Article

Deep Reinforcement Learning Based Decision Making for Complex Jamming Waveforms

Yuting Xu¹, Chao Wang¹, Jiakai Liang¹, Keqiang Yue¹²*, Wenjun Li¹*, Shilian Zheng² and Zhijin Zhao³

¹ Key Laboratory of RF Circuits and Systems, Ministry of Education, Hangzhou Dianzi University, Hangzhou 310018, China
² Science and Technology on Communication Information Security Control Laboratory, The No. 011 Research Center, Jiaxing 314033, China
³ The School of Communication Engineering, Hangzhou Dianzi University, Hangzhou 310018, China
* Correspondence: kqyue@hdu.edu.cn (K.Y.); liwenjun@hdu.edu.cn (W.L.)

Abstract: With the development of artificial intelligence, intelligent communication jamming decision making is an important research direction of cognitive electronic warfare. In this paper, we consider a complex intelligent jamming decision scenario in which both communication parties choose to adjust physical layer parameters to avoid jamming in a non-cooperative scenario and the jammer achieves accurate jamming by interacting with the environment. However, when the situation becomes complex and large in number, traditional reinforcement learning suffers from the problems of failure to converge and a high number of interactions, which are fatal and unrealistic in a real warfare environment. To solve this problem, we propose a deep reinforcement learning based and maximum-entropy-based soft actor-critic (SAC) algorithm. In the proposed algorithm, we add an improved Wolpertinger architecture to the original SAC algorithm in order to reduce the number of interactions and improve the accuracy of the algorithm. The results show that the proposed algorithm shows excellent performance in various scenarios of jamming and achieves accurate, fast, and continuous jamming for both sides of the communication.

Keywords: cognitive radio; intelligent jamming; deep reinforcement learning; Wolpertinger architecture; soft actor-critic

1. Introduction

As a key link in cognitive electronic warfare, cognitive electronic jamming decisions are susceptible to adversarial attacks, such as spoofing attacks or mimicry attacks, because of the inherent openness of the wireless medium [1]. In these cases, attackers can claim to be real users by imitating them, especially in device-to-device (D2D)-dependent loosely distributed infrastructures [2]. The emergence of various anti-jamming communication techniques to ensure secure transmission, such as frequency modulation (FM) technology [3], burst communication technology [4], and cognitive radio technology [5], increases the difficulties of jamming techniques. Therefore, the study of cognitive jamming decision methods is crucial.

Traditional jamming methods, including continuous jamming [6–8], reactive jamming [9,10], spoofing jamming [11,12], random periodic jamming [10], sweeping jamming [13], etc. [14], cannot perform accurate jamming or efficiently utilize jamming resources in complex battlefield communication environments.

As an important branch of mathematics, various metaheuristics algorithms are widely used in optimization problems and achieve excellent performance. For example, the hybrid sine and cosine algorithm combined with the fuzzy k-nearest neighbor method (SCA FKNN) proposed in [15] has achieved good accuracy in classification and prediction results with 10 different types of data sets. The clonal selection algorithm (CSA) and logic
mining approach were used to solve the problem of Amazon employee resource access data extraction [16]. The binary artificial bee colony method was used to optimize non-systematic weighted satisfiability [17]. Similarly, various intelligent optimization theories have been applied to the field of wireless communication jamming. Q., Z. et al. ran game theory separately on cognitive radio networks and cognitive jammers and the Nash equilibrium became the final regularly chosen behavior [18]. Fang Ye et al. proposed a taboo search artificial bee colony (TSABC) algorithm for the cognitive cooperative jamming decision, which performed better than the improved ant colony (IAC) algorithm and the artificial bee colony (ABC) algorithm, with better search capability and probability [19]. Intelligent optimization theory often requires known channel and signal priori information. However, it is difficult for the jammer to obtain this information in a battlefield environment.

Reinforcement learning (RL) has received a lot of attention in recent years as a branch of machine learning that has reached and even surpassed human levels in Go and video games. Reinforcement learning is an intelligent agent that optimizes its own action strategy through the acquisition of reinforcement signals (reward feedback) from the environment [20]. Prior information is not necessary to the agent, and the training data of reinforcement learning come from the continuous interaction with the environment. Because of these characteristics, reinforcement learning has become a powerful tool for resource optimization, jamming decisions, and other areas of intelligent jamming. Meanwhile, deep reinforcement learning (DRL) [21] technology has been employed to solve non-convex optimization problems in communication systems [22]. The development of artificial intelligence technology has provided new solutions to cognitive jamming technology [23,24]. Amuru et al. [1] used the multi-armed bandit (MAB) [25] model to develop a cognitive jammer in unknown battlefield environments, where the jammer can select physical layer parameters for flexible jamming. Zhuan Sun, S. et al. [26] combined the advantages of orthogonal matching tracking (OMP) [27] and MAB to significantly reduce the number of interactions. However, the MAB model mainly targets the static transmitter-receiver and requires re-learning for dynamic targets with changing parameters. Yangyang Li et al. [28] designed an intelligent jamming method based on reinforcement learning to combat the DRL-based user and experimentally demonstrated that the proposed algorithm can effectively limit the performance of the DRL-based anti-jamming method, which targets only the physical layer parameter of the spectrum. In [29], a multi-intelligent reinforcement learning framework with an optimal power control strategy in a dynamic game between smart jammers and training base stations was designed, and experiments showed that smart jammers with eavesdropping capabilities can seriously degrade the performance of the jammed communication parties.

When the communication parties choose to adjust the communication parameters after being jammed, the situation faced by the jammer becomes complex, i.e., the problem of a large-scale discrete action space in reinforcement learning develops. To our knowledge, there are few studies on intelligent jamming decisions in large action spaces. When the number of action spaces is very large, a straightforward idea is to serialize discrete actions and then use reinforcement learning algorithms that solve continuous actions to solve large-scale discrete action problems. In solving the problem of large-scale discrete actions in reinforcement learning, the DeepMind team proposed the Wolpertinger architecture in 2015 [30], and the authors validated the superiority of the proposed algorithm on three environments using the deep deterministic policy gradient (DDPG) algorithm in cart-pole, puddle world, and recommender systems; Haokun Chen et al. proposed a tree-structured policy gradient recommendation (TPGR) framework in 2019 [31], where a balanced hierarchical clustering tree is built over the items and picking an item is formulated as seeking a path from the root to a certain leaf of the tree.

The case considered in this paper is as follows: We investigate a soft actor-critic [32] optimization for an intelligent jamming waveform decision system. A complex jamming scenario in which communication parties choose to adjust the physical layer parameters
in order to avoid jamming is considered. In particular, the jammer learns the anti-jamming strategy for both sides of the communication procedure with as few interactions as possible in the face of a complex jamming waveform decision situation. We apply the SAC technique to solve the considered problem that occupies continuous and large-dimensional optimization variables. The contributions of the paper are mainly as follows:

A DRL-based algorithm to optimize intelligent jamming waveform decision problems is proposed. The correspondence between the communication state of the communication parties and the optimal jamming policy is established. The proposed DRL-based algorithm is based on the Markov decision process (MDP), which is utilized to deal with the problems of goal-directed learning from interaction.

To solve the considered problems of slow convergence and the long interaction times of the intelligences in a large-scale discrete action space, we propose a deep reinforcement learning smart jamming decision algorithm based on a maximum-entropy SAC that incorporates the improved Wolpertinger architecture based on the original SAC algorithm. Experiments show that the proposed algorithm has good convergence speed and high jamming accuracy for both small and large jamming action space scenarios. In some scenarios, the jamming success rate can quickly reach 100%. It is worth noting that, to the best of our knowledge, the SAC method has not yet been used in the communication jamming waveform control field.

To prevent jammers from blindly pursuing high rewards and choosing the highest power series of actions, we design a power penalty factor for the reward. To balance the exploration and exploitation dilemmas in different periods of jamming, we design the dynamic entropy coefficient.

The rest of the paper is organized as follows: In Section 2, we introduce the reinforcement learning algorithm and the intelligent jamming system model. Section 3 presents the improved SAC algorithm and the details of the algorithm. In Section 4, extensive simulation experiments are conducted to verify the performance of the algorithm proposed in this paper and the results are analyzed. Section 5 summarizes the contributions of this paper and discusses some conclusions obtained from this study.

2. System Model and Problem Formulation

In this paper, a communication jamming system in a non-cooperative scenario is considered. In the communication process, anti-jamming technologies, such as power enhancement, channel switching, and modulation switching, are adopted to suppress the effects caused by jamming signals. The jamming target in such a scenario is changing over time, making traditional MAB models for static targets and tasks unavailable.

2.1. Reinforcement Learning

The essence of reinforcement learning can be described as maximizing the rewards that can be obtained in uncertain environments. It consists of two major elements: the agent and the environment. As shown in Figure 1 [33], the agent interacts with the environment and outputs an action, , based on the state, , of the environment, and the environment goes to the next state, , and gives a feedback or reward, , under the influence of this action. The goal of the agent is to maximize the sum of the rewards,

\[ G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \]

The discount factor, , indicates less attention to the longer-term reward. The state value function can be obtained from the discounted reward function, which is used to evaluate the value of a state. In addition, the action state value function \( Q_\pi(s,a) = E[G_t | s_t = s, a_t = a] \) is introduced to represent the possible reward for taking a certain action in a certain state.
Reinforcement learning can be divided into policy-based and value-based approaches. In the policy-based approach, the policy $\pi(\theta) = (a|s)$ is assumed to be a continuous differentiable function with respect to $\theta$. The policy-based approach uses a gradient ascent method to optimize $\theta$ to maximize the gradient of the policy, $E[G]$. In the value-function-based approach, the agent continuously updates the state value function, $Q_\pi$, based on feedback and selects the action with the largest $Q$ value as the actual policy. The algorithm used in this paper is based on policy gradients and value-based actor-critic structures, which can solve both the problem of the slow convergence speed of policy-based methods and the inability of value-function-based methods to adapt to high-dimensional or continuous actions.

2.2. System Model

A cognitive electronic warfare jamming scenario is considered, in which the communication parameters of the communication parties include modulation $\{M_1, M_2, ..., M_n\}$, transmission power $\{P_1, P_2, ..., P_m\}$, and communication frequency points $\{C_1, C_2, ..., C_l\}$. A combination action of lowering the modulation order, increasing the transmission power, and switching the communication frequency is taken when the communication parties are jammed. The jammer can obtain the signal modulation mode, communication band, and approximate transmit power of the jammed party through communication reconnaissance in the actual jamming. Each jammer in this paper is equipped with a cognitive engine that can reconnoiter some basic communication parameters of the communication parties. The parameters of the jammer are also composed of the modulation mode, jamming power, and jamming frequency. The jammer generates each sample by interacting with the environment and stores it in experience pool D. Then, the algorithm is trained by the samples randomly extracted from experience pool D. Figure 2 shows the main components and data flow of SAC algorithm based on this system.

In order to conduct accurate and continuous jamming, the jammer needs to learn the complex anti-jamming strategies of the communication parties. The state, $s_t$, is composed of the communication state of the communication parties at moment $t$ and the jamming action, $a_t$, of the jamming policy to the communication parties at moment $t$. In addition, the action, $a_t$, is the jamming action for the communication parties at the next moment,
The purpose of this design is as follows: Due to the confidentiality of the communication system and the complexity of the electromagnetic environment, the communication parameters of the communication parties are difficult to detect in a short period of time. Even if communication parameters can be detected in a shorter period of time, the jamming effect may be poor due to the insufficient jamming time. Figure 3 shows the jamming timing diagram of this paper.

![Jamming timing schematic](image)

**Figure 3.** Jamming timing schematic.

### 2.3. Reward Function

In reinforcement learning, an agent interacts with and obtains a reward from the environment. The agent updates its policy based on the reward. However, the reward is often difficult to obtain in intelligent jamming scenarios. In [1], the authors used the symbol error rate (SER) as a criterion for the reward, assuming that the communication parties used TCP/IP protocol as a precondition. In this paper, we similarly assume that the jamming party can obtain approximate information about the SER of the communication parties and use it as an evaluation criterion. The reward function is therefore designed as follows:

$$
Reward = \begin{cases} 
0.3 \times 100 + a_p \times (-1) \times 3 + 60 + F_1 \times 30; & \text{if } (\text{SER} \geq X) \\
10 \times (\text{fabs}(s_c - a_c)) + 1; & \text{if } (\text{SER} < X)
\end{cases}
$$

(1)

where $a_p$ represents the jamming power, $a_c$ denotes the jamming frequency, and $s_c$ denotes the communication frequency of both communication parties. $F_1$ denotes the frequency alignment parameter, which is 1 if the frequency of the jamming action and the frequency of both communication parties are the same; otherwise, it is 0. $X$ denotes the threshold value of the SER. The above equation shows that if the SER is greater than a certain threshold, a higher reward is given, with the addition of a penalty factor and a reward factor. The penalty factor, $a_p \times (-1) \times 3$, is a penalty on the power, which is used for the purpose of preventing the jammer from blindly selecting some series of actions with maximum power. The reward factor, $F_1 \times 30$, represents the prize for channel alignment, which is designed so that channel alignment is a prerequisite for successful jamming. If the SER is less than the threshold, $X$, the absolute value of the difference between the frequency point of the jamming action and the frequency point of the communication parties is used as the reward criterion, which is designed so that the agent can learn information, even from the experience of jamming failure.

### 3. Proposed Jamming Scheme Based on SAC Model

#### 3.1. Introduction of SAC Algorithm

The SAC algorithm is an off-policy approach to optimize stochastic policy. The core idea is maximum entropy reinforcement learning, in which the goal of the agent in the SAC algorithm is to maximize the expected reward and entropy. The introduction of entropy allows the policy to be as random as possible. The goal of standard reinforcement learning is to maximize the reward sum, $\sum _t E_{(s_t, a_t) \sim \rho_{\pi}}[r(s_t, a_t)]$, where $\rho_{\pi}$ represents the distribution of policy. For in the SAC algorithm, the goal of its optimization is defined as:
where \( H(\pi(\cdot | s_t)) = -\log(\pi(a_{t+1} | s_{t+1})) \) represents the entropy part of the SAC algorithm. \( \pi(a_{t+1} | s_{t+1}) \) denotes a stochastic policy, \( \pi \), of selecting an action, \( a_{t+1} \), under a state, \( s_{t+1} \). The \( \alpha \) is the temperature coefficient, which determines the importance of entropy for the reward and controls the randomness of the optimal strategy. It becomes standard reinforcement learning at \( \alpha = 0 \) [32,34].

Compared with the proximal policy optimization (PPO) [35] reinforcement learning algorithm for online learning, the SAC algorithm follows the experience replay technique in deep Q learning (DQN) [12]. Sample utilization is important in the jamming scenarios mentioned in this paper, where each interaction is a valuable experience, and we hope that the switching strategy of the communication parties can be learned with as few interactions as possible. Compared with deep deterministic policy gradient (DDPG) [36], which is sensitive to hyperparameters and unstable in performance, the SAC algorithm integrates the three major frameworks of actor-critic, off-policy, and the maximum entropy model. The intelligent jamming algorithm sets a larger entropy coefficient in the early stage of jamming to increase the exploration of the environment and gradually reduces it in the later stage to improve the accuracy of jamming. The SAC algorithm not only greatly improves sample utilization but also has fewer hyperparameters. The addition of entropy also makes it insensitive to hyperparameters. The algorithm assigns approximately equal probabilities to actions with similar Q values, avoiding the condition where the agent repeatedly selects actions and falls into suboptimal situations. Experience has shown that the SAC algorithm surpasses other reinforcement learning algorithms in continuous control problems.

The SAC algorithm contains two kinds of networks: the policy network \( \pi_\theta(a_t | s_t) \) with parameter \( \theta \) and the value network \( Q_\theta(s_t, a_t) \) with parameter \( \theta \). The policy network outputs actions, and the value network evaluates the merits of the actions. The continuous actions output by the policy network are discretized into the parameters of the jamming actions in this study. The update and optimization of the network is usually performed using stochastic gradients, which can be found in more detail in [32].

### 3.2. Improved SAC Algorithmic Framework

In this paper, the case of the communication parties adjusting the communication parameters to avoid jamming is considered. At this time, the jamming party needs to learn a very large action. For example, there are four modulation modes (QPSK, BPSK, 64QAM, and 16QAM), thirty transmission powers (1, 2, 3, ..., and 30), and ten transmission frequencies (1, 2, 3, 4, 5, 6, 7, 8, 9, and 10) on the communication parties. From the state space and action space defined in the previous section, the state at moment \( t \) is \( S_t = S_t^r + a_t \), where \( S_t^r \) is the communication parameters of the communication parties at moment \( t \) and \( a_t \) is the jamming action of the jamming party at moment \( t \). Then, the number of \( S_t \) is 1,440,000 \((4 \times 30 \times 10 \times 4 \times 30 \times 10)\), and the number of jamming actions, \( a_t \), is 1200 \((4 \times 30 \times 10)\). It is assumed that the approximate parameter range of the communication parties has been obtained by the jammer through the preliminary communication reconnaissance, which means that the jamming action of the jammer and the communication state of the communication side are equal. The jamming action of the jammer is even larger than this number in practice.

According to the scenarios and problems raised above. This paper introduces an improved SAC algorithm based on the Wolpertinger architecture. The policy network outputs a continuous action space, \( R^n \). This output is then mapped to the discrete set \( A \). Define the function \( f_\phi: S \rightarrow R^n \), \( f_\phi(s) = \hat{a} \) to denote from the state representation space \( R^m \) to the action representation space \( R^n \). In this thesis, the continuous action of the SAC algorithm value network output is discretized in the following way:

\[
\sum E(s_t, a_t) - \rho_n[r(s_t, a_t) + \alpha H(\pi(\cdot | s_t))]
\]
\( \hat{a} = \{ \text{round}(a_m), \text{round}(a_c), \text{round}(a_p), \ldots \} \)  

(3)

In the Equation (3), \( a_m, a_c, \) and \( a_p \) denote the jamming parameters output by the SAC algorithm. The \text{round} parameter denotes the rounded mathematical symbol. This operation outputs the proto-action \( \hat{a} \), but the action may not be a valid action when mapping the continuous action to the discrete space, i.e., \( \hat{a} \in \mathcal{A} \). The K-nearest neighbor (KNN) algorithm is used to solve this problem, i.e., the function \( g: \mathbb{R}^n \rightarrow \mathcal{A} \) is defined, and Equation (4) returns the k actions that are most similar to the proto-action to form action set \( \hat{A} \):

\[
g_k(\hat{a}) = \operatorname{argmin}_{a \in \mathcal{A}} |a - \hat{a}|_2
\]

(4)

where \( a \) is the action in the jamming action library. In the jamming environment proposed in this paper, we assume that there is only one jammer, and only one action is executed each time. The second stage of the Wolpertinger architecture is to optimize the selection of actions by choosing the action with the highest score according to Equation (5):

\[
\pi_\theta(s) = \operatorname{argmax}_{a \in \hat{A}} Q_\theta(s, a)
\]

(5)

where \( Q_\theta(s, a) \) is the Q value of the state, \( s \), and the action, \( a \).

In order to speed up the convergence of the algorithm and improve the accuracy of the jamming, we propose an appropriate expansion of the set of actions, \( \hat{A} \), in this paper. A large segment of the current algorithms about smart jamming are based on channel targeting because it is the prerequisite for accurate jamming. Therefore, we propose to retain the jamming modulation pattern, \( \tilde{M} \), and jamming power, \( \tilde{P} \), of the proto-action then compose them with the jamming frequencies in the jamming library into jamming actions and add these actions to action set \( \hat{\mathcal{A}} \), i.e., \( \{(\tilde{M}, C_1, \tilde{P}), (\tilde{M}, C_2, \tilde{P}), \ldots, (\tilde{M}, C_n, \tilde{P})\} \rightarrow \hat{\mathcal{A}} \). The action set, \( \hat{\mathcal{A}} \), contains two types of actions, as in Figure 4, one class for the k-nearest neighbors found by the KNN algorithm and another composed of the channel in the jamming library plus the jamming modulation, \( \tilde{M} \), and jamming power, \( \tilde{P} \), of the proto-action. Finally, the action with the largest Q value in \( \hat{\mathcal{A}} \) is selected as the actual action of the agent. Figure 5 is the main construction of the improved SAC algorithm.

Figure 4. The composition of the action set, \( \hat{\mathcal{A}} \).
3.3. Construction of the Network

The policy network and Q network of the algorithm are constructed by the fully connected layer, and the whole algorithm contains one policy network and four Q networks. The policy network parameter is $\phi_\pi$. The four Q networks include two Q networks (Q1 network and Q2 network) and two Q target networks (Q1 target network and Q2 target network) whose parameters are $\phi_{q1}$, $\phi_{q2}$, $\phi_{q1'}$, and $\phi_{q2'}$. The use of the target network is a continuation of the fixed Q target strategy of the DQN algorithm. The purpose of using two Q networks is to solve the problem of the Q function overestimating the Q value and making the learned strategy biased. The SAC algorithm uses a pruned twin network where the Q value with the smaller value in the twin network is put into the value error function each time, as in Equations (6) and (7):

$$q = \min(Q_1(s_t, a_t; \phi_{q1}), Q_2(s_t, a_t; \phi_{q2})) \quad (6)$$

$$q' = \min(Q_{1'}(s_{t+1}, a_{t+1}; \phi_{q1'}), Q_{2'}(s_{t+1}, a_{t+1}; \phi_{q2'})) \quad (7)$$

Both the policy network and the Q network have an input layer and an output layer. The policy network contains four hidden layers with 128, 256, 512, and 128 neurons. Each hidden layer is followed by a ReLU activation function. The activation function of the output layer is sigmoid, which limits the parameter range of the action to (0, 1). The Q network contains four hidden layers with 128, 256, 512, and 128 neurons, and each hidden layer is followed by a ReLU activation function. The input of the policy network is the state, $s_t$, and the output is the mean, $\mu$, and the covariance, $\sigma$, of the Gaussian distribution. Then, the action and logarithm of its probability are obtained by sampling. Finally, a representation of the action, $a_t$, is obtained as follows:

$$a_t \sim \pi(a_t|s_t; \phi_{\pi}; \mu; \sigma) \quad (8)$$

where the mean, $\mu$, and the standard deviation, $\sigma$, of the action distribution are output from the policy network and $\pi(a_t|s_t; \phi_{\pi}; \mu; \sigma)$ denotes the policy distribution parameterized by $\phi_{\pi}$, $\mu$, and $\sigma$. Each dimension represents the parameters of the jamming waveform.

For the training process of the SAC algorithm, the agent learns the strategy by sampling batches of $\{s_t, a_t, r, s_{t+1}\}$ from the experience pool D each time. The input to the policy network is the state, $s_t$, and the output is policy $\pi$, which is the action distribution for $a_t$.

The inputs of the Q1 network and the Q2 network are state $s_t$ and action $a_t$, and the output dimension is 1, which represents the value of the state action ($Q_1$ value and $Q_2$ value). Similarly, the inputs to the target Q1 network and the target Q2 network are state $s_{t+1}$ and action $a_{t+1}$, and the output dimension is 1, which indicates the value of the state action ($Q_{1'}$ value and $Q_{2'}$ value). The Q networks are optimized using the Adam optimizer, and the Q network parameters are updated by minimizing the mean squared Bellman error (MSBE). The MSBE is defined as follows:

Figure 5. Construction of the improved SAC algorithm.

![Figure 5: Construction of the improved SAC algorithm.](image-url)
\[ L(\varphi_{q_1}) = \frac{1}{N_{\theta}} \sum_{k=1}^{N_{\theta}} (Q_1'(s_{t+1}, a_{t+1}; \varphi_{q_1'}) - Q_1(s_t, a_t; \varphi_{q_1}))^2 \]  

(9)

\[ L(\varphi_{q_1}) = \frac{1}{N_{\theta}} \sum_{k=1}^{N_{\theta}} (Q_2'(s_{t+1}, a_{t+1}; \varphi_{q_2'}) - Q_2(s_t, a_t; \varphi_{q_2}))^2 \]  

(10)

where \( Q_1'(s_{t+1}, a_{t+1}; \varphi_{q_1'}) \) denotes the target Q value output by the target Q1 network and \( Q_1(s_t, a_t; \varphi_{q_1}) \) denotes the Q value output by the Q1 network. \( N_{\theta} \) denotes the size of a minibatch. \( Q_2'(s_{t+1}, a_{t+1}; \varphi_{q_2'}) \) denotes the target Q value output by the target Q2 network, and \( Q_2(s_t, a_t; \varphi_{q_2}) \) denotes the Q value output by the Q2 network. According to Equations (9) and (10), the parameters of the Q1 network and Q2 network can be respectively updated by:

\[ \varphi_{q_1} = \varphi_{q_1} - \beta \nabla_{\varphi_{q_1}} L(\varphi_{q_1}) \]  

(11)

\[ \varphi_{q_2} = \varphi_{q_2} - \beta \nabla_{\varphi_{q_2}} L(\varphi_{q_2}) \]  

(12)

where \( \beta \) denotes the learning rate of the Q network and \( \nabla \) denotes the gradient operator. The policy network parameters are updated by minimizing the Kullback–Leibler (KL) divergence, which is defined as follows:

\[ d(\varphi_{\pi}) = \frac{1}{N_{\theta}} \sum_{k=1}^{N_{\theta}} (\lambda \log \pi(a_t|s_t; \varphi_{\pi}) - \min_{i=1,2} Q_i(s_t, a_t; \varphi_{q_i})) \]  

(13)

where \( \pi(a_t|s_t; \varphi_{\pi}) \) denotes the policy distribution of the policy network output, \( Q_i(s_t, a_t; \varphi_{q_i}) \) denotes the Q value distribution of the \( Q_i \) network, and \( \lambda \) denotes the entropy coefficient. According to Equation (13), the parameters of the policy network are updated by:

\[ \varphi_{\pi} = \varphi_{\pi} - \beta' \nabla_{\varphi_{\pi}} d(\varphi_{\pi}) \]  

(14)

where \( \beta' \) denotes the learning rate of the policy network and \( \nabla \) denotes the gradient operator.

Figure 6 shows the framework of the SAC algorithm.
3.4. Overall Algorithm Flow

In summary, the proposed improved SAC algorithm adds an improved Wolpertinger architecture to the original SAC algorithm for solving complex jamming scenarios in large-scale discrete jamming spaces. Algorithm 1 gives the pseudocode of the improved SAC algorithm proposed in this paper. The output of Algorithm 1 is the parameter values of the jamming waveform, which generates the corresponding waveform to impose the jamming.

**Algorithm 1**: The proposed improved soft actor-critic algorithm

**Initialization**: Randomly initialize the parameters of the policy network and the two Q networks. Set the experience reply D with size of 100,000.

**Input**: The current communication parameters of the communication parties and the current jamming action of the jammer

1: for episode i = 1, 2, ..., J do
2:    for step j = 1, 2, ..., N do
3:        According to the state, $s_t$, input to the policy network sampling output action, $a_t$;
4:        The proto-action, $a_t$, is input to the improved Wolpertinger architecture to obtain the actual executed action, $a_t$;
5:        Executing action $a_t$;
6:        Obtaining the next state, $s_{t+1}$, and feedback and calculating the actual reward, $r$;
7:        Storing ($s_t$, $a_t$, $s_{t+1}$, $r$) into experience pool D;
8:    Sampling the smallest batch, $N_B$, from experience pool D for training;
9:    Updating network parameters A and B for Q1 and Q2;
10:   Updating the parameters of the policy network;
11:   Updating the parameters of the target Q1 and target Q2 networks;
12:   Setting $s_t = s_{t+1}$;
13: end for
14: end for

**Output**: Jamming action for the communication parties at the next moment

3.5. Computational Complexity

In the Q1 and Q2 networks, the dimensions of the input layer, the first hidden layer, the second hidden layer, the third hidden layer, the fourth hidden layer, and the output layer are, respectively, 3, $L_1$, $L_2$, $L_3$, $L_4$, and 1. In the policy network, the dimensions of the input layer, the first hidden layer, the second hidden layer, the third hidden layer, the fourth hidden layer, and the output layer are, respectively 9, $L_1$, $L_2$, $L_3$, $L_4$, and 3. The number of actions in set $A$ is $K$. Therefore, the complexity of the improved SAC algorithm is $O[2K(3L_1 + L_1L_2 + L_2L_3 + L_3L_4 + L_4) + 9L_1 + L_1L_2 + L_2L_3 + L_3L_4 + 3L_4 ]$. Where $L_1$, $L_2$, $L_3$, and $L_4$ are, respectively, 128, 256, 512, and 128 and the symbol $O$ represents the amount of multiplying and accumulating calculations.

4. Simulation Results

To demonstrate the advantages of the proposed algorithm in intelligent jamming in this paper, we designed a large number of comparative experiments, including the jamming effects in different scenarios and the effects of algorithm parameters. The experimental results show that the proposed algorithm in this paper has excellent performance in terms of the number of algorithm interactions and jamming accuracy.
4.1. Simulation Environment

For the experimental part, we first simulated the learning behavior of the jammer for the communication parties to switch strategies in order to resist jamming. Then, the algorithm performance of the communication parties under different switching strategies and while increasing the action space of the jammer was simulated to verify the adaptability of the algorithm. Finally, the effects of some parameters in the algorithm on the results were also simulated.

To verify the learning performance of the algorithm, we assumed that the communication parties send N symbols of data every \( \Delta t \) time, and if 10% of the symbols on the receiver side were incorrect, it was considered by both communication parties that the message transmission had been jammed and they needed to change the communication strategy for anti-jamming. The jamming party did not know the specific conversion method at the beginning of the period. It was assumed that the jamming party could estimate the SER of the receiver by ACK and NACK and use it as the basis for the evaluation index of the effect of jamming. The channel model was additive white Gaussian noise (AWGN), and the signal-to-noise ratio (SNR) was 20 dB for all simulations in this paper.

Our simulation environment was Matlab and PyCharm co-simulation. Matlab has powerful engine APIs that support executing Matlab commands using other programming languages without having to initiate a Matlab desktop session. PyCharm is an efficient Python IDE, and Python has the advantages of being easy to learn, supporting multiple deep learning frameworks, and being portable. These make it feasible to use the advantages of both Matlab and Python to implement decision simulations of communication intelligence jamming algorithms. The Matlab side was responsible for communication signal generation, modulation, Gaussian channel transmission, demodulation, filtering, SER calculation, conversion strategy, and other steps related to communication signals. The PyCharm side was responsible for the overall design of the algorithm proposed in this paper, the decision making of jamming actions, data processing, environmental reconnaissance, and other steps related to the decision algorithm steps. The parameter settings in the proposed algorithm are shown in Table 1. The learning rate parameters \( \beta \) and \( \beta' \) in Table 1 were adjusted to 0.003 when the jamming action space was 20. The adjustment of the entropy coefficient is discussed in detail in Section 4.2.7.

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning rate of actor network</td>
<td>( \beta = 0.0015 )</td>
</tr>
<tr>
<td>Learning rate of critic network</td>
<td>( \beta' = 0.0015 )</td>
</tr>
<tr>
<td>Size of experience pool</td>
<td>( D = 10^6 )</td>
</tr>
<tr>
<td>Return discount factor</td>
<td>( \gamma = 0.1 )</td>
</tr>
<tr>
<td>Learning rate of soft update</td>
<td>( \tau = 0.005 )</td>
</tr>
<tr>
<td>Training batch size</td>
<td>( N_B = 100 )</td>
</tr>
<tr>
<td>Entropy coefficient</td>
<td>Dynamic</td>
</tr>
</tbody>
</table>

4.2. Comparative Experiment

In this subsection, we compare the jamming effects of the improved SAC algorithm, the SAC algorithm, the classic DQN algorithm, and the Q-learning algorithm adopted in [28] under different jamming scenarios. The DDPG algorithm performed poorly or even failed to train in each of the environments proposed in this paper. Therefore, the results are not shown in this paper. It was verified that the algorithm proposed in this paper has adaptability in different scenarios.
4.2.1. The Number of Jamming Actions Was 150

We assumed that the anti-jamming method of the communication parties was to change the communication parameters. The communication parameters included the modulation mode, transmit power, and transmission frequency. Specifically, there were three modulation modes (QPSK, BPSK, and FSK), five transmission powers (1, 2, 3, 4, and 5), and ten transmission frequencies (1, 2, 3, 4, 5, 6, 7, 8, 9, and 10). The anti-jamming method of the communication parties was as follows:

A. The initial communication parameters were that the modulation mode was QPSK, the transmission power was 1, and the communication frequency was 5. If the receiver SER exceeded 10%, they turned to procedure B.

B. The transmit power was increased. If the increase in the maximum transmit power still exceeded the threshold of SER, they turned to procedure C.

C. The communication frequency was switched according to the rule of frequency points (5, 6, 9, 10, 4, 7, 1, 3, 8, and 2), and each new frequency point chose the minimum transmit power until this frequency point to increase the maximum transmit power, which was still jammed. Then, it was considered that this frequency point was always jammed when the signal was transmitted using QPSK modulation. If there was still jamming after all frequencies had been switched, they turned to procedure D.

D. The communicator switched the modulation mode and repeated the above procedure in the new modulation mode.

The performance of each algorithm with 150 jamming actions is shown in Figure 7. Table 2 gives some details of the data during the experiment. The three columns on the right side of the table represent the number of rounds where the jamming accuracy exceeded 80% for the first time, the number of rounds where the jamming accuracy exceeded 90% for the first time, and the average jamming accuracy after the accuracy exceeded 80%.

![Figure 7. Performance comparison of each algorithm when the number of jamming actions was 150: (a) accuracy of jamming; (b) smoothed average reward.](image)

Table 2. Detailed data comparison of the performance of each algorithm when the jamming action space was 150.

<table>
<thead>
<tr>
<th>Method</th>
<th>≥80% (First)</th>
<th>≥80% (First)</th>
<th>Average (≥80%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improved-SAC</td>
<td>11</td>
<td>29</td>
<td>93.85%</td>
</tr>
<tr>
<td>SAC</td>
<td>12</td>
<td>\</td>
<td>83.26%</td>
</tr>
<tr>
<td>Q-learning</td>
<td>61</td>
<td>81</td>
<td>85.25%</td>
</tr>
<tr>
<td>DQN</td>
<td>95</td>
<td>\</td>
<td>52.4%</td>
</tr>
</tbody>
</table>
The algorithm parameters of Q-learning and DQN are, respectively, shown in Tables 3 and 4, where \( i \) denotes the number of training rounds and \( j \) denotes the number of interactions per round. The network structure of DQN is consistent with the parameters of the policy network in the SAC algorithm. The network inputs and outputs of DQN are consistent with the inputs and outputs of the policy network in the SAC algorithm. The size of Q-form in Q-learning is the number of states, \( N_s \), multiplied by the number of actions, \( N_a \). The exploration utilization factor of the DQN algorithm is as follows:

\[
1 - (0.01 + C_1 \times \exp(-1 \times j \times i \times C_2))
\]

where \( \exp \) represents the exponential notation in mathematics. \( i \) and \( j \) are the same as defined in the previous paragraph. \( C_1 \) is a constant equal to 0.98. \( C_2 \) is the attenuation coefficient. A large \( C_2 \) value is set when the jamming action space is large, and a small \( C_2 \) value is set when the jamming action space is small. The \( C_2 \) value set in this paper was taken as the value with the best result in the experiment. Equation (15) represents an exponential form of the plot, indicating that more exploration was given at the beginning of the jamming process and more exploitation was given at the end of the jamming process.

Table 3. Q-learning algorithm parameters.

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update Steps</td>
<td>0.9</td>
</tr>
<tr>
<td>Discount Factor</td>
<td>( \gamma = 0.1 )</td>
</tr>
<tr>
<td>Explore–Exploit Factor</td>
<td>( \text{Ceil} (i/10)/10 )</td>
</tr>
</tbody>
</table>

Table 4. DQN algorithm parameters.

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discount Factor</td>
<td>( \gamma = 0.1 )</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.003</td>
</tr>
<tr>
<td>Batch Size</td>
<td>( N_B = 100 )</td>
</tr>
<tr>
<td>Experience Pool Size</td>
<td>( D = 10^6 )</td>
</tr>
<tr>
<td>Explore–Exploit Factor</td>
<td>Dynamic</td>
</tr>
</tbody>
</table>

It can be seen from Figure 7a that, among the compared algorithms, the SAC-based algorithm converged faster and reached 80% accuracy quickly. The Q-learning algorithm [28] and the DQN algorithm took a long time to explore the environment when the jamming space was 150. In terms of accuracy, the improved SAC algorithm exceeded 90% jamming accuracy for the first time in the 29th round and continued to improve. It reached 100% accuracy for the first time in the 60th round and mostly stayed above 98% accuracy after that. The original SAC algorithm was not able to make any significant breakthrough after reaching an accuracy of 80%. From the previous section, it is clear that the goal of an agent in reinforcement learning is to pursue higher rewards. Figure 7b shows the smoothed average rewards of the agent during training. It can be seen that the rewards of the agent in the improved SAC algorithm in this paper all exceed or are equal to the other comparison algorithms for the same number of training steps.

4.2.2. The Number of Jamming Actions Was 600

We increased the number of jamming actions, which assumed that there were three modulation modes (QPSK, BPSK, and FSK), ten transmission powers (1, 2, 3, 4, 5, 6, 7, 8, 9, and 10), and twenty transmission frequencies (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, and 20). The anti-jamming method for both sides of the communication followed the method in Section 4.2.1, which meant that the anti-jamming proceeded by way of increasing the power first, then switching the communication frequency points,
and finally adjusting the communication mode. The frequency point started from the first one for each new modulation mode, and the transmitting power started from the minimum for each new frequency point. The frequency switching mode of the communication parties was 16, 3, 8, 2, 19, 15, 10, 12, 11, 14, 4, 1, 6, 7, 9, 5, 20, 18, 13, and 17. The initial communication parameters were as follows: the modulation mode was QPSK, the transmission power was 1, and the communication frequency was 16. Figure 8 shows a comparison of the jamming effects of each algorithm. In order to visualize the performance of different algorithms, detailed accuracy data are given in Table 5.

![Figure 8](image)

**Figure 8.** Performance comparison of each algorithm when the number of jamming actions was 600: (a) accuracy of jamming; (b) smoothed average reward.

**Table 5.** Detailed data comparison of the performance of each algorithm when the jamming action space was 600.

<table>
<thead>
<tr>
<th>Method</th>
<th>≥80% (First)</th>
<th>≥80% (First)</th>
<th>Average (≥80%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improved-SAC</td>
<td>13</td>
<td>14</td>
<td>91.95%</td>
</tr>
<tr>
<td>SAC</td>
<td>23</td>
<td>37</td>
<td>85.34%</td>
</tr>
<tr>
<td>Q-learning</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
</tr>
<tr>
<td>DQN</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
</tr>
</tbody>
</table>

It can be seen from Figure 8a that, when the number of actions increased to 600, Q-learning [28] was no longer able to perfect the Q-form in 100 rounds of interaction. DQN showed a trend of convergence by virtue of the excellent fitting ability of the neural network, but it still required a longer time to explore. The convergence speed based on the SAC algorithm was significantly better than the other algorithms. The improved SAC algorithm not only improved the convergence speed compared to the original SAC algorithm but also improved the accuracy rate by 7.7%, as shown in Table 5. At the same time, Figure 8b also clearly shows that the improved SAC algorithm proposed in this paper had a faster convergence speed.

### 4.2.3. The Number of Jamming Actions Was 1200

We continued to increase the number of jamming actions to 1200, which assumed that there were three modulation modes (QPSK, BPSK, and FSK), ten transmission powers (1, 2, 3, 4, 5, 6, 7, 8, 9, and 10), and forty transmission frequencies (1, 2, 3, 4, ..., and 40). The anti-jamming mode of the communication parties followed the method in Section 4.2.1. The frequency switching mode of the communication parties was 3, 38, 9, 35, 24, 1, 23, 12, 30, 6, 19, 25, 17, 36, 33, 7, 10, 16, 37, 40, 8, 4, 31, 22, 2, 21, 11, 28, 29, 39, 18, 5, 32, 13, 15, 26, 27, 20, 14, and 34. The initial communication parameters were as follows: the modulation mode was QPSK, the transmission power was 1, and the communication
frequency was 3. Figure 9 shows a comparison of the jamming effects of each algorithm. In order to visualize the performance of different algorithms, detailed accuracy data are given in Table 6.

![Figure 9](image-url)  
**Figure 9.** Performance comparison of each algorithm when the number of jamming actions was 1200: (a) accuracy of jamming; (b) smoothed average reward.

<table>
<thead>
<tr>
<th>Method</th>
<th>≥80% (First)</th>
<th>≥80% (First)</th>
<th>Average (≥80%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improved-SAC</td>
<td>15</td>
<td>18</td>
<td>87.83%</td>
</tr>
<tr>
<td>SAC</td>
<td>43</td>
<td>45</td>
<td>78.83%</td>
</tr>
<tr>
<td>Q-learning</td>
<td>\</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>DQN</td>
<td>\</td>
<td>\</td>
<td>\</td>
</tr>
</tbody>
</table>

Table 6. Detailed data comparison of the performance of each algorithm when the jamming action space was 1200.

It can be seen from Figure 9a that the improved SAC algorithm had a much higher convergence speed than other algorithms for 1200 actions. It is worth mentioning that both the SAC algorithm and the improved SAC algorithm in the figure exhibited regular sawtooth waveforms. This is due to the fact that when the number of actions increased to 1200 there were 40 cases of channel conversion, and not all conversion strategies could be fully learned within 100 rounds. The regular sawtooth waveforms in the figure have a period of about 12, which corresponds to exactly 1200 actions, and it takes 12 rounds to execute all these actions. The improved SAC algorithm also showed a slow upward trend in zigzagging. The original SAC algorithm also started to converge after 40 rounds, but it was slower and the accuracy was not as high as the improved SAC algorithm in the later stages. The DQN and Q-learning algorithms [28] could not converge in the first 100 rounds of training with 1200 actions, which indicates that DQN and Q-learning are not suitable for handling large-scale action spaces. Figure 9b indicates that the improved SAC algorithm in this paper showed a more obvious advantage in terms of both convergence speed and average reward when the space of jamming actions of the agent reached 1200.

4.2.4. The Number of Jamming Actions Was 20

The current jamming models about communication parties changing the communication parameters after jamming have many assumptions that the communication parties have fewer conversions. Therefore, the case where the number of jamming actions was 20 was also simulated, which meant that there were two modulation modes (QPSK and BPSK), two transmission powers (1 and 2), and five transmission frequencies (1, 2, 3, 4, and 5). The anti-jamming mode of the communication parties followed the method in Section 4.2.1. The frequency switching mode of the
communication parties was 2, 1, 3, 5, and 4. The initial communication parameters were as follows: the modulation mode was QPSK, the transmit power was 1, and the communication frequency point was 2. Figure 10 shows a comparison of the jamming effects of each algorithm. In order to visualize the performance of different algorithms, detailed accuracy data are given in Table 7.

![Figure 10](https://example.com/figure10.png)

**Figure 10.** Performance comparison of each algorithm when the number of jamming actions was 20: (a) accuracy of jamming; (b) smoothed average reward.

**Table 7.** Detailed data comparison of the performance of each algorithm when the jamming action space was 20.

<table>
<thead>
<tr>
<th>Method</th>
<th>≥80% (First)</th>
<th>≥80% (First)</th>
<th>Average (≥80%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improved-SAC</td>
<td>6</td>
<td>9</td>
<td>98.29%</td>
</tr>
<tr>
<td>SAC</td>
<td>16</td>
<td>82</td>
<td>90.99%</td>
</tr>
<tr>
<td>Q-learning</td>
<td>5</td>
<td>63</td>
<td>96.69%</td>
</tr>
<tr>
<td>DQN</td>
<td>13</td>
<td>56</td>
<td>93.70%</td>
</tr>
</tbody>
</table>

It can be seen from Figure 10a that both Q-learning [28] and DQN were better in this situation with a small number of actions. In particular, Q-learning took a short time to build the Q table and had high stability when there was a finite number of states and actions. It is worth mentioning that the improved SAC algorithm showed even better performance than Q-learning in the scenario of 20 action spaces. It exceeded 90% accuracy in the 6th round of training and reached 100% accuracy in the 9th round. This shows that the algorithm proposed in this paper has excellent performance, even in the case of small action spaces. Significantly, Figure 10b shows that the accuracy and sliding average reward curves are not uniform. The improved SAC algorithm proposed in this paper performed best in terms of accuracy, but the Q-learning algorithm performed best in terms of average rewards. The difference between the improved SAC algorithm and Q-learning in terms of jamming accuracy was not very large. However, due to the power penalty factor and channel reward factor set in the rewards, the improved SAC algorithm quickly learned the power conversion method, while the Q-learning algorithm had a higher average probability of predicting channel switching accurately, making its overall average reward greater than that of the improved SAC algorithm. The DQN algorithm also performed well in the early rewards but was unstable in the later stages.

4.2.5. Selecting 20 Jamming Actions from the Jamming Library

In practice, the jammer in many cases does not know the exact range of anti-jamming parameters of the communication parties, which means that the number of jamming actions and the number of communication parameters of the communication parties are
unequal. In this case, the jammer needs to select actions from the jamming library to perform jamming and learn the anti-jamming strategy of the communication parties. Therefore, the case of 20 switching strategies for both communication sides when the jamming action library was 600 was simulated. The frequency switching mode of the communication parties was 2, 1, 3, 5, and 4, and the other conversion parameters remained the same as in Section 4.2.4. The waveform parameters of the jamming action library were composed of modulation modes (QPSK, BPSK, and FSK), transmit powers (1–10), and communication frequencies (1–20). Figure 11 shows a comparison of the jamming effects of the different algorithms. In order to visualize the performance of different algorithms, detailed accuracy data are given in Table 8.

![Figure 11](image1.png)

**Figure 11.** Performance comparison of each algorithm when selecting 20 actions from the action library: (a) accuracy of jamming; (b) smoothed average reward.

**Table 8.** Detailed data comparison of the performance of each algorithm when 20 actions were selected from the jamming action library.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy ≥80% (First)</th>
<th>Accuracy ≥80% (First)</th>
<th>Average (≥80%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improved-SAC</td>
<td>12</td>
<td>24</td>
<td>95.31%</td>
</tr>
<tr>
<td>SAC</td>
<td>13</td>
<td>48</td>
<td>93.65%</td>
</tr>
<tr>
<td>Q-learning</td>
<td>19</td>
<td>\</td>
<td>90.59%</td>
</tr>
<tr>
<td>DQN</td>
<td>24</td>
<td>\</td>
<td>82.18%</td>
</tr>
</tbody>
</table>

As can be seen from Figure 11a, it took longer to explore to select the jamming action from the action library and learn its conversion strategy than to learn the conversion strategy directly when the size of the action library was 600. Compared with Figure 10a, it can be seen that the convergence speed was slower for the same scenario and training parameters. All four algorithms in the figure could converge in less than 100 rounds, while the improved SAC algorithm converged the fastest and achieved 100% accuracy for the first time in round 24. The original SAC algorithm reached 100% accuracy for the first time in round 48. The average reward curve in Figure 11b is consistent with the accuracy curve.

4.2.6. Selecting 150 Jamming Actions from the Jamming Library

We increased the number of conversion strategies to 150 for both communication parties based on Section 4.2.5. The frequency switching mode of the communication parties was 16, 1, 9, 3, 13, 18, 8, 7, 4, 5. Figure 12 shows a comparison of the jamming effects of the different algorithms. In order to visualize the performance of different algorithms, detailed accuracy data are given in Table 9.
Figure 12. Performance comparison of each algorithm when selecting 150 actions from the action library: (a) accuracy of jamming; (b) smoothed average reward.

Table 9. Detailed data comparison of the performance of each algorithm when 150 actions were selected from the jamming action library.

<table>
<thead>
<tr>
<th>Method</th>
<th>≥80% (First)</th>
<th>≥80% (First)</th>
<th>Average (≥80%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improved-SAC</td>
<td>6</td>
<td>22</td>
<td>88.36%</td>
</tr>
<tr>
<td>SAC</td>
<td>17</td>
<td>44</td>
<td>84.37%</td>
</tr>
<tr>
<td>Q-learning</td>
<td>\</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>DQN</td>
<td>\</td>
<td>\</td>
<td>\</td>
</tr>
</tbody>
</table>

From Figure 12a, it can be seen that the accuracy of each algorithm decreased when the jamming action range was extended to 150. However, the algorithm proposed in this paper had the fastest convergence speed and the highest accuracy rate among the compared algorithms. This conclusion is also reflected in Figure 12b.

4.2.7. The Effect of Temperature/Entropy Coefficient

As mentioned in the previous section, the entropy coefficient controls the randomness of the optimal strategy. The inclusion of the entropy coefficient not only encourages exploration but also allows the agent to learn the near-optimal behavior. The larger the entropy coefficient, the more the agent explores the environment. In [34], the authors proposed an automatic adjustment of the entropy coefficient, which means that more exploration should be given in the interval of action uncertainty. In the non-cooperative environment proposed in this paper, the jammer does not know the strategy transformation pattern of the communication parties in the initial stage of jamming. The jammer can only figure out the conversion strategy of the communication parties by randomly selecting the jamming parameters. Therefore, more exploration is needed by the jammer in the early stage of jamming. The jammer continuously learns from its historical experience as the number of jams increases. At this time, it should make full use of the environmental information already learned and give the algorithm a smaller entropy factor. In this paper, we refer to the strategy of the automatic adjustment of entropy coefficients in [34] and use the following formula for the strategy adjustment:

\[ J(\alpha) = 1 - \log(\text{ceil}((\text{eps} + 1)/100),10) \times 0.5 \]  \hspace{1cm} (16)

where \( \text{eps} \) represents the number of training rounds, \( \text{ceil} \) represents the upward rounding function, and \( \log \) is the logarithmic function.

As can be seen from Figure 13, in the scenario proposed in this paper, it should be based on different exploration probabilities in different periods to better improve the jamming accuracy and save jamming resources. Therefore, the effect is not as effective as
that of dynamic entropy coefficients when setting fixed entropy coefficients. From Figure 13, it can be seen that the jamming accuracy of the jammer does not increase after reaching a certain value when the entropy coefficient is set to 0.2 and 0.8.

![Figure 13](image)

Figure 13. The effect of the entropy coefficient on algorithm performance when the action space was 150.

4.2.8. The Effect of Discount Factor $\gamma$

In most cases of algorithms for reinforcement learning, the agent is required to consider long-term rewards. If $\gamma = 0$, the agent focuses only on maximizing the timely reward, which means that the goal of the agent is to maximize $R_{t+1}$ [33]. In the scenario proposed in this paper, the jammer gives a jamming action for the next moment based on the current communication condition, and we want the jammer to be able to successfully jam at each step, which means that the reward for each step is real and significant. There is no state where the reward is much higher than the other rewards according to the rewards covered in this paper. Therefore, the impact of different discount factors on the convergence performance of the algorithm was simulated. As shown in Figure 14, the algorithm converged fastest when $\gamma = 0.1$, followed by $\gamma = 0.5$, and finally $\gamma = 0.99$. The speed of convergence of the algorithm has been given more attention than the long-term cumulative reward. This is because in a real battlefield environment it is important to acquire more experience in fewer interactions. Therefore, the discount rate, $\gamma$, in the algorithms designed in this paper are all equal to 0.1.

![Figure 14](image)

Figure 14. The effect of discount factor $\gamma$ when the action space was 20.

4.3. Discussion

In this section, we discuss the effect of the algorithm proposed in this paper in different jamming scenarios. The initial intention of our proposed algorithm was to solve intelligent jamming problems in complex scenarios. In particular, when the situation is complex, ordinary reinforcement learning algorithms require a high number of
interactions or even fail to converge. The experiments demonstrated that the improved SAC algorithm showed better performance than other intelligent algorithms in several scenarios proposed in this paper. They showed that the maximum-entropy-based reinforcement learning algorithm can effectively balance the dilemma of exploration and exploitation in reinforcement learning. The proposed algorithm is more trusting of the value network than the general actor-critic framework. This is due to the fact that in the scenario proposed in this paper it was more difficult to accurately yield the parameters of the jamming waveform than to evaluate the value of the action. This provides a new perspective for subsequent research on reinforcement learning. We also found in our experiments that the proposed algorithm outperformed ordinary reinforcement learning algorithms, even in the case of small action spaces. However, the present algorithm has shortcomings that will be addressed in future work:

1. The parameters in the jamming action library will be further extended so that it can cope with more complex communication jamming scenarios;
2. The case where the channel is occupied will be taken into account, i.e., the scenario where the communication parties choose the channel to communicate after negotiation. Intelligent algorithms will incorporate algorithms such as case-based reasoning to avoid unnecessary exploration and further accelerate the convergence of algorithms;
3. In this paper, the use of SER as a reward was not optimal, and more realistic and simple rewards will be tried in the future. On the basis of maintaining the original model, the model structure will be further improved to enhance the robustness of the system.

5. Conclusions

In this paper, we propose a cognitive electronic jamming decision model based on an improved SAC for the intelligent jamming decision system. The Wolpertinger architecture for solving large-scale discrete action spaces was applied to the SAC algorithm so that the algorithm proposed in this paper showed good performance in scenarios with both small and large action spaces. We designed numerous comparative experiments to demonstrate the excellent performance of the proposed algorithm. Compared with ordinary reinforcement learning algorithms, the proposed algorithm in this paper was improved in terms of convergence speed and accuracy. The penalty factor of reward and the dynamic entropy coefficient were also designed to further optimize the structure of the algorithm in this paper. The algorithm proposed in this paper greatly reduces the number of interactions of the jammer and uses effective jamming resources to achieve accurate jamming. A new solution is provided for the field of intelligent communication jamming.

Author Contributions: Conceptualization, K.Y. and S.Z.; methodology, Y.X.; software, Y.X.; validation, Y.X., C.W. and J.L.; formal analysis, Z.Z.; investigation, Y.X.; resources, W.L.; data curation, C.W.; writing—original draft preparation, Y.X.; writing—review and editing, K.Y.; visualization, Y.X.; supervision, Z.Z.; project administration, W.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Natural Science Foundation of China under Grants U19B2016.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: Not applicable.

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


