learns from its environment and interacts with it to achieve a given goal. During the interaction process, the intelligent agent will decide upon a course of action in response to environmental cues, implement that action within the environment, and subsequently receive feedback from the environment in the form of rewards, as well as the state of the next round. This feedback is then used as a learning sample, and the intelligent entity strives to maximize the cumulative rewards obtained over multiple rounds of continuous learning. The workflow of reinforcement learning is illustrated as shown in Figure 1. In the process of interaction of reinforcement learning, the agent will decide the action A_t to be taken by the agent according to the current state S_t of the environment and execute it. The environment interacts with the agent after the agent executes an action. Then the environment jumps to a new state S_{t+1}, giving the agent the reward R_{t+1}. After many episodes, the agent can learn about the environment through reinforcement learning and finally find the optimal solution.

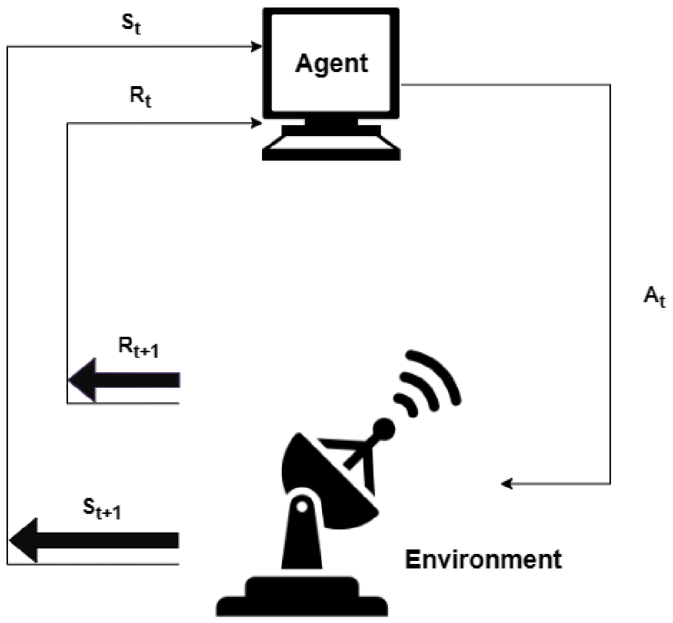

Figure 1. Workflow for reinforcement learning.

To apply the reinforcement learning technique, the radar jammer can be modeled as an intelligent agent in the electromagnetic environment [6–9]. Through this, the radar jammer can learn intelligently the essential information contained within the environment. When deciding on the jamming method, the radar jammer itself can then make optimal decisions [10–12]. This paper focuses on the environment of constrained radar states and limited radar-jamming methods. Among many reinforcement learning algorithms, Q-Learning is believed to be able to work well in this kind of environment using the establishment of a Q-table for query decision-making [13]. However, the traditional Q-Learning algorithm also has shortcomings [14], such as the action-value overestimation and the unstable and non-ideal training results. In this paper, we proposed an improved Q-Learning algorithm, which greatly reduced the possibility of action-value overestimation.

Aside from the above issue, how to efficiently implement the intelligent radar jammer is also full of challenges. The FPGA combines the advantages of an Application Specific Integrated Circuit and a Central Processing Unit. It can be repeatedly programmed and has a large number of parallel distributed storage resources, resulting in high computational performance. Therefore, implementing reinforcement learning algorithms on FPGA platforms can significantly improve the model speed.

When deploying reinforcement learning algorithms onto FPGA platforms, several challenges often arise. For instance, fixed-point computation accuracy on hardware platforms is significantly reduced, and timing design becomes complicated. This paper proposed scaling up the value of fixed points involved in action-value functions, rewards, and so on in an equal ratio to minimize data loss caused by the shift operation. And the intelligent radar-jamming decision-making was split into individual modules. The modules work in conjunction with each other to accomplish the calculations required for the jamming decision-making process. Then, we could solve the difficulties encountered in the FPGA deployment we described above. The main contributions of this paper are as follows:

(1) An improved Q-Learning algorithm for radar-jamming decision-making is proposed, implemented, and evaluated on a software platform. The performance of the improved algorithm is compared with the original Q-Learning algorithm to demonstrate its effectiveness.
(2) The reinforcement learning algorithm is designed with hardware language timing logic on the FPGA platform, and the hardware platform is deployed for the radar-jamming decision-making algorithm based on reinforcement learning for the first time.
(3) A comparison is made between the computation speed and performance of the algorithm in both the FPGA hardware platform and the software platform environments, indicating a considerable speed advantage brought by implementing the FPGA hardware platform.

The rest of this paper is organized as follows. Section 2 describes the background of radar-jamming decision-making, which is also the research context of this paper, including the historical research as well as the current state of development. Section 3 unfolds the description of Q-Learning-based reinforcement learning algorithms and introduces the improved Q-Learning algorithm proposed in this paper. Section 4 describes how the mentioned reinforcement learning algorithms are deployed on an FPGA platform. The results and analyses of the specific experiments are then shown in Section 5. Finally, the conclusion of this paper is given in Section 6.

2. Radar-Jamming Decision-Making

Radar-jamming decision-making methods could be categorized into traditional and reinforcement learning-based methods based on their self-adaptation and self-learning capabilities. Traditional methods lack these abilities, relying instead on a priori knowledge, whereas reinforcement learning-based methods exhibit greater adaptability and knowledge acquisition.

Traditional radar-jamming decision-making algorithms consist of three primary categories [15]: template matching-based jamming decision-making algorithms, game theory-based jamming decision-making algorithms, and inference-based jamming decision-making algorithms. The template matching-based jamming decision-making method is one of the fundamental pattern recognition algorithms, which compares the similarity between the samples to be recognized and those in the template library. Nonetheless, it is excessively reliant on a priori knowledge and is time-consuming.

The process of radar confrontation is constructed as a zero-sum game model using a jamming decision-making method based on game theory. Nonetheless, the success of this method is overly reliant on the establishment of the profit matrix. In the event that a suitable profit matrix cannot be established, the game theory method may not be able to make the optimal decision.

The method of jamming decision-making, which is based on inference, breaks down the radar confrontation process into a distinct set of events. These events are then analyzed for correlation or causation from historical events, using the D-S evidence theory or Bayesian networks to deduce the likelihood of a particular event occurring in the future under specific circumstances. Ultimately, a decision is made. However, in a real-life combat situation, the jammer’s understanding of the enemy’s signals is severely restricted, and obtaining valuable a priori information before the confrontation commences proves challenging. Thus, the research carries some limitations.

The reinforcement learning-based jamming decision-making method has emerged as an area of research with unique “cognitive” abilities [16–18] that go beyond those of traditional methods. This method is capable of adapting its decision-making model by repeatedly trying and learning from errors when prior data are unavailable. As a result, the most effective interference strategy can be achieved through continuous “trial and error” modification of the decision model.

Q-Learning serves as an algorithm for decision-making in jamming, according to literature [19]. It is a table-based reinforcement learning algorithm that can discover the optimal jamming strategy by learning, particularly when the quantity of radar states is relatively small. Reference [20] utilized DQN to address the issue of decreased decision-making efficiency as the number of radar states increases in radar-jamming decision-making. DQN combines Q-Learning with deep learning, training the neural network to output Q-values. This approach saves time by eliminating the need to find the Q-table in the decision-making process and conserves space by not storing the Q-values. By reducing the decision-making time in situations where the number of radar states increases, DQN can effectively enhance decision-making efficiency. However, deep learning in DQN results in more over-parameterization, making it more challenging to adjust parameters compared to Q-Learning.

Moreover, research into applying reinforcement learning algorithms to hardware platforms is still in its initial stages, with literature [21–23] showcasing examples of Q-Learning algorithms applied to FPGA hardware platforms in other fields. As can be seen from the above, the main application of radar-jamming decision-making algorithms is still at the level of software platforms, while FPGAs are gradually becoming an important means of implementation of many algorithms by virtue of their programmable, high-performance, low-latency, and other advantages, and the use of FPGAs for the deployment of reinforcement learning algorithms to achieve will bring more obvious speed advantages. In this paper, we aimed to examine the timing design and necessary steps for transferring the radar-jamming decision-making algorithm based on reinforcement learning to FPGA. Additionally, we accomplished the deployment of the algorithm on the hardware platform.

3. Reinforcement Learning Based on Q-Learning

3.1. Reinforcement Learning

Reinforcement learning, also known as augmented learning or evaluation learning, was developed from theories such as animal learning as well as behavioral psychology. It is essentially an algorithm for solving sequential decision-making problems. Its most important feature is that it does not need to prepare a large number of samples in the learning process but can autonomously and continuously interact with the environment so as to learn. In this process, it is only necessary to set reasonable rewards for each state of the environment so that the intelligent body can make the best decision by trial and error according to the existing strategy in the case of an unknown environment. The environment of radar-jamming decision-making is exactly an environment full of unknown information about the enemy, and reinforcement learning can be fully utilized in this field.

The framework of reinforcement learning has been demonstrated in Figure 1 as a method for studying sequential decision problems, which is mathematically normalized to a Markov decision process. The Markov decision process is a simplification of the reinforcement learning process with respect to the environmental transformation model, action strategies, and the reward function. The schematic diagram of its decision process is shown in Figure 2, where the next state S_{t+1} of the system is only related to the current state S_t of the system, independent of previous states, a property also known as Markovianity. Under the combined action of state S_{t+1} and action a_t, the environmental state jumps to S_{t+1} and gives the corresponding reward r_{t+1}.

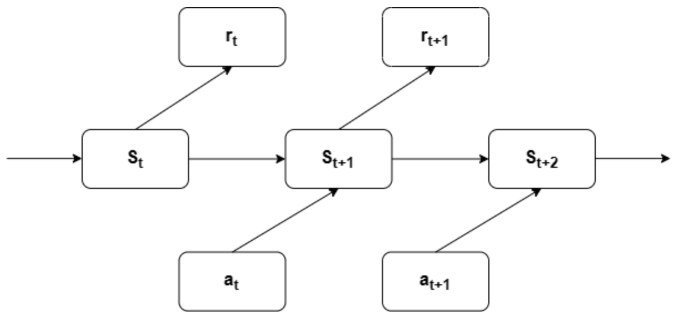

Figure 2. Procedure of MDP.

There are three important elements in the Markov decision process: state, action, and reward. The state is usually the input part of the reinforcement learning algorithm, which contains all the information needed for the agent to make an action judgment. The action is usually the output of the reinforcement learning algorithm, and for the reinforcement learning agent, its goal is to give the action that the agent should take in the current state. Actions are the means by which intelligence interacts with the environment, and the ultimate goal is to learn to obtain the highest possible benefit from this constant interaction. Rewards play a guiding role in reinforcement learning algorithms. This is because the goal of reinforcement learning is to maximize the rewards of interaction. The setting of rewards directly affects the intelligence's evaluation of each action, so setting appropriate rewards is also an important issue in reinforcement learning. The reinforcement learning process can be viewed as a process in which an agent searches for an optimal strategy given a Markov decision process.

In the Markov decision process, there are two important concepts named state-value function $V^{\pi}(s)$ and action-value function $Q^{\pi}(s,a)$. They mean the expected reward the agent receives on the current state s using strategy π and the expected reward the agent receives for performing an action a on the current state s using strategy π. Here, we use G_t to denote the expected reward.

$$V^{\pi}(s) = E_{\pi}[G_t|S_t = s] \tag{1}$$

$$Q^{\pi}(s,a) = E_{\pi}[G_t|S_t = s, A_t = a] \tag{2}$$

We express the expected reward as the sum of the reward in the current state and the reward in the future state multiplied by the discount factor γ. Then, we could obtain the Bellman expectation equation. Here, we use P to denote the state transition probability matrix.

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}[R_t + \gamma V^{\pi}(s')|S_t = s] \\ &= \sum_{a\in A} \pi(a|s)(r(s,a) + \gamma \sum_{s'\in S} P(s'|s,a)V_{\pi}(s')) \end{aligned} \tag{3}$$

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}[R_t + \gamma Q^{\pi}(s',a')|S_t = s, A_t = a] \\ &= r(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) \sum_{a'\in A} \pi(a'|s')Q^{\pi}(s',a') \end{aligned} \tag{4}$$

When the strategy is optimized, we can further obtain the Bellman optimality equation. It means that the state-value function $V^{\pi}(s)$ and action-value function $Q^{\pi}(s,a)$ now are the best. These two equations will help us learn about the following algorithm.

$$V^*(s) = \underset{a\in A}{max}\left\{ r(s,a) + \gamma \sum_{s'\in S} P(s'|s,a)V^*(s') \right\} \tag{5}$$

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) \underset{a'\in A}{max}\, Q^*(s',a') \tag{6}$$

From the perspective of methods, reinforcement learning is mainly divided into model-based reinforcement learning methods and model-free reinforcement learning methods, and the general classification is shown in Figure 3. In the model-based reinforcement learning method, the environment model is known, i.e., the parameters involved, such as action set, state set, value set, etc. are known. Common model-based reinforcement learning algorithms are policy iteration, value iteration, and policy search, among others. However, such cases are rare, and in many cases, the environment to be faced with is a model-free reinforcement learning environment. In this case, the agent has to interact with the environment and try to obtain various information from the environment. Common model-less reinforcement learning methods include temporal difference methods and Monte Carlo methods.

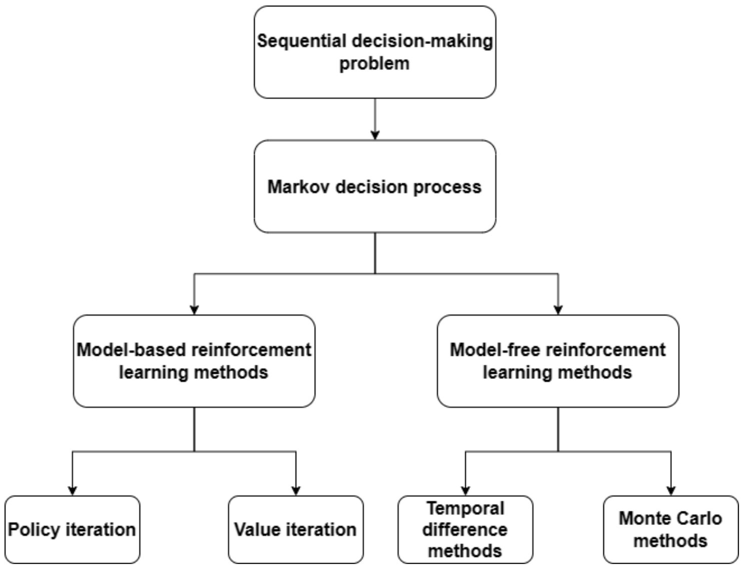
Sequential decision-making problem
Markov decision process
Model-based reinforcement learning methods
Model-free reinforcement learning methods
Policy iteration
Value iteration
Temporal difference methods
Monte Carlo methods

Figure 3. Classification of reinforcement learning methods.

3.2. *Q-Learning Algorithm*

The Q-Learning algorithm is a conventional reinforcement learning method grounded on the temporal difference method [24]. The sequential difference algorithm is utilized to assess a policy's value function. It merges the principles of Monte Carlo and dynamic programming algorithms. The algorithm can learn from sample data similar to Monte Carlo algorithms and does not necessitate prior knowledge of the environment. Additionally, it can update the value estimation of the present state using Bellman's equation, which is similar to dynamic programming algorithm value assessments of the ensuing state's value estimation.

Unlike Monte Carlo algorithms, which require learning after a full episode, Q-Learning algorithms can update the Q function after each state update. This incremental dynamic planning calculates the reward obtained from the current state, which is the current reward plus the value estimate of the next state. As it is a value-based algorithm, it is less reliant on the outcome of the round. In the equations, the parts in parentheses are called temporal difference errors (TD errors). In addition, we only need to use temporal difference error to update the policy after every action of the agent.

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)] \tag{7}$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{8}$$

The pseudo-code for the implementation of the Q-Learning algorithm is as Algorithm 1. In the algorithm, $Q(s,a)$ was called Q-table. It lists the value of each action in each state. The agent will constantly update it to find the best action in each state. In addition, the algorithm combines the Bellman optimality equation with the temporal difference error.

It updates the Q-table with the optimal action-value function each time. It is radical and effective.

Algorithm 1: Implementation of the Q-Learning algorithm.

```
Initialise the $Q(s,a)$ table;
foreach cycle do
    Initialise state s;
    foreach update step in the cycle do
        Select action a in the current state s according to a strategy (e.g., an
          $\epsilon$-greedy strategy);
        Observe the new state s′ and the obtained reward r according to the chosen
          action;
        Update the $Q(s,a)$ table:
            $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
```

3.3. Improved Q-Learning Algorithm

While Q-Learning is a classic algorithm, it does have limitations. In the case of using a single estimator, the conventional approach is prone to significant overestimation of the action value, as well as poor and unstable training outcomes. One of the primary reasons for the overestimation of the action value is the update process of the action-value function Q. At each step, the maximum value of $Q(s',a')$ is used for updating. However, this approach can result in a significant overestimation of the action taken by the intelligent agent. Such overestimation can adversely affect the performance of Q-Learning after multiple updates and accumulations.

To address the above problems, this paper proposes an improved algorithm for the Q-Learning algorithm. The algorithm will initialize two $Q(s,a)$ tables, referred to as the Q_A and Q_B tables, and improve the updating approach based on the original Q-Learning algorithm's updating of Q-values:

$$Q_A(s,a) \leftarrow Q_A(s,a) + \alpha[r + \gamma \max_{a'} Q_B(s',a') - Q_A(s,a)] \tag{9}$$

In Equation (9), α is the step size of the value estimate. γ is the discount factor for future estimation in the temporal difference process. r is the reward for the state jump. It also means that in this update process, we will use the value of Q_B to update Q_A. At the same time, after every certain number of updates, the Q_A table is synchronized to the Q_B table. Such an operation can solve the problem of overestimation of action values, i.e., the workflow of the improved Q-Learning algorithm is as Algorithm 2.

Algorithm 2: Implementation of the improved Q-Learning algorithm.

```
Initialise the $Q_A(s,a)$ and $Q_B(s,a)$ tables;
foreach cycle do
    Initialise state s;
    foreach update step in the cycle do
        Select action a in the current state s according to a policy (e.g., $\epsilon$-greedy
          policy);
        Observe the new state s′ and the obtained reward r according to the chosen
          action;
        Update the $Q_A(s,a)$ table:
            $Q_A(s,a) \leftarrow Q_A(s,a) + \alpha[r + \gamma \max_{a'} Q_B(s',a') - Q_A(s,a)]$
        Update state s;
    Synchronise $Q_A(s,a)$ to $Q_B(s,a)$ after updating N steps.
```

4. Implementation on FPGA

4.1. Hardware Platform

The system architecture of the hardware platform involved in this paper is shown in Figure 4. The system comprises a personal computer (PC) host computer, an FPGA board, and an ADRV9009 RF agile transceiver. The FPGA board is categorized into a programmable logic unit (PL) and a processing unit (PS). The PC can directly connect with the PS of the FPGA board to transmit the necessary hardware configuration parameters to the PS of the FPGA board. PS then sends it down to PL. The ADRV9009 is responsible for interacting with the environment to obtain the necessary information. JESD refers to the JESD204B transport protocol, which allows high-speed communication between ADRV9009 and PL.

The system will be initialized, and initial parameters will be configured by the PC host computer through serial port communication. Subsequently, the ADRV9009 will transmit received signals to the PL of the FPGA board via the JESD interface. During this process, the received signals will be demodulated into IQ signals. The PL's pulse measurement module will scrutinize the parameters of received IQ signals and deduce the current radar's operational condition.

The radar-jamming decision module will receive this input. Subsequently, relying on the reinforcement learning algorithm to make decisions about radar jamming, the radar-jamming decision-making module will direct the necessary jamming action to the jamming module after collecting radar work data. The module for jamming converts the actions taken by the module for radar-jamming decisions into a waveform signal for the corresponding jamming method. This signal is then transmitted through the JESD bus to the ADRV9009 RF agile transceiver, which transmits the jamming signal to complete the jamming work once.

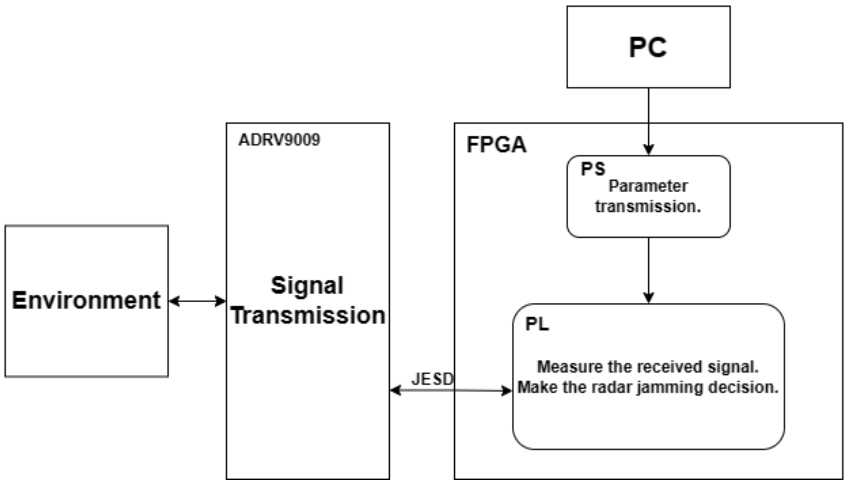

Figure 4. The system architecture of the hardware platform.

4.2. Migration and Deployment

For the FPGA hardware platform, numerous logic gate circuits are present. The principal task is to design the timing logic of the algorithm. This entails pulling up the corresponding timing control signals at the moment of every signal input and output and passing it to the respective module. For each module, the specific workflow is illustrated as shown in Figures 5 and 6.

We can see that in Figure 5, the agent first compares the Q-values and updates the Q-table using the state changes in the past step after obtaining the environmental information. The agent then outputs the highest Q-value to the action-selected module. The action-selected module will use the ϵ-greedy strategy to make decisions and execute actions. Finally, the environment will interact with the agent performing the action and make a change in state. And in Figure 6, different from Figure 5, Q-table goes from one to two.

After each state jump, the Q_A table is updated with the maximum Q-value obtained from the Q_B table comparison. The Q_A table will be synchronized to the Q_B table after updating several steps.

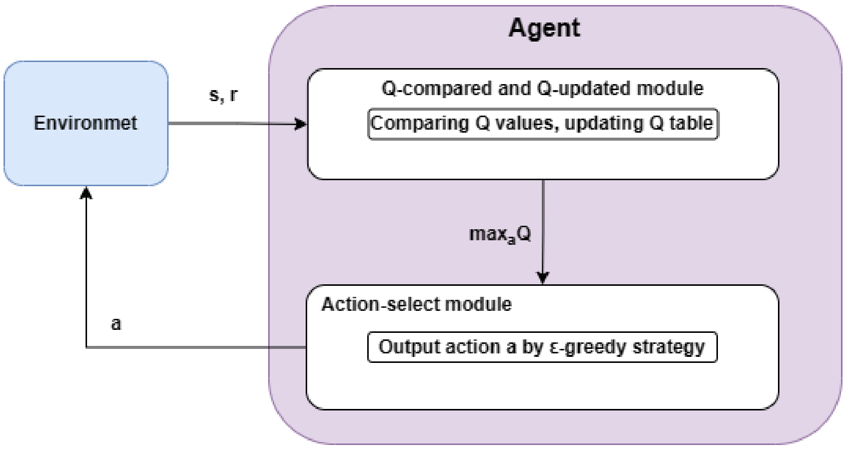
Agent
Environmet
s, r
Q-compared and Q-updated module
Comparing Q values, updating Q table
max$_a$Q
Action-select module
Output action a by ε-greedy strategy
a

Figure 5. Q-Learning algorithm implementation.

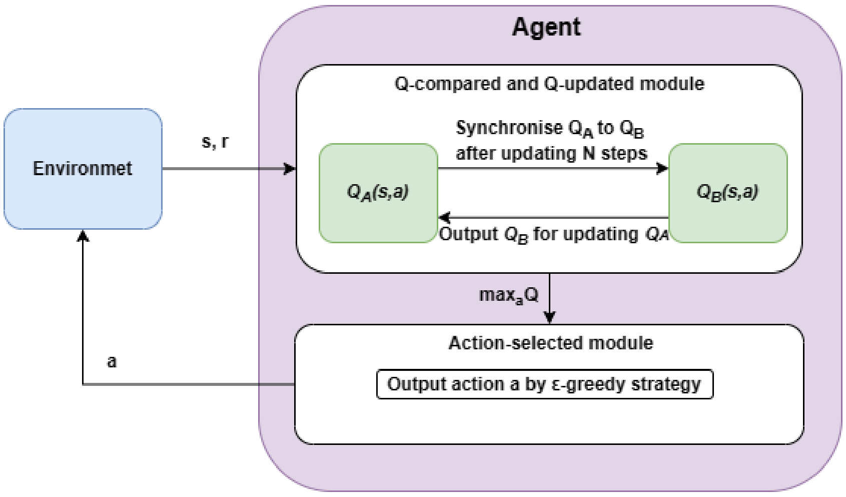
Agent
Q-compared and Q-updated module
Environmet
s, r
Synchronise Q_A to Q_B after updating N steps
$Q_A(s,a)$
$Q_B(s,a)$
Output Q_B for updating Q_A
max$_a$Q
Action-selected module
Output action a by ε-greedy strategy
a

Figure 6. Improved Q-Learning algorithm implementation.

After system initialization, the state will reset to its initial state, and the system will commence reinforcement learning training.

(1) Compare the value functions of each available action in the current state and output the highest Q-value and the index of the action with the highest Q-value after conducting the comparison. Also, pull up the valid signal by one pulse while indicating the end of the comparison. Furthermore, provide an explanation of any technical terms when introduced.

(2) After detecting the completion signal, the action selection module implements the ϵ-greedy strategy to choose an action for the intelligent agent. The selected action is then outputted, and a valid signal for a one-pulse action change is raised simultaneously.

(3) Once the action change valid signal is pulled high, the state transfer module jumps to the next state based on the state transfer probability matrix and increments the state change valid signal by one pulse.

(4) Upon completion of the state transfer, the Q-table update module updates the Q-table by utilizing the highest Q-value in the new state along with the completed state jump. The system generates the Q-value update valid signal.

(5) Upon receiving the Q-value update valid signal, the system will iterate steps (1) to (4) until one cycle is complete and the state transitions to the endpoint or failure state.

(6) The procedure will be repeated numerous times until convergence is reached after running steps (1) to (5).

5. Experiments

5.1. Environment Modeling

To achieve the goal of reinforcement learning, it is essential to create a training model of the radar-jamming environment. The current "AN/SPY" radar series represents the latest radar technology, which encompasses a range of functions, including search, tracking, ranging, imaging, guidance, and so on. Additionally, the efficiency of different jamming methods on the radar varies depending on the state of the radar. Therefore, the use of reinforcement learning to intelligently select the appropriate jamming method in different target radar states becomes crucial. It is essential to apply reinforcement learning techniques to enable the jammer to intelligently choose the appropriate jamming type according to various states of the target radar.

In this paper, the characteristics of a particular radar were analyzed. We assumed that the radar follows the Markov decision-making process in its application environment and considered the state of the target radar as the environment state in the Markov decision-making process. We also looked at different radar-jamming methods as potential actions in the Markov decision-making process. Using the process of interaction between the intelligent system and its environment to obtain information on the next step and its rewards, followed by the application of reinforcement learning to develop an intelligent radar jammer that can selectively jam the target radar in different states, with the ultimate aim of maximizing revenue.

In this paper's established model species within this radar environment, there were 10 radar states, ranging from the search state with the lowest danger level to the guidance state with the highest danger level. This is to quantitatively describe changes in the radar threat level. Radar jamming is mainly composed of suppressive jamming and deceptive jamming. In this experiment, we selected the following four representative methods of suppressive jamming: comb spectrum jamming, noise amplitude modulation jamming, noise phase modulation jamming, and noise frequency modulation jamming in suppressive jamming. We also selected the following four representative methods of deceptive jamming: smart noise jamming, full pulse forwarding jamming, slice forwarding jamming, and intermittent sampling jamming. After modeling states and actions, we established the state transfer matrix P. This matrix contained the probability of transitioning from the current state to each state that corresponds to a jamming action in all states. Additionally, the reward value of the endpoint was set at 100, while the values of the remaining states decreased as the distance from the endpoint increased. It was defined that the radar jammer performed one jamming method decision as one interaction process of reinforcement learning.

In this process, the action strategy adopted for intelligent jamming decision-making was designed as an ϵ-greedy strategy.

The block diagram of radar-jamming is shown in Figure 7. First, our radar jammer will enter the range of the enemy radar and then start working. The radar jammer will then start making decisions about radar-jamming methods based on the state of the enemy radar. Once the jammer implements jamming, the enemy radar will be affected, and the radar state will change. After the radar jammer obtains information about the change in the enemy's radar state, it will make the next judgment according to the current state of the enemy's radar. If the enemy radar is jammed to a safe state, the radar jammer will judge the success of the jamming and then end the jamming work. Otherwise, the jammer will adjust its own strategy based on the feedback information and make the next jamming decision, which is a learning process. Jamming decisions will take place until the enemy's radar is successfully jammed into a safe state.

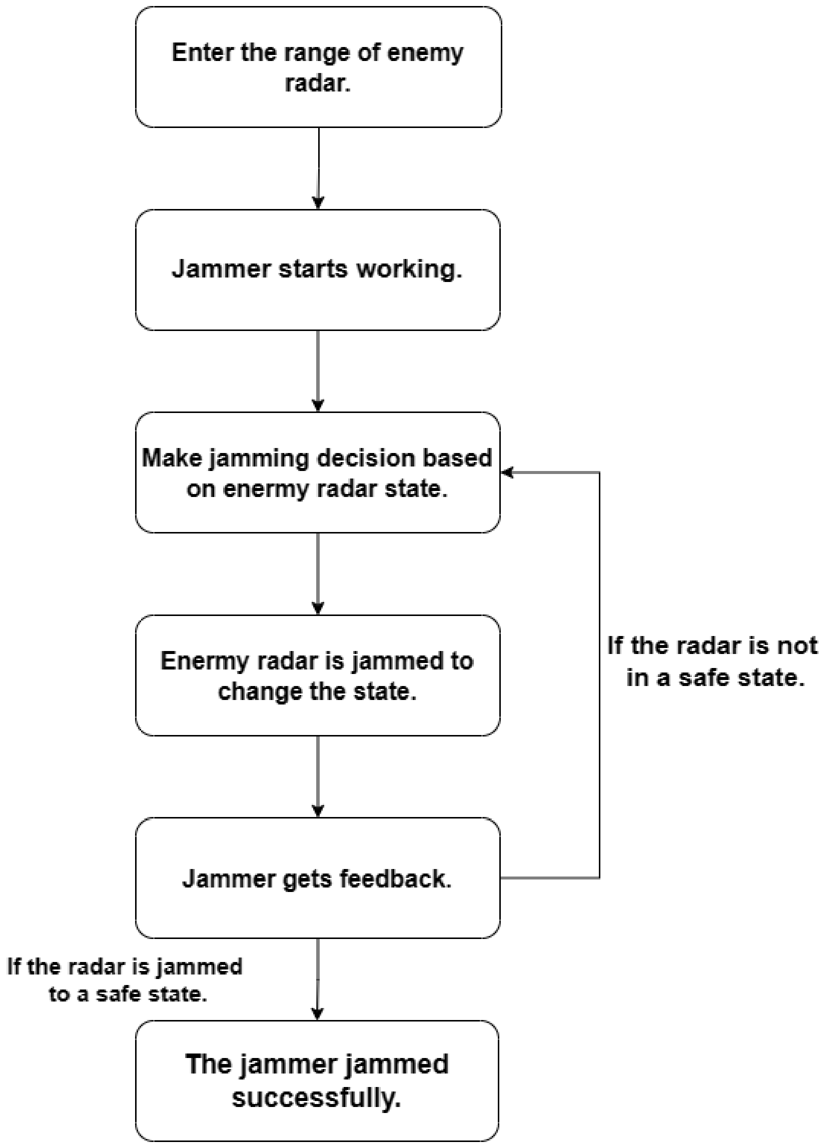
Enter the range of enemy radar.
Jammer starts working.
Make jamming decision based on enermy radar state.
Enermy radar is jammed to change the state.
Jammer gets feedback.
If the radar is not in a safe state.
If the radar is jammed to a safe state.
The jammer jammed successfully.

Figure 7. The block diagram of radar-jamming process.

5.2. Experiments on Software Platform

Unlike some simple environments, the radar-jamming decision environment is much more complex. In the radar-jamming decision-making environment, due to the existence of the probability transfer matrix, even if we clearly know the states and actions of the environment, we cannot be sure how the state will be jumped by selecting a certain action in a certain state, so it is more necessary for the agent to help make decisions.

First, the Q-Learning simulation was recorded using the PyTorch framework, where the number of steps required for the agent to reach the endpoint and the rewards obtained in each episode and the results were visualized in Figures 8 and 9. To make the data more intuitive, the moving average algorithm was used here to process the data results, as shown in Figures 10 and 11. It can be seen that the number of steps to be used in each episode showed a decreasing trend with the advancement of the episode, while at the same time, the rewards obtained in each episode showed an increasing trend with the advancement of the episode. After about 6600 episodes, the model stabilizes.

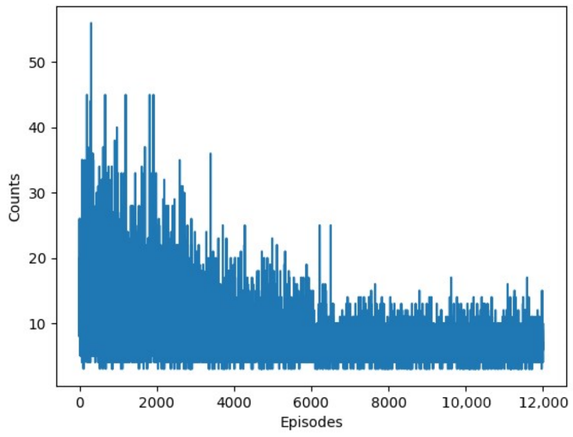

Figure 8. Changes in the number of steps per episode.

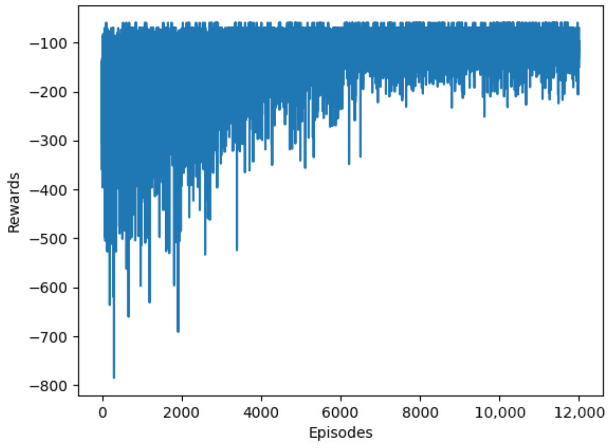

Figure 9. Changes in the reward values per episode.

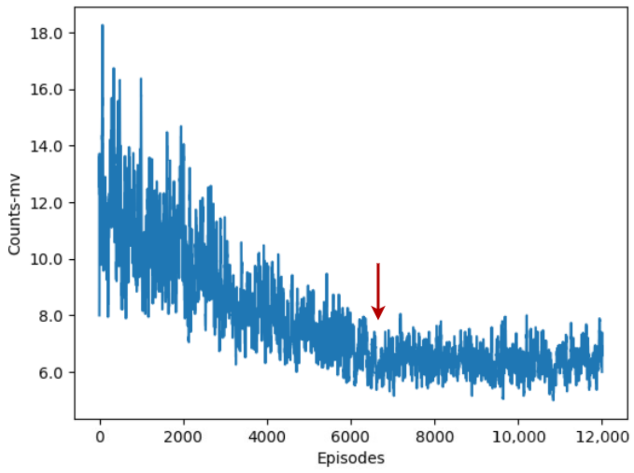

Figure 10. Changesin the number of steps per episode (MA).

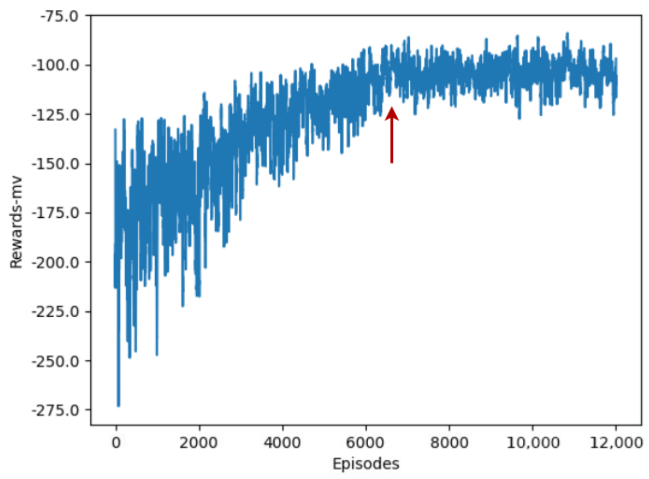

Figure 11. Changes in the reward values per episode (MA).

Then, the improved Q-Learning simulation was carried out using the PyTorch framework as well, recording the number of steps required for the agent to reach the endpoint and the rewards obtained in each episode, and the results were visualized as shown in Figures 12 and 13. Again, in order to make the data more intuitive, the moving average algorithm was used here to process the data results, as shown in Figures 14 and 15.

Comparing the results of the improved Q-Learning algorithm with the Q-Learning algorithm, it can be seen that the improved Q-Learning algorithm and the Q-Learning algorithm share a similar convergence trend, and the improved Q-Learning algorithm model tended to stabilize after about 5800 episodes, which was a considerable improvement in the convergence speed compared to the 6600 episodes of the Q-Learning algorithm.

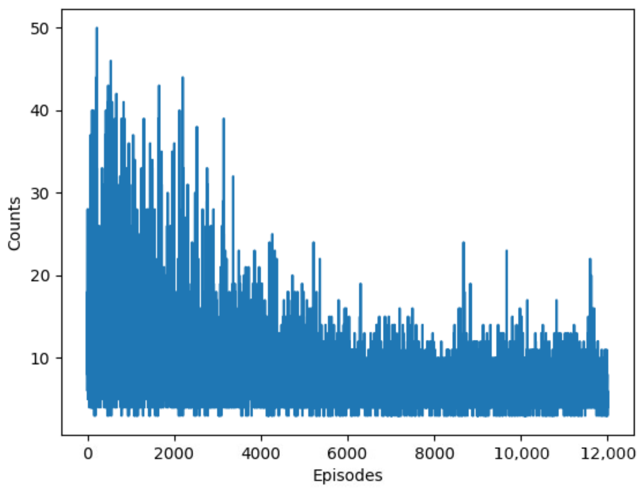

Figure 12. Changes of Improved Q-L in the number of steps per episode.

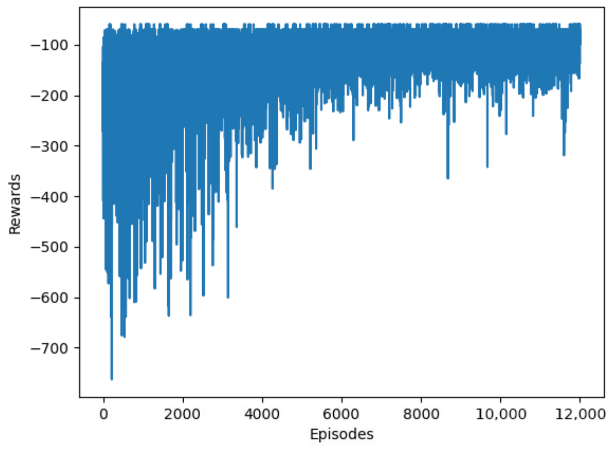

Figure 13. Changes of Improved Q-L in the reward values per episode.

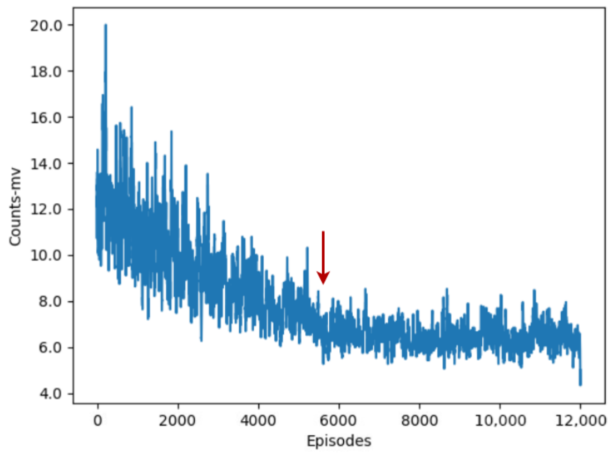

Figure 14. Changes of Improved Q-L in the number of steps per episode (MA).

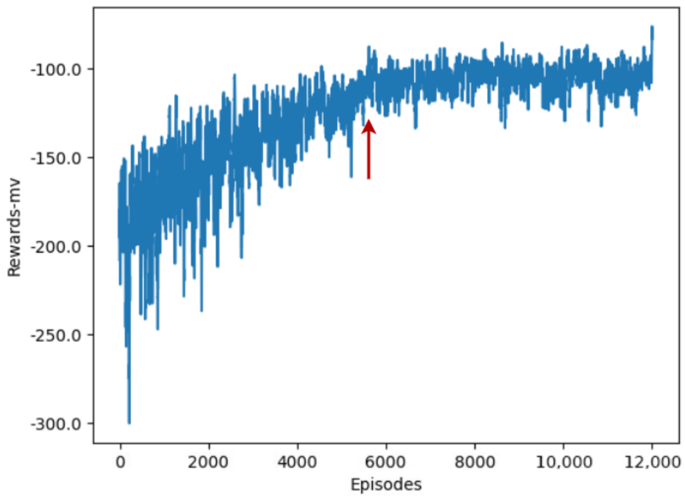

Figure 15. Changes of Improved Q-L in the reward values per episode (MA).

5.3. The Impact of Hyperparameters

In this section, we will analyze the impact of various hyperparameters in the improved Q-Learning algorithm. We would analyze their results in terms of the speed at which the model converged. First, we would focus on the impact of learning rate α. We traversed the α and performed a centralized value analysis around the optimal value. We set the α to 0.0625, 0.125, and so on, respectively. The simulation calculation was carried out several times, respectively, and the average value was taken after the unreasonable result was discarded. And here we were just going to pick the hundreds of digits for convenience. Then we obtained the results shown in Table 1.

Table 1. Convergence speed of different α settings.

α	**Speed (Episodes)**
0.0625	6400
0.125	5800
0.25	6300
0.375	6400
0.5	6400

We could see that as the α moved away from the optimal value, the convergence rate became slower and slower. This was because when the learning rate was too small, there was a possibility that the model would not fit. However, a learning rate that is too large could also lead to the risk of overfitting.

In the same way, we analyzed the impact of discount factor γ. We also traversed the γ and performed a centralized value analysis around the optimal value. Then, the results were given in Table 2. We could also see that as the γ moved away from the optimal value, the convergence rate became slower and slower. That was because the model was prone to local high-yield decisions, and there was a risk of failing to learn to the end when the γ was setting small. When the discount factor is too large, focusing too much on the future may cause the current reward to become blurred.

Table 2. Convergence speed of different γ settings.

γ	**Speed (Episodes)**
0.6875	6300
0.75	6200
0.8125	5800
0.875	6100
0.9375	6400

Finally, we looked at the impact of reward r settings. We set the r at the end as 100. The r of other states were all set as negative numbers. Then, we changed the r given by the environment when the state dropped. For example, when the penalty was set as 1, the r of the state closest to the end was -1. Then, the results were given in Table 3. We could see that when the r was set as 2, the convergence speed was the fastest. It was because a reward gap that is too small may lead to an ambiguous direction of state jumps. However, a reward gap that is too large would cause the agent to be conservative.

Table 3. Convergence speed of different *r* settings.

r	Speed (Episodes)
The penalty is 1 for each state away from the end	6100
The penalty is 2 for each state away from the end	5800
The penalty is 3 for each state away from the end	5900
The penalty is 4 for each state away from the end	6400

5.4. Experiments on Hardware Platform

After implementation on the software platform, the next step was to migrate to the FPGA hardware platform, which required the use of a hardware language to describe Q-Learning, so some timing adjustments were needed here. Because the number of actions was 8 when selecting actions in this environment, if we used combinational logic to compare the 8 actions at once, the complexity of the combinational logic would be greatly increased, which would lead to difficulties in layout and wiring. Therefore, we utilized an additive tree for two-by-two comparisons and stored and passed them. The Q-values of each action were divided into four groups for two-by-two comparison, and each group selected the larger value and gave the valid signals for output (i.e., the first four VALID signals in Figure 16). Based on this, the four larger Q-values obtained in the first step were again divided into two groups for comparison; two larger Q-values were obtained, and valid signals were output (i.e., the fifth and sixth valid signals in the figure). From this, a final comparison was made to obtain the action with the largest Q-value.

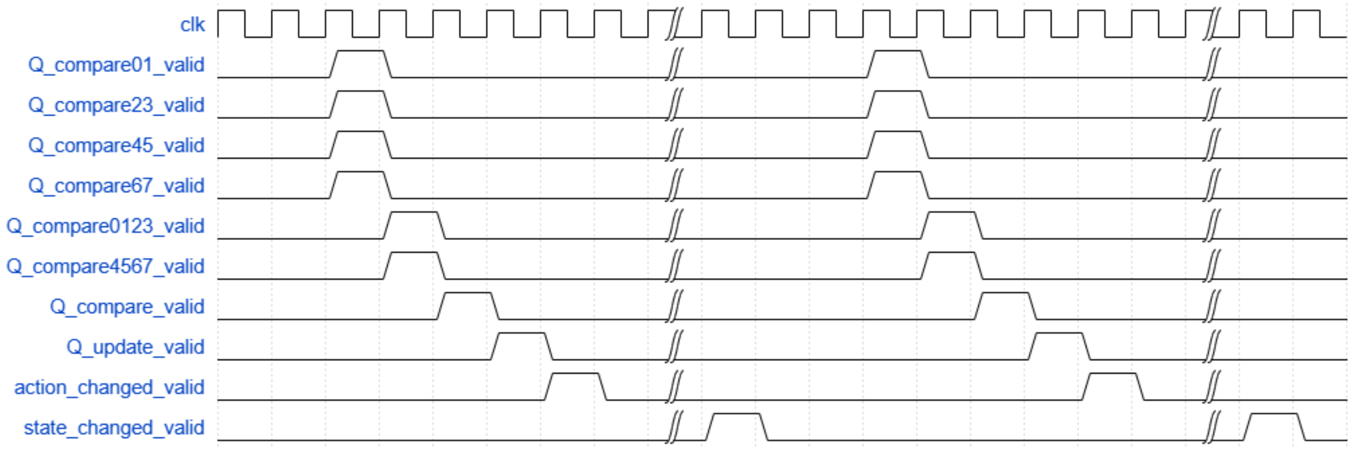

Figure 16. Q-Learning algorithm implementation.

After obtaining the action with the largest Q-value, Q-Learning used the ϵ-greedy strategy for action selection. Here, the probability transfer matrix for each action was involved, but it was difficult to implement the fractions in hardware circuits. Linear Feedback Shift Register (LFSR) was introduced to obtain a pseudo-random sequence. We chose a 16-bit Linear Feedback Shift Register to generate a set of pseudo-random sequences of length 65,535. The probability shift matrix was multiplied with 65,535 to obtain the value domain corresponding to each action, and the size of the LFSR value at that point was used to decide the selection of the action at the moment of action selection. The state transfer and the updating of the Q-values were kept the same as the original timing design.

After completing the timing design, we proceeded with the deployment implementation of the Q-Learning algorithm in the FPGA to allow the agent to start training and learning for 12,000 episodes. Here, the results were analyzed using the cumulative rewards obtained by the agent and the number of steps required to reach the endpoint in each episode. The learning data were exported and visualized using Matlab tools, and the results are shown in Figures 17 and 18. Similarly, the resultant data were processed using Matlab's moving average function to make them more intuitive in Figures 19 and 20.

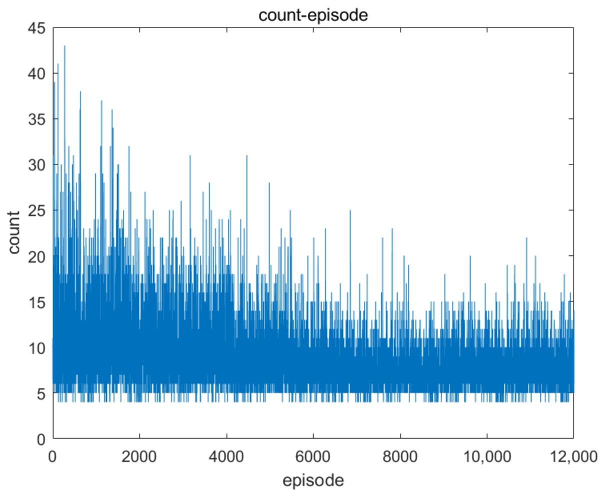

Figure 17. Changes in the number of steps per episode on FPGA.

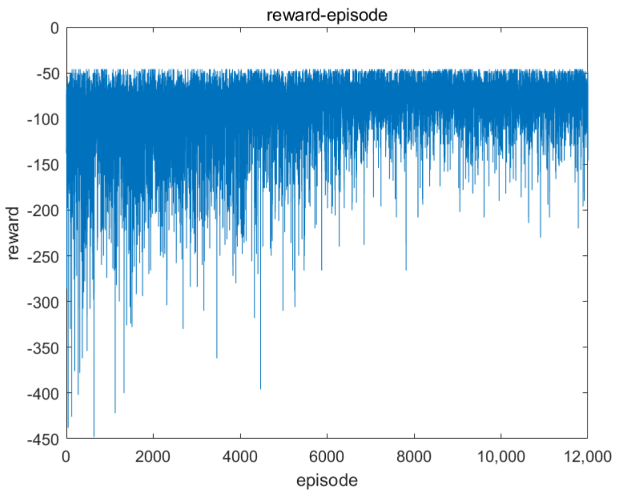

Figure 18. Changes in the reward values per episode on FPGA.

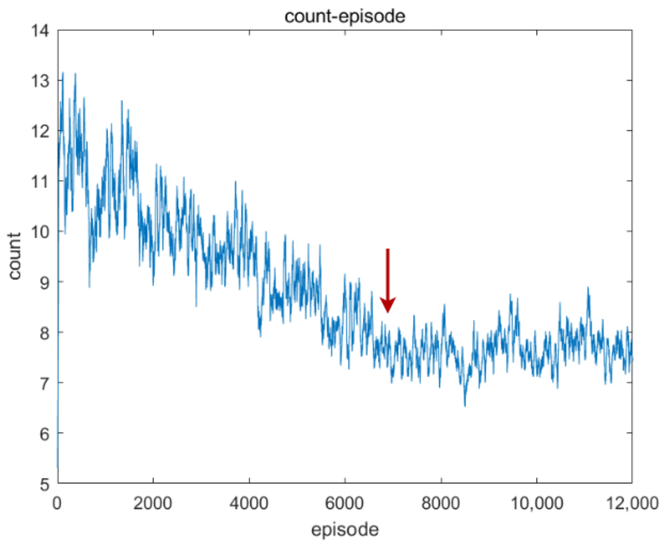

Figure 19. Changes in the number of steps per episode on FPGA(MA).

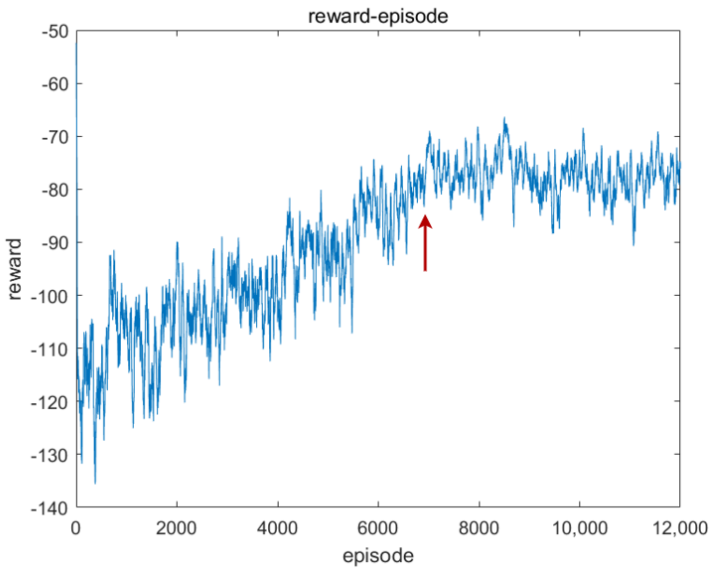

Figure 20. Changes in the reward values per episode on FPGA(MA).

It could be seen that, as with the software platform, the number of steps used per episode showed a downward trend, while the rewards gained per episode showed an upward trend as the episode progressed. And the Q-Learning algorithm would tend to finish learning after about 7000 episodes. At this point, the agent would move steadily towards the end, there would be almost no state jumps in the opposite direction, and a judgment could be made that the experiment is successful. On this basis, the algorithm was updated to the improved Q-Learning algorithm, and the experimental results were as follows (Figures 21–24).

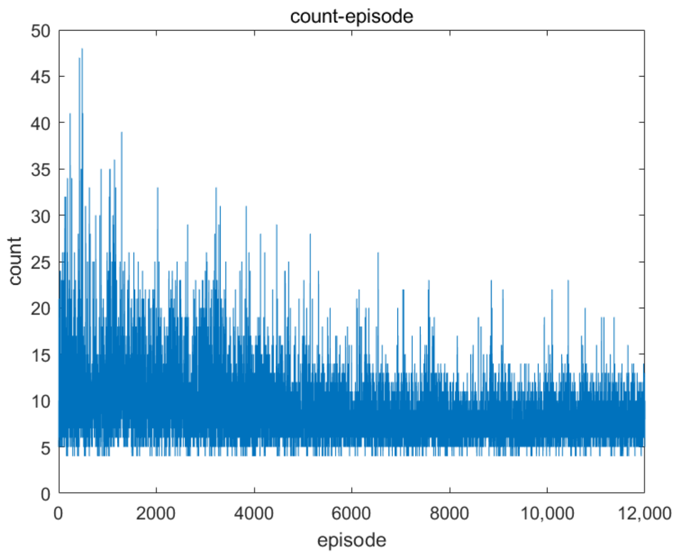

Figure 21. Changes of Improved Q-L in the number of steps per episode on FPGA.

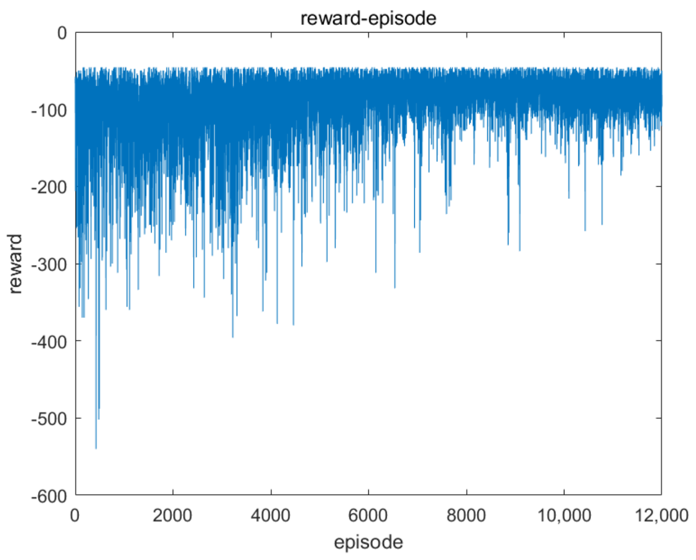

Figure 22. Changes of Improved Q-L in the reward values per episode on FPGA.

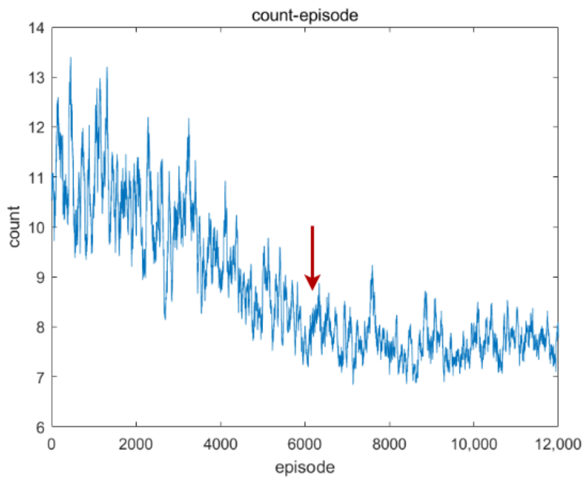

Figure 23. Changes of Improved Q-L in the number of steps per episode on FPGA (MA).

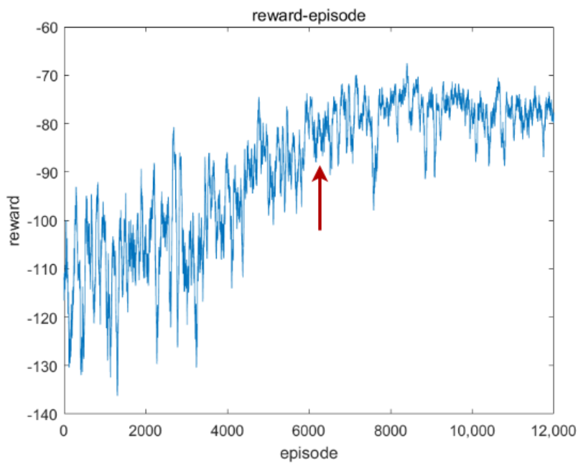

Figure 24. Changes of Improved Q-L in the reward values per episode on FPGA (MA).

Comparing the results of the improved Q-Learning algorithm and the Q-Learning algorithm implemented on the hardware platform, it could be seen that the improved Q-Learning algorithm and the Q-Learning algorithm also shared a similar convergence trend on the hardware platform. After about 6100 episodes, the improved Q-Learning algorithm model tended to stabilize, which was a considerable improvement in the computation speed compared to the Q-Learning algorithm's 7000 cycles. In line with the conclusions of the software platform, the experiment could be considered successful.

5.5. Analysis and Discussion

Comparing the results of implementing the Q-Learning algorithm and the improved Q-Learning algorithm on a software platform as well as on a hardware platform, it could be seen that both algorithms took only about 2.82 ms to train for 12,000 cycles at the operating frequency of the hardware platform using a clock with a frequency of 250 MHz. Even if the clock frequency is reduced to 100 MHz, it only takes about 7.05 ms. In contrast, timing the software platform environment reveals that 12,000 episodes of model computation for both algorithms on an i9-10900K CPU took about 9.36 s. Even with the hardware platform operating at a 100 MHz clock, the computation speedup compared to the software platform is nearly 1200 times faster. For the convergence-specific speed, we could see Tables 4 and 5.

Table 4. Convergence speed (ms) of different algorithms on different platforms.

Item	Speed (ms)
CPU with Q-L	5336.90
CPU with improved Q-L	4798.28
FPGA with Q-L at 100 MHz clock frequency	4.43
FPGA with improved Q-L at 100 MHz clock frequency	3.95

Table 5. Convergence speed (episodes) of different algorithms on different platforms.

Item	Speed (Episodes)
CPU with Q-L	6600
CPU with improved Q-L	5800
FPGA with Q-L at 100 MHz clock frequency	7000
FPGA with improved Q-L at 100 MHz clock frequency	6100

On the other hand, the number of episodes required for convergence of reinforcement learning computation performed on the hardware platform would be slightly more than the number of episodes required on the software platform. This was because fixed-point computation is used for computation on the hardware platform. Compared to the floating-point computation method used on the software platform, fixed-point computation would have lower accuracy than floating-point computation while achieving higher speed. It meant that errors would be generated, which was a drawback of computation on hardware platforms.

Compared to the extension of the computation cycle, the advantage of the magnitude of the speed increase was more obvious, which showed the superiority of implementing reinforcement learning algorithms on FPGA hardware platforms.

6. Conclusions

This paper presented an improved Q-Learning algorithm for making decisions regarding radar jamming and covered the timing and specifics of its successful migration onto a hardware platform. The feasibility of our method has been thoroughly investigated through experimentation, including a thorough comparison of the performance differences between the improved Q-Learning algorithm and the standard Q-Learning algorithm. Additionally, the speed differences between the different algorithms in software and hardware platforms were compared. The experimental results proved that the improved Q-Learning algorithm had considerable improvement in the speed of model convergence compared with the original algorithm. Meanwhile, it could be seen that the computation speed of the Q-Learning algorithm and the improved Q-Learning algorithm on the hardware platform have an obvious advantage over the software platform.

Following the successful use of reinforcement learning algorithms on FPGA hardware platforms, a crucial area for future research lies in implementing neural networks and deploying deep reinforcement learning algorithms on FPGA hardware platforms. This will benefit radar-jamming decision-making, improving both efficiency and accuracy.

Author Contributions: Methodology, S.Z. and C.Z.; Investigation, S.Z.; Data curation, S.Z.; Writing—original draft, S.Z.; Writing—review & editing, C.Z. and J.H.; Visualization, J.H.; Supervision, C.Z., J.H. and S.X.; Project administration, J.H. and S.X.; Funding acquisition, J.H. and S.X. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partly supported by the National Key R&D Program of China under Grant 2021YFA0716600, the Shenzhen Fundamental Research Program under Grant No. JCYJ20180307151430655 and the Shenzhen Science and Technology Program under Grant No. KQTD20190929172704911.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Haykin, S. Cognitive radar: A way of the future. *IEEE Signal Process. Mag.* **2006**, *23*, 30–40. [CrossRef]
2. Gong, L.; Wu, S.; Lv, T. A radar emitter identification method based on pulse match template sequence. *Mod. Def. Technol.* **2008**, *3*, V3-153–V3-156.
3. Hao, H.; Zeng, D.; Ge, P. Research on the Method of Smart Noise Jamming on Pulse Radar. In Proceedings of the 2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC), Qinhuangdao, China, 18–20 September 2015; pp. 1339–1342. [CrossRef]
4. Morris, G.; Kastle, T. Trends in electronic counter-countermeasures. In Proceedings of the NTC '91—National Telesystems Conference Proceedings, Atlanta, GA, USA, 26–27 March 1991; pp. 265–269. [CrossRef]
5. Sutton, R.; Barto, A. Reinforcement Learning: An Introduction. *IEEE Trans. Neural Netw.* **1998**, *9*, 1054–1054. [CrossRef]
6. Zhang, C.; Wang, L.; Jiang, R.; Hu, J.; Xu, S. Radar Jamming Decision-Making in Cognitive Electronic Warfare: A Review. *IEEE Sens. J.* **2023**, *23*, 11383–11403. [CrossRef]
7. Liu, H.; Zhang, H.; He, Y.; Sun, Y. Jamming Strategy Optimization through Dual Q-Learning Model against Adaptive Radar. *Sensors* **2022**, *22*, 145. [CrossRef] [PubMed]
8. Li, W.G.; Wu, C.H. Research on Self-Learning Model of Intelligent Radar Jamming System. *Mod. Def. Technol.* **2009**, *37*, 83–86.
9. Xing, Q.; Zhu, W.-G.; Jia, X. Intelligent countermeasure design of radar working-modes unknown. In Proceedings of the 2017 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC), Xiamen, China, 22–25 October 2017; pp. 1–5. [CrossRef]
10. Mao, S. Research on Intelligent Jamming Decision-Making Methods Based on Reinforcement Learning. Master's Thesis, Harbin Institute of Technology, Harbin, China, 2021.
11. Gao, L.; Liu, L.; Cao, Y.; Wang, S.; You, S. Performance analysis of one-step prediction-based cognitive jamming in jammer-radar countermeasure model. *J. Eng.* **2019**, *2019*, 7958–7961. [CrossRef]
12. Zhang, B.; Zhu, W. Research on Decision-making System of Cognitive Jamming against Multifunctional Radar. In Proceedings of the 2019 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC), Dalian, China, 20–23 September 2019; pp. 1–6. [CrossRef]
13. Zhang, C.; Song, Y.; Jiang, R.; Hu, J.; Xu, S. A Cognitive Electronic Jamming Decision-Making Method Based on Q-Learning and Ant Colony Fusion Algorithm. *Remote Sens.* **2023**, *15*, 3108. [CrossRef]
14. Li, H.; Li, Y.; He, C.; Zhan, J.; Zhang, H. Cognitive Electronic Jamming Decision-Making Method Based on Improved Q-Learning Algorithm. *Int. J. Aerosp. Eng.* **2021**, *2021*, 8647386. [CrossRef]
15. Zhu, B.; Zhu, W.; Li, W.; Yang, Y.; Gao, T. A Review on Reinforcement Learning Based Radar Jamming Decision-Making Technology. *Electron. Opt. Control.* **2022**, *29*, 52.
16. Wang, S.; Bao, Y.; Li, Y. The architecture and technology of cognitive electronic warfare. *Sci. Sin. Inf.* **2018**, *48*, 1603–1613. [CrossRef]
17. Ye, F.; Che, F.; Tian, H. Cognitive cooperative-jamming decision method based on bee colony algorithm. In Proceedings of the 2017 Progress in Electromagnetics Research Symposium—Fall (PIERS-FALL), Singapore, 19–22 November 2017; pp. 531–537. [CrossRef]
18. Shao, S.; Zhang, T.; Ye, F. Jamming resource allocation aimed to data link based on simulant annealing algorithm. In Proceedings of the 2017 Progress in Electromagnetics Research Symposium-Fall (PIERS-FALL), Singapore, 19–22 November 2017; pp. 981–987. [CrossRef]

19. Xing, Q.; Jia, X.; Zhu, W. Intelligence radar countermeasure based on Q-learning. *Syst. Eng. Electron.* **2018**, *40*, 1031–1035.
20. Zhang, B.; Zhu, W. DQN Based Decision-making method of congnitive jamming against multifunctional radar. *Syst. Eng. Electron.* **2020**, *42*, 819–825.
21. Oguchi, D.; Moriya, S.; Yamamoto, H.; Sato, S. An investigation of the relationship between numerical precision and performance of Q-learning for hardware implementation. *Nonlinear Theory Its Appl. IEICE* **2022**, *13*, 427–433. [CrossRef]
22. Cardarilli, G.C.; Di Nunzio, L.; Fazzolari, R.; Giardino, D.; Matta, M.; Nannarelli, A.; Re, M.; Spanò, S. FPGA Implementation of Q-RTS for Real-Time Swarm Intelligence Systems. In Proceedings of the Conference Record—Asilomar Conference on Signals, Systems and Computers, Virtual, 1–4 November 2020.
23. Da Silva, L.M.; Torquato, M.F.; Fernandes, M.A. Parallel Implementation of Reinforcement Learning Q-Learning Technique for FPGA. *IEEE Access* **2019**, *7*, 2782–2798. [CrossRef]
24. Li, Y.; Zhu, Y.-P.; Gao, M.-G. Design of Cognitive Radar Jamming Based on Q-Learning Algorithm. *Trans. Beijing Inst. Technol.* **2015**, *35*, 1194–1199. [CrossRef]