SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

October 2022, Vol. 65 202204:1–202204:17
https://doi.org/10.1007/s11432-021-3347-8

Adaptive deep reinforcement learning for non-stationary environments

Jin ZHU[1*], Yutong WEI[1], Yu KANG[1*], Xiaofeng JIANG[1] & Geir E. DULLERUD[2]

[1]*Department of Automation, University of Science and Technology of China, Hefei* 230022*, China;*
[2]*Department of Mechanical Science and Engineering, University of Illinois at Urbana-Champaign, Urbana* IL 61801*, USA*

Received 2 April 2021/Revised 4 August 2021/Accepted 1 September 2021/Published online 27 September 2022

Abstract Deep reinforcement learning (DRL) is currently used to solve Markov decision process problems for which the environment is typically assumed to be stationary. In this paper, we propose an adaptive DRL method for non-stationary environments. First, we introduce model uncertainty and propose the self-adjusting deep Q-learning algorithm, which can achieve the rebalance of exploration and exploitation automatically as the environment changes. Second, we propose a feasible criterion to judge the appropriateness of parameter setting of deep Q-networks and minimize the misjudgment probability based on the large deviation principle (LDP). The effectiveness of the proposed adaptive DRL method is illustrated in terms of an advanced persistent threat (APT) attack simulation game. Experimental results show that compared with the classic deep Q-learning algorithms in non-stationary and stationary environments, the adaptive DRL method improves performance by at least 14.28% and 30.56%, respectively.

Keywords adaptive DRL, non-stationary environment, model uncertainty, exploration and exploitation problem, parameter setting, LDP

Citation Zhu J, Wei Y T, Kang Y, et al. Adaptive deep reinforcement learning for non-stationary environments. Sci China Inf Sci, 2022, 65(10): 202204, https://doi.org/10.1007/s11432-021-3347-8

1 Introduction

The Markov decision process (MDP) [1], which models the long-term interaction between an agent and its environment, is used commonly because of its generality, flexibility, and applicability to a wide range of practical problems, such as cyber-physical energy systems [2], dynamic spectrum access [3], transboundary pollution [4], and digital signal processing [5]. Despite these advantages, there are three necessary restrictions on MDP: (1) knowledge of the environmental model must be available; (2) the state of the environment must be fully observable; (3) the parameters of the environment must be stationary [6]. These restrictions limit the effectiveness of MDP until reinforcement learning (RL) is added to MDP, making it possible to compute the optimal policy in unknown environments [7]. Interacting with the environment in a manner that enables the environmental model to be estimated has been proposed as model-based RL [8], although it is difficult to model a complex unknown environment sufficiently accurately to produce effective policies [9]. Comparatively, model-free RL produces policies by learning directly from trials and errors that can be applied in arbitrary environments [10]. Moreover, this model-free RL provides a breakthrough for the second restriction in conditions in which the state of the environment is not fully-observable [11–13].

Q-learning, one of the most popular and universal model-free RL algorithms, uses the Q-value to measure the quality of the actions for a given state and adopts an ϵ-greedy policy to explore the environment, with the parameter ϵ playing an important role [14]. Selecting appropriate ϵ values to balance the exploration and exploitation enables a better optimal policy to be derived, showing that the Q-learning algorithm is a powerful one [15], e.g., a Bayesian ensemble approach is proposed to tune ϵ based on a Bayesian model combination containing the intractability of computing some posterior distribution and to balance exploration and exploitation efficiently [16].

* Corresponding author (email: jinzhu@ustc.edu.cn, kangduyu@ustc.edu.cn)

Based on this, a deep Q-learning algorithm is formulated for large-scale problems that the traditional Q-learning algorithm might fail to solve [17,18]. Using a deep neural network called the Deep Q-learning Network (DQN) to learn and approximate Q-values enables the deep Q-learning algorithm to achieve more stable and reliable learning of the environment [19]. Then, various deep reinforcement learning (DRL) algorithms emerge, e.g., the deep recurrent Q-learning algorithm. This algorithm, combining the deep Q-learning algorithm with a deep recurrent neural network, makes it possible to tackle the decision-making problem in unknown and incompletely observable environments [11]. In general, the current DRL method has successfully transcended the first two restrictions imposed on the MDP problem; nevertheless, there are still challenges such as the well-known exploration & exploitation (EE) balance problem [20] and the parameter setting problem of the Q-network [21], especially for non-stationary environments.

The EE problem has been a matter of wide concern in the application of the DRL method or other model-free RL methods [22], and many achievements that can be referred to Dong et al. [23], who combined quantum theory and RL, achieving greater learning speeds and a better balance of EE; Pathak et al. [24] proposed a curiosity formulation that enables the agent to learn more efficiently; Liu et al. [25] introduced imitation learning and improved the exploration strategy with the help of financial domain knowledge in a financial scenario.

Recently, regularization techniques on policy networks, such as L_1, L_2 regularization and dropout, have been shown to bring substantial improvement, especially on more difficult tasks [26]. For this reason, dropout, one of the promising stochastic regularization methods, has been used for discovering exploration, which can avoid the selecting of ϵ. For example, Sung et al. [27] applied this method to neuromorphic chips and find it promising using cooperating hardware chips. Xie et al. [28] proposed a neural adaptive dropout policy exploration (NADPEx) method using dropout and solved tasks with sparse rewards, while naive exploration and parameter noise fail. Taking a stationary environment as a default condition enables EE balance to be achieved through the above-mentioned accomplishments, while there is doubt that these methods still work in non-stationary environments. Moreover, for non-static environments, we require the rebalance of EE as soon as possible to catch up with environmental changes. However, to achieve this goal, prior knowledge of the environment is required such that parameters can be appropriately selected.

In general, the parameter setting of the Q-network has also remained difficult and time-consuming but easily ignored. Lack of knowledge of the environment model results in trials being required to determine the appropriate parameters of the Q-network, and we must wait until the end of the training to judge whether a set of parameters is feasible [29]. An agent can perform defectively for a period of time for two reasons: the inappropriate setting of layers and nodes in the Q-network or an unfinished training process. Under normal circumstances, we tend to take the unfinished training process as the primary reason and waste substantial time waiting for the Q-network to converge.

Generally speaking, these two challenges are widespread in the application of DRL [30,31] and become more severe when the unknown environment is non-stationary. Considering a non-stationary environment, the achieved EE balance will be broken when the environment varies. In addition, we require DRL to possess the ability to achieve a rebalancing of EE automatically as the environment changes. Moreover, the parameter selection process of the Q-network in DRL is blind, and substantial training time and reward are sacrificed waiting for the end of the Q-network training process.

In this paper, we propose a novel adaptive DRL method for general MDP problems in non-stationary environments. First, a self-adjusting deep Q-learning algorithm is proposed in which model uncertainty is introduced into the DQN. By applying dropout technology in not only the training process but also the prediction process of the Q-network, we can obtain the predictive distribution of Q-values by performing multiple forward propagations on the same input [32]. Based on this, the action is chosen by maximizing the Q-value sampled from this predictive distribution rather than the definite prediction of Q-value as in classic deep Q-learning algorithms, reflecting how well the agent learns the environment. When the environment changes, the predictive distribution varies, adjusting action selection accordingly in such a manner that EE can be rebalanced automatically. Second, a simple and feasible criterion is proposed to judge whether the Q-network is set appropriately before the end of training. This criterion, by comparing the times that the rewards are below a pre-given baseline in an observation window with the desired threshold, can reach a quick conclusion regarding whether we must reset the Q-network parameters. The optimal threshold can be calculated based on the large deviation principle (LDP) to minimize the misjudgment probability. With this criterion, substantial time and reward costs of selecting parameters of the Q-network can be saved. Finally, experiments are performed on the FlipIt game, an advanced

persistent threat (APT) attack simulation game, in both non-stationary and stationary environments, and the results show that our method can achieve better learning effects than the classic DRL methods in different scenarios. This proposed adaptive DRL method, which combines the self-adjusting deep Q-learning algorithm and the criterion of inappropriate network settings, ensuring the automatic balancing of EE and accelerating the parameter selection process, enlarges the scope of DRL methods and makes them applicable to general MDP problems in non-stationary environments.

The remainder of this paper is organized as follows. In Section 2, we introduce the framework of the deep Q-learning algorithm, and we propose the self-adjusting deep Q-learning algorithm method with model uncertainty to achieve the automatic rebalancing of EE when the environment changes. Furthermore, a criterion is given to judge whether the Q-network is set appropriately together with its quantitative analysis. The experiments are described in Section 3. Finally, we present concluding remarks in Section 4.

Notation. Let $(\mathbf{\Omega}, \mathcal{F}, P)$ be the probability space in which $\mathbf{\Omega}$ is the sample space, $\mathcal{F}$ is the sigma algebra of events, and P is the probability measure. $P(\cdot|\cdot)$ and p denote the conditional probability and probability density function, respectively. $\mathbb{E}$ is the mathematical expectation, and $\lfloor n \rfloor$ is the floor function of n. The MDP problem is a standard 5-tuple$(S, \mathcal{A}, R, P, \gamma)$ [10], where S is the set of states s_t at time t, $t = 1, 2, \ldots$, $\mathcal{A} = \{a^0, a^1, \ldots a^{N-1}\}$ is the set of actions a_t, r_t is the reward of the transition from s_{t-1} to s_t, P is the transition probability matrix, and γ is the discount factor for long-term rewards.

2 Adaptive deep reinforcement learning method

Among the existing DRL algorithms, the deep Q-learning algorithm is a classic and effective one, especially for MDP with large state spaces [33]. Without loss of generality, the contributions made to the deep Q-learning algorithm in this paper can also be migrated to other DRL algorithms. In the deep Q-learning framework [17], the long-term expected return of the action a_t under the state s_t is defined as follows:

$$Q(s_t, a_t) = \mathbb{E}\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdot | s_t, a_t\}. \tag{1}$$

Based on this, a two-layer fully connected network (Q-network) with input s_t is used to predict the state-action value $Q(s_t, a_t), a_t \in \mathcal{A}$. At each time step, after obtaining the state s_t at time point t, the agent stores this interaction history $(s_{t-1}, a_{t-1}, r_{t-1}, s_t)$ for learning the environment, then calculates $Q(s_t, a_t)$ of all actions a_t in s_t, and finally chooses the optimal action $a_t^* = \arg\max_{a_t} Q(s_t, a_t)$ with probability ϵ, or else randomly selects an action with probability $1 - \epsilon$.

Taking into account the fact that the balance of EE is determined by ϵ and the decay speed of ϵ, the classic deep Q-learning algorithm cannot guarantee the rebalance of EE as the environment changes, and it takes a long time in practice to verify if the Q-network is appropriately set, which is usually ignored by the existing achievements of DRL. Thus in this paper, we investigate how to achieve an automatic balance of EE and how to judge quantitatively whether the network parameter setting is appropriate for a non-stationary environment. This section contains three main parts: first, we introduce model uncertainty to represent the Q-network learning process for the environment and propose a self-adjusting deep Q-learning algorithm that can balance EE automatically as the learning process continues. By this means, the EE balance can be achieved automatically in both non-stationary and stationary environments. Second, we explain in detail the principle of why the learning process of such model-uncertainty based DRL is interpretive and how to extract useful information from the predictive distribution. Then, we can have a visual perspective of feeling the changes in the environment intuitively. Finally, considering the common problem that it is difficult to distinguish an unfinished training process from inappropriate Q-network settings when the reward function remains below the baseline for a time period, we propose a criterion with its quantitative analysis conducted through the LDP.

2.1 Self-adjusting deep Q-learning algorithm

For the classic deep Q-learning algorithm in which the Q-network is used to learn the historical interaction and make a prediction about the state-action value $Q(s_t, a_t)$, the ratio of EE ϵ is difficult to set appropriately for making exploring the state-action space efficiently. Thus, we describe the learning process with the model uncertainty [34] and propose the self-adjusting deep Q-learning algorithm. With the help of model uncertainty, we can measure how well the Q-network learns the environment and, meanwhile,

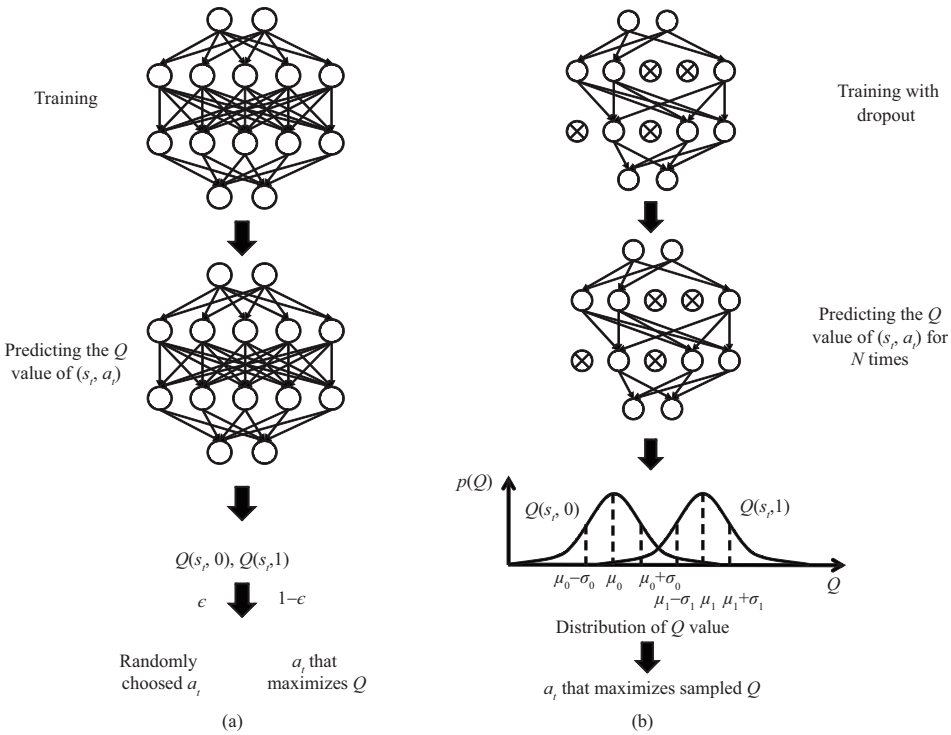

Figure 1 Process of classic deep Q-learning and self-adjusting deep Q-learning algorithms ($N = 2$). (a) Classic deep Q-learning algorithm; (b) self-adjusting deep Q-learning algorithm.

achieve an automatic balance of EE. Figure 1 shows the detailed processes of the classic deep Q-learning algorithm and the self-adjusting deep Q-learning algorithm taking $N = 2$ as an example. The classic deep Q-learning is shown in Figure 1(a) with the ratio of EE ϵ, and the self-adjusting deep Q-learning is shown in Figure 1(b).

In detail, dropout technology is applied to the Q-network not only in the training process but also in the prediction process of Q-value. With this technology, the prediction of Q-value will be a random variable obeying the Gaussian distribution [35], rather than a constant. Consequently, the Q-network calculates different predictions regarding $Q(s_t, a_t), a_t \in \mathcal{A}$, even if the input s_t is the same. To obtain the predictive distribution of $Q(s_t, a_t)$, in practice, we can perform multiple forward propagations on the same input s_t, and then calculate the mean μ^i and variance $\sigma^i, i = 1, 2, \ldots N$ under action $a_t = a^i$, where the mean is a prediction of Q-value, and the variance represents the model uncertainty. In this manner, we can determine the distribution of the Q-value, $Q(s_t, a^i) \sim \mathcal{N}(\mu^i, \sigma^i)$, which is equivalent to the theoretical predictive distribution. When choosing actions, we randomly sample $Q(s_t, a^i)$ in accordance with $\mathcal{N}(\mu^i, \sigma^i)$ and choose the action to maximize $Q(s_t, a_t)$. By this means, the greater the uncertainty of the model is, the larger the prediction variance is, and the stronger the randomness of the sampled Q-value is, with the result that the action selection plays an exploration role. In contrast, the smaller the uncertainty of the model is, the smaller the prediction variance is, and then the sampled Q-value is closer to the mean of the prediction, and the selected action is more likely to be the optimal one, which plays an exploitation role. Based on the fact that the points near the mean value will be sampled with a large probability, the Q-network gradually converges with continuous training. The proposed self-adjusting deep Q-learning algorithm is presented in Algorithm 1. In Algorithm 1, the number of forward propagations N_{forward} affects the accuracy of the estimation of model uncertainty. When N_{forward} increases, the estimation of model uncertainty becomes more accurate, while the program running time increases. According to the law of large numbers, the estimation of model uncertainty will converge to the mean in probability when N_{forward} tends to infinity. For considerations of accuracy, we usually choose N_{forward} to be no less than ten.

2.2 Perception of the environment changing

Applying dropout technology in both training and predicting yields the predictive distribution $\mathcal{N}(\mu^i, \sigma^i)$,

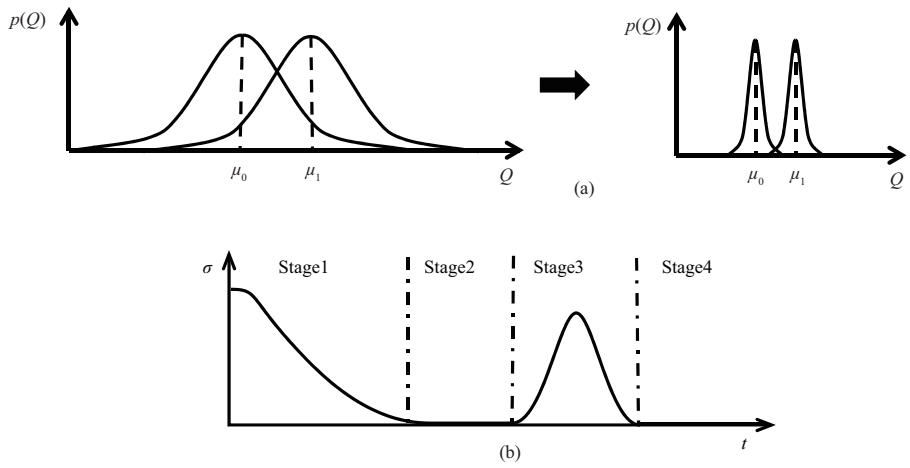

Figure 2 Perception of the environment changing through model uncertainty ($N = 2$). (a) Predictive distribution; (b) predictive variance.

Algorithm 1 Self-adjusting deep Q-learning algorithm

Initialize the Q-network;
Initialize interaction memory M according to capacity C;
Initialize the number of forward propagation N_{forward};
Initialize the environment and obtain initial state s_0;
for $t = 0, T$ **do**
  for $j = 0, N_{\text{forward}}$ **do**
    Calculate $Q^*(s_t, a_t)$ and store in q_{list};
  end for
  Calculate $Q_{\text{mean}}(s_t, a_t) = \mathbb{E}[q_{\text{list}}]$, $Q_{\text{var}}(s_t, a_t) = \text{Var}[q_{\text{list}}]$;
  Sample $Q^*(s_t, a_t)$ from $\mathcal{N}(Q_{\text{mean}}(s_t, a_t), Q_{\text{var}}(s_t, a_t))$;
  Select a_t to maximize $Q^*(s_t, a_t)$;
  Execute a_t and observe s_{t+1} and r_t;
  Store $h = (s_t, a_t, r_t, s_{t+1})$ in M;
  if the number of h is lager than C **then**
    Sample random minibatch H of h from M;
    Train the Q-network with H;
  end if
end for

which contributes substantially to achieving EE balance. Based on these distributions, in this subsection, we can visualize the Q-network learning process and propose a perceiving algorithm to reveal the environment changing.

2.2.1 *Visual analysis of the learning progress*

Figure 2 visualizes the change in the predictive distribution and variance in the learning process. Figure 2(a) shows the evolution of distribution with the learning process proceeding, and Figure 2(b) shows the variance varying with the environment. In general, the greater the model uncertainty of the Q-network is, the larger the prediction variance σ is, and the smaller the uncertainty of the model is, the smaller the prediction variance σ is.

As shown in Figure 2(a), at the beginning of Q-network training, the training data is insufficient, and thus, there are many samples of Q-network parameters and structures satisfying the distribution of training data, resulting in greater model uncertainty. In this case, randomly selected actions comprise the vast majority and play the role of exploration. With the learning process proceeding, the training data become more abundant, and the model parameters and structure become more definite, with the result that the model uncertainty gradually decreases. At this time, the optimal actions comprise the majority and then take the responsibility of exploitation.

Thus in the general cases, the variance σ of predictive Q-values gradually decreases with the convergence of the Q-network against the environment, as shown in Stage1 of Figure 2(b), eventually arriving at a constant value, as shown in Stage2 of Figure 2(b), indicating that the EE reaches balance. If σ increases suddenly, as shown in Stage3 of Figure 2(b), we can conclude that the distribution of training data for the Q-network changes substantially, indicating that the environment is changing. Based on the above

analysis, we propose a perceiving algorithm that can distinguish whether the environment changes by tracking σ.

2.2.2 *Perceiving algorithm for the environment changing*

By giving an appropriate observation window with length n, and defining a time sequence $\Psi_t = (\sigma_t^1, \sigma_t^2, \ldots, \sigma_t^N)$, $t = 0, \ldots, n-1$, we can perform a trend analysis using the Cox-Stuart test [36] to perceive the changing of the environment. Generally speaking, if Ψ_t presents an increasing trend, it can be concluded that the environment changes. Otherwise, the environment remains the same in the observing window. The perceiving algorithm for detecting the environment changing is presented in Algorithm 2. Here d is the floor function value of $\frac{n}{2}$, and n is the observation window length, which is roughly chosen as one-thousandth of the number of experimental time steps.

Algorithm 2 Perceiving algorithm for environment changing

Initialize observation window length n and significance level λ;
for $t \in n$ **do**
  Choose a_t according to Algorithm 1;
  Record the prediction variance Ψ_t;
end for
Calculate observed variance Ψ_t and Ψ_{d+t} to form a pair (Ψ_t, Ψ_{d+t}), $d = \lfloor n/2 \rfloor$;
Calculate $D_t = \Psi_t - \Psi_{d+t}$;
Calculate $N_{\text{pos}} = \sum \text{sign}(D_t)$, $N_{\text{neg}} = n - N_{\text{pos}}$;
Calculate $p(+) = N_{\text{pos}}/n$, $p(-) = N_{\text{neg}}/n$;
Calculate $k = \min(N_{\text{pos}}, N_{\text{neg}})$;
Calculate $p_{\text{sig}} = \binom{n}{k} 0.5^k 0.5^{n-k}$;
if $N_{\text{pos}} > N_{\text{neg}}$ and $p_{\text{sig}} < \lambda$ **then**
  Ψ_t has an increasing trend and the environment changes;
else
  Ψ_t has no increasing trend and the environment has no change;
end if

Remark 1. In most RL studies, the task under consideration is assumed to be stationary, allowing all the RL convergence properties. However, this assumption does not hold in real-world scenarios in which the environment tends to be non-stationary rather than stationary, and detection of non-stationary status has presented an interesting issue in RL. For example, a theoretically-grounded change detection mechanism is proposed by Canonaco et al. [37] to detect non-stationeries related to performance degradation during the learning process of the RL algorithm. As a comparison, Algorithm 2 provides a visible method for perceiving environmental changes by examining the predictive distribution, enabling an intuitive understanding of the moment and severity of environmental changes. When environmental changes occur in practical applications, there is no need to change the parameters of dropout. The proposed self-adjusting Q-learning algorithm can achieve an automatic balance of EE and generate the corresponding optimal strategy for a new environment.

2.3 Criterion of inappropriate network settings

The above proposed self-adjusting deep Q-learning algorithm ensures the balance of EE automatically and contributes to reducing the number of hyper-parameters for EE balance. Furthermore, the learning process is visualized in such a manner that our agent can perceive changes in the environment. However, a common problem persists in the application of the DRL method where the reward is less than a pre-given baseline R for a period of time steps. Here R is the baseline of reward given in accordance with practical MDP problems. The reason for this phenomenon can be summarized as two possible factors: an inappropriate setting of layers and nodes in the Q-network or an unfinished training process. Under normal circumstances, we tend to take the unfinished training process as the primary reason and waste substantial time waiting for the Q-network to converge. In this paper, we conduct a deep investigation and provide the following criterion for quantitative analysis.

Define a window with length n to observe the time sequence of reward, and use $\xi_t(\omega), t = 1, 2, \ldots, n, \omega \in \Omega$ as the random variable sequence of reward when the training process is finished and $\eta_t(\omega), t = 1, 2, \ldots, n, \omega \in \Omega$ as the random variable sequence of reward when the training process is unfinished [38]. Let the probability of an inappropriate setting for the Q-network be p_e, and let the probability of the event that the reward is less than the baseline R for the finished training process be p_d. Assuming that

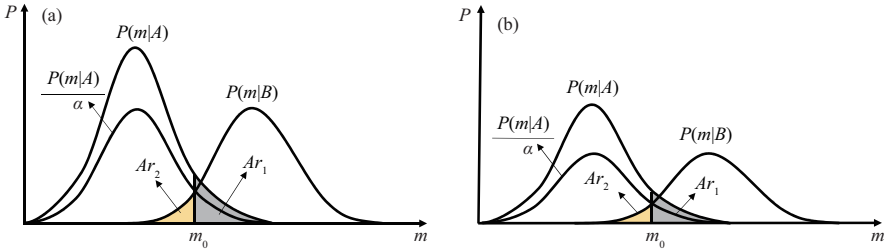

Figure 3 (Color online) Analysis of misjudgment probability based on Bayesian method ($n_1 > n_2$). (a) Misjudgment probability with larger observation window $n = n_1$; (b) misjudgment probability with smaller observation window $n = n_2$.

the random variable sequences $\xi_t(\omega)$ and $\eta_t(\omega)$ are independent and that both are subject to the Bernoulli distribution, their distributions are, respectively,

$$\xi_t(\omega) \sim \begin{pmatrix} 0 & 1 \\ p_e & 1-p_e \end{pmatrix}, \tag{2}$$

$$\eta_t(\omega) \sim \begin{pmatrix} 0 & 1 \\ p' & 1-p' \end{pmatrix}, \tag{3}$$

where $p' = p_d + p_e - p_d p_e$. Here, zero indicates that the reward is less than R, while one indicates that the reward is greater than or equal to R.

Considering an observation window with length n, let m be the number of times that the reward is less than R. Events A and B indicate that the training process is unfinished and finished, respectively. For the case in which the reward is less than R, event B corresponds to the inappropriate setting of layers and nodes in the Q-network, since only two factors account for this. Consequently, we have the following probability distribution:

$$P(m|A) = \binom{n}{m} p_e^m (1-p_e)^{n-m}, \tag{4}$$

$$P(m|B) = \binom{n}{m} p'^m (1-p')^{n-m}, \tag{5}$$

where $P(m|A)$ and $P(m|B)$ are the conditional probability that the event of reward less than R occurs m times under the unfinished and finished training process within the observation window.

According to the Bayesian formula [39], the following conditional probability formula can be obtained:

$$P(A|m) = \frac{P(m|A)P(A)}{P(m)}, \tag{6}$$

$$P(B|m) = \frac{P(m|B)P(B)}{P(m)}. \tag{7}$$

Using the minimum deviation estimate [40] and letting $\alpha = \frac{P(B)}{P(A)}$, the condition for the fact that the training process is unfinished can be defined as

$$P(B|m) > P(A|m) \to P(m|B) > \frac{P(m|A)}{\alpha}. \tag{8}$$

The principal approach of distinguishing the reason for reward that is less than R is to determine whether Eq. (8) holds. "Yes" indicates, roughly, that Q-network is inappropriately set. "No" indicates an unfinished training process. Figure 3 shows the distribution function of the conditional probabilities, providing an intuitive understanding of (8). Here, Figure 3(a) corresponds to the distribution function with greater observation window length, and Figure 3(a) corresponds to that with greater observation window length.

From Figure 3, we can observe that there is shaded overlapping areas Ar_1 and Ar_2 between $P(m|B)$ and $\frac{P(m|A)}{\alpha}$. The shaded areas Ar_1 and Ar_2 represent P_{mis}, indicating the probability of misjudgment of these two factors:

$$P_{\text{mis}} = P(A)Ar_1 + P(B)Ar_2. \tag{9}$$

Typically, if we choose a larger observation window $n = n_1$, i.e., we observe the training process for a longer time, then we can make more accurate judgments such that P_{mis} decreases, and this corresponds to a smaller shaded overlapping area, as shown in Figure 3(a). In contrast, if we choose a smaller observation window $n = n_2$, i.e., we observe the training process for a shorter time, then P_{mis} increases, corresponding to a larger shaded overlapping area, as shown in Figure 3(b). Generally speaking, this shaded area decreases with an increase in n. Under ideal circumstances, we would expect there to be no intersection of $P(B|m)$ and $P(A|m)$ to avoid misjudgment. However, this is almost impossible since we cannot let the length observation window be infinite. If one wants to reduce the probability of misjudgment further, then increasing n is a good choice, but this will have the consequence of increasing observing time and wasting substantial time waiting.

According to the above analysis, taking into account a limited observation window length n, we aim to find the optimal threshold m_0, minimizing the misjudgment probability P_{mis}. For this purpose, we present the following theorem.

Theorem 1. Taking into account the observation window with fixed length n, the misjudgment probability P_{mis} is given in (9). The optimal boundary m_0 that can minimize the misjudgment probability is calculated using the following equation:

$$\inf_{m_0}\{P_{\text{mis}}\} = P(A)\inf_{m_0}\left\{\mathrm{e}^{-nl_{-\xi}(-\frac{n-m_0}{n})+o(\frac{1}{n})} + \alpha\mathrm{e}^{-nl_{\eta}(\frac{n-m_0}{n})+o(\frac{1}{n})}\right\}, \tag{10}$$

where $l_{-\xi}(-\frac{n-m_0}{n})$ and $l_{\eta}(\frac{n-m_0}{n})$ are convergence rate functions of the probability of misjudgment satisfying:

$$\begin{aligned} l_{-\xi}\left(-\frac{n-m_0}{n}\right) &= \frac{m_0-n}{n}\log\left(\frac{m_0-n}{n(1-p_e)}\right) + \frac{2n-m_0}{n}\log\left(\frac{2n-m_0}{np_e}\right), \\ l_{\eta}\left(\frac{n-m_0}{n}\right) &= \frac{n-m_0}{n}\log\left(\frac{n-m_0}{n(1-p')}\right) + \frac{m_0}{n}\log\left(\frac{m_0}{np'}\right). \end{aligned} \tag{11}$$

Proof. According to (3), the moment-generating functions of random variables $\xi_t(\omega)$ and $\eta_t(\omega), \omega \in \Omega$ can be expressed, respectively, as

$$g_{\xi} = \mathbb{E}\{\mathrm{e}^{\theta}\xi_t(\omega)\} = p_e + (1-p_e)\mathrm{e}^{\theta}, \tag{12}$$

$$g_{\eta} = \mathbb{E}\{\mathrm{e}^{\theta}\eta_t(\omega)\} = p' + (1-p')\mathrm{e}^{\theta}, \tag{13}$$

where θ is the parameter of the moment-generating function.

According to the Legendre transformation and Cramés theorem in LDP [41, 42],

$$P\left(\omega : \frac{\xi_1(\omega)+\xi_2(\omega)+\cdots+\xi_n(\omega)}{n} \geqslant b\right) = \mathrm{e}^{-nl_{\xi}(b)+o(\frac{1}{n})}, \tag{14}$$

$$P\left(\omega : \frac{\eta_1(\omega)+\eta_2(\omega)+\cdots+\eta_n(\omega)}{n} \geqslant b\right) = \mathrm{e}^{-nl_{\eta}(b)+o(\frac{1}{n})}, \tag{15}$$

where

$$\begin{aligned} l_{\xi}(b) &= b\log\left(\frac{b}{1-p_e}\right) + (1-b)\log\left(\frac{1-b}{p_e}\right), \\ l_{\eta}(b) &= b\log\left(\frac{b}{1-p'}\right) + (1-b)\log\left(\frac{1-b}{p'}\right). \end{aligned} \tag{16}$$

According to the definitions of $\xi_t(\omega)$ and $\eta_t(\omega)$, we know that $\xi_1(\omega)+\xi_2(\omega)+\cdots+\xi_n(\omega)$ and $\eta_1(\omega)+\eta_2(\omega)+\cdots+\eta_n(\omega)$ indicate the number of times that rewards are greater than or equal to R, when the training process is finished and unfinished, respectively. For the boundary m_0, the area of the shaded part of Figure 3 can be expressed as

$$Ar_1 = P\left(\omega : \tfrac{\xi_1(\omega)+\xi_2(\omega)+\cdots+\xi_n(\omega)}{n} \leqslant \tfrac{n-m_0}{n}\right), \tag{17}$$

$$Ar_2 = P\left(\omega : \tfrac{\eta_1(\omega)+\eta_2(\omega)+\cdots+\eta_n(\omega)}{n} \geqslant \tfrac{n-m_0}{n}\right). \tag{18}$$

Substituting the above equations into (9), we can obtain the theoretical lower bound of P_{mis}:

$$\begin{aligned}
&\inf_{m_0}\{P_{\text{mis}} = P(A)Ar_1 + P(B)Ar_2\}\\
&\quad= \inf_{m_0}\Bigg\{P(A)P\left(\omega : \frac{\xi_1(\omega)+\xi_2(\omega)+\cdots+\xi_n(\omega)}{n} \leqslant \frac{n-m_0}{n}\right)\\
&\qquad+ P(B)P\left(\omega : \frac{\eta_1(\omega)+\eta_2(\omega)+\cdots+\eta_n(\omega)}{n} \geqslant \frac{n-m_0}{n}\right)\Bigg\}\\
&\quad= \inf_{m_0}\left\{P(A)\mathrm{e}^{-nl_{-\xi}(-\frac{n-m_0}{n})+o(\frac{1}{n})} + P(B)\mathrm{e}^{-nl_{\eta}(\frac{n-m_0}{n})+o(\frac{1}{n})}\right\},
\end{aligned} \tag{19}$$

where $l_{-\xi}(-\frac{n-m_0}{n})$ and $l_{\eta}(\frac{n-m_0}{n})$ are convergence rate functions of the probability of misjudgment.

Bearing in mind that $\alpha = \frac{P(B)}{P(A)}$, we rewrite (19) as

$$\inf_{m_0}\{P_{\text{mis}}\} = P(A)\inf_{m_0}\left\{\mathrm{e}^{-nl_{-\xi}(-\frac{n-m_0}{n})+o(\frac{1}{n})} + \alpha\mathrm{e}^{-nl_{\eta}(\frac{n-m_0}{n})+o(\frac{1}{n})}\right\}. \tag{20}$$

We can see from (20) that for seeking the optimal threshold m_0, only α is required, while the probabilities $P(A)$ and $P(B)$ are not necessarily known. Here, α is given by the prior knowledge of $\frac{P(B)}{P(A)}$, which characterizes the ratio of the probability that the training process is finished to the probability that the network is inappropriately set.

To use this criterion, we first obtain $m_{\text{statistical}}$, the number of times that the reward is less than a baseline R within an observation window, and then compare it with the threshold m_0, which is obtained using Theorem 1. If $m_{\text{statistical}} < m_0$, then the factor that causes this can be considered as an unfinished training process. Otherwise, if $m_{\text{statistical}} \geqslant m_0$, then the factor that causes this can be considered as an inappropriate setting of layers and nodes in the Q-network. The specific judgment algorithm is given in Algorithm 3. Based on the LDP, we know that m_0, calculated using Theorem 1, can minimize the probability of misjudgment.

Algorithm 3 Judgment algorithm for the inappropriate network setting

```
Initialize observation window length n, baseline R, and counter m_statistical;
Calculate m_0 according to (20);
for t ∈ n do
    Choose a_t according to Algorithm 1;
    Get reward r_t;
    if r_t < R then
        m_statistical = m_statistical + 1;
    end if
end for
if m_statistical ⩾ m_0 then
    The setting of Q-network is inappropriate;
    Reset the parameter of Q-network;
else
    Continue the training process of Q-network;
end if
```

3 Experiments

In this section, we investigate the performance of the proposed adaptive DRL method in APT-like scenarios [43]. The investigation can be divided into two aspects. First, we verify the advantage of our method in a non-stationary environment, with the classic deep Q-learning algorithm and the Q-learning algorithm chosen as comparisons. Second, for a stationary environment, we examine the improvements in the performance of our method compared with the classic deep Q-learning algorithm.

3.1 FlipIt game and the MDP model

FlipIt is designed to model APT-like scenarios in which the defender and attacker compete for control of a sensitive resource (for instance, a password, cryptographic key, computer system, or network) [44].

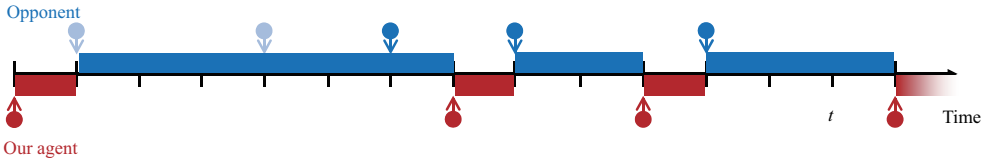

Figure 4 (Color online) The process of LM FlipIt game.

Table 1 State transition probability

$s_t s_{t+1}$	$(0, \tau_{t+1}^1 = \mathrm{LM}_t^1)$	$(\mathrm{LM}_t^0 + 1, \tau_{t+1}^1 = \tau_t^1 + 1)$
$(\mathrm{LM}_t^0, \tau_t^1), a = 1$	1	0
$(\mathrm{LM}_t^0, \tau_t^1), a = 0$	0	1

In the last move (LM) version of the FlipIt game, two players compete for a shared resource and can neither perceive who is currently in control nor observe all of the opponent's actions. The process of the LM FlipIt game is described in Figure 4; for each time point the player chooses to flip, he or she will obtain control of the resource and determine the time since the opponent's last flipping, but has to pay the flipping cost simultaneously. Here, the blue rectangle represents the time period that the opponent is in control, while the red rectangle represents the time period that our agent is in control. Owing to the LM observation mechanism, the opponent's flipping represented by the light blue circle can never be determined, but the opponent's flipping represented by the darker blue circle can only be observed with our own flipping. The game score is calculated according to the following rules: the final score equals the cumulative rewards for being in control of the resource minus the total flipping costs, and the player with the higher score wins.

We focus on the discrete version of LM FlipIt with P_0 being our agent and P_1 being the opponent. LM_t^i is the time since P_i's last flipping at time point t. P_0 has full knowledge of LM_t^0, but not always a clear understanding of LM_t^1. Then, P_0 must decide whether to flip or not at each time point t based on the information received, which can be modeled as an MDP $(S, \mathcal{A}, R, P, \gamma)$ as follows [45].

(1) State. At each time point t, our agent P_0 possesses the information regarding two parts: our own last move time $\mathrm{LM}_t^0 \in (0, +\infty))$ and $\tau_t^1 \in (0, +\infty))$, which is the time since the opponent P_1's known last flipping. In the MDP model under consideration, the state s_t can be defined as $s_t = (\mathrm{LM}_t^0, \tau_t^1) \in S$.

(2) Action. a_t is the action at time t, which has two possible choices 1 or 0 corresponding to flipping or not flipping, respectively. If our agent chooses action $a_t = 1$, then P_0 will take control and determine the time since P_1's real last flipping LM_t^1, but will have to pay the flipping cost c. Meanwhile, P_0 will gain the benefit g for each time step in control.

Otherwise, if $a_t = 0$, the ownership of control maintains unchanged, and P_0 will pay no cost or receive the information of P_1's last flipping.

(3) State transition probability. $P(s_{t+1}|s_t, a_t)$ is the transition probability from s_t to s_{t+1}, when choosing action a_t at time point t. The state transition probability table is given in Table 1.

(4) Reward. r_t is the reward function at time point t, with three situations being taken into consideration.

(i) $a_t = 0$, which indicates that we choose not to flip, with the result that we neither need to pay any cost nor gain any benefit.

(ii) $a_t = 1$ and $\mathrm{LM}_t^0 \leqslant \tau_{t+1}^1$, that is, we find that the control was originally in our hands after flipping, with the result that we will not benefit from this flipping but must pay cost instead.

(iii) $a_t = 1$ and $\mathrm{LM}_t^0 > \tau_{t+1}^1$, which indicates that through this flip, we regain control from P_1, thereby obtaining the benefit and paying the cost simultaneously.

To summarize, the reward function r_t is as follows:

$$r_t = \begin{cases} 0, & \text{if } a_t = 0, \\ -c, & \text{if } a_t = 1, \text{ and } \mathrm{LM}_t^0 \leqslant \tau_{t+1}^1, \\ \dfrac{\mathrm{LM}_t^0 - \tau_{t+1}^1 + 1 - c}{\mathrm{LM}_{t+1}^0 - \tau_{t+1}^1}, & \text{if } a_t = 1, \text{ and } \mathrm{LM}_t^0 > \tau_{t+1}^1. \end{cases} \tag{21}$$

In the reward function of the third situation, a coefficient $1/(\mathrm{LM}_{t+1}^0 - \tau_{t+1}^1)$ is added, because maximizing the cumulative reward in the ideal situation requires flipping immediately after the opponent

flips, i.e., $\mathrm{LM}_{t+1}^0 - \tau_{t+1}^1 = 1$. The introduction of this coefficient encourages the player to flip as soon as possible after the opponent does.

(5) Discount factor. $\gamma \in [0, 1]$ represents the discount factor for the long-term gains of FlipIt.

3.2 Environmental settings

In this experiment, we set a discount factor as $\gamma = 0.9$ and the ending time step of the FlipIt game as 500000. For display convenience, we average the scores of every 100 time steps to replace the large number of data points on single time steps. The benefit of being in control of the shared resource for one time step is set as one, and the cost of flipping c is set as five. The strategies of the opponent include two renewal strategies [46].

• Periodic. The opponent selects a time point randomly from $[0, \delta]$ to flip, and then flips every δ time steps.

• Exponential. The interval between two flippings follows an exponential distribution with rate φ, whose probability density function is

$$p(x) = \begin{cases} \varphi \mathrm{e}^{-\varphi x}, & x > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{22}$$

The stationary strategy of the opponent corresponds to the stationary environment, while the non-stationary strategy of the opponent corresponds to the non-stationary environment; for example, δ and φ change in the middle of the game.

3.3 Our agent vs. a non-stationary strategy opponent

We test our method against an opponent's non-stationary periodic and exponential strategies, respectively, where the parameters will change in the middle of the game.

3.3.1 *Our agent vs. a non-stationary periodic opponent*

The opponent adopts a periodic strategy with δ and changes the strategy to δ' at time step 250000, where $\delta, \delta' \in (10, 100)$. The parameter settings of the three methods are as follows.

• The Q-learning algorithm. The Q-table method is adopted and used to record the value of the state-action. Each step in each episode updates the Q-table according to the Bellman formula. The probability of choosing not flipping when the Q-values are the same is 0.5; the ϵ of the classic Q-learning algorithm is initialized as 0.5, and the decay parameter for ϵ as 0.01.

• The classic deep Q-learning algorithm. The Q-network consists of two fully connected layers with 110 hidden states; the parameters of the Q-network are initialized with a Kaiming uniform distribution; during training of the Q-network, the minibatch size of the training samples is set to 64; the learning rate is 0.0005; the ϵ of the classic deep Q-learning algorithm is initialized as 0.5 and decreased by 0.1 every 100000 time steps.

• The adaptive DRL method. The structure and training settings of the Q-network in our method are the same as above; the number of forward propagations when calculating the Q-values is $N_{\text{forward}} = 30$; the probability of dropout is 0.5; the length of the observation window for judging an inappropriate Q-network setting is $n = 800$ and $\alpha = 0.67$, which is given by the prior knowledge that $\frac{P(B)}{P(A)} = \frac{2}{3}$; and the baseline is $R = 0$.

For both the Q-learning algorithm and the classic deep Q-learning algorithm for which the ϵ-greedy exploration method is used, a decay parameter is necessary because if the random behavior continues with a small but fixed probability, the total regret value increases linearly. For this reason, a decay parameter is added, making ϵ decrease over time, with the result that the total regret value is sublinear logarithmic with respect to the time step. Meanwhile, ϵ marks the trade-off between exploration and exploitation. In the beginning, we make ϵ a little bit greater to enable it to take big leaps and learn more. As we learn about future rewards, ϵ must decay to enable us to exploit the higher Q-values we have found.

Figures 5 and 6 show the average rewards against a non-stationary periodic opponent who changes strategy slightly and significantly. Here, Figures 5(a) and 6(a) correspond to the classic deep Q-learning algorithm being used, Figures 5(b) and 6(b) correspond to the Q-learning algorithm, and Figures 5(c) and

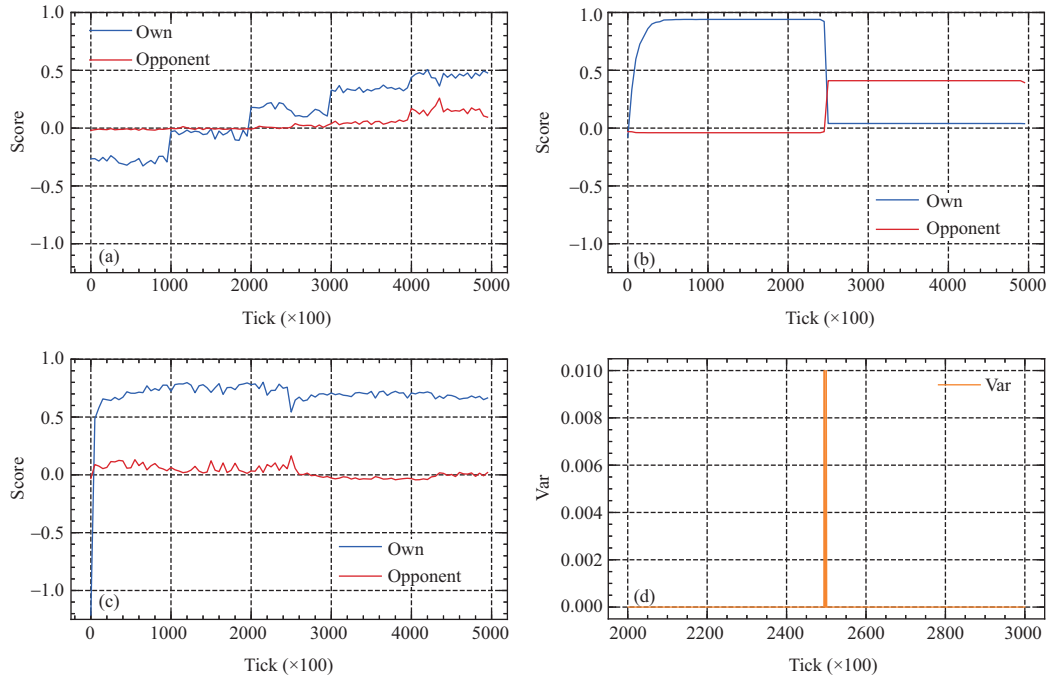

Figure 5 (Color online) Average rewards and predictive variance against a non-stationary periodic opponent ($\delta = 100 \to \delta' = 50$). (a) Classic deep Q-learning algorithm; (b) Q-learning algorithm; (c) adaptive DRL method; (d) predictive variance.

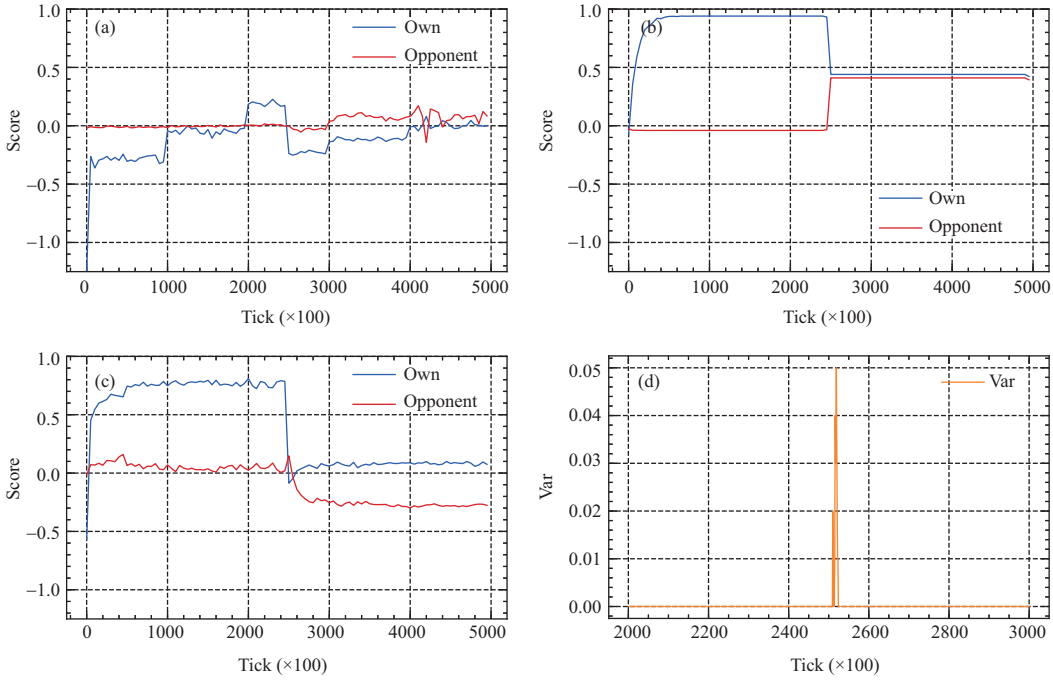

Figure 6 (Color online) Average rewards and predictive variance against a non-stationary periodic opponent ($\delta = 100 \to \delta' = 10$). (a) Classic deep Q-learning algorithm; (b) Q-learning algorithm; (c) adaptive DRL method; (d) predictive variance.

6(c) correspond to the adaptive DRL method along with Figures 5(d) and 6(d) yielding the corresponding predictive variance.

If the opponent's strategy changes slightly, for example, if δ changes from 100 to 50, we can see from Figure 5 that our method can rebalance the EE and converge to the new optimal strategy quickly. Meanwhile, with ϵ not decayed to zero, the classic deep Q-learning algorithm can still continue to learn the opponent's new strategy and finally converge, though the convergence speed is not satisfying. The Q-learning algorithm cannot take effect at all if the opponent changes strategy because its ϵ has decayed

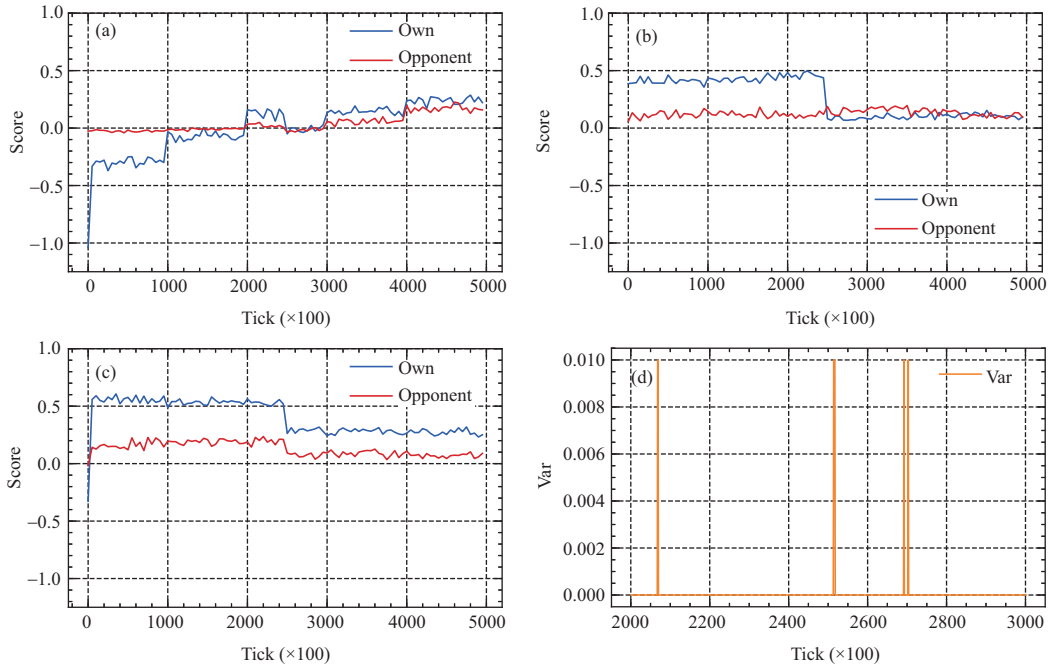

Figure 7 (Color online) Average rewards and predictive variance against a non-stationary exponential opponent ($\varphi = 0.02 \to \varphi' = 0.06$). (a) Classic deep Q-learning algorithm; (b) Q-learning algorithm; (c) adaptive DRL method; (d) predictive variance.

to zero.

In contrast, if the opponent's strategy changes significantly, for example, if δ changes from 100 to 10, we can see from Figure 6 that our method can still rebalance the EE successfully and converge to the optimal strategy quickly. However, the classic deep Q-learning algorithm fails to learn the opponent's new strategy and cannot converge even if ϵ has not decayed to zero. Similarly, the Q-learning algorithm fails for the same reason.

3.3.2 *Our agent vs. a non-stationary exponential opponent*

The opponent uses an exponential strategy with parameter φ, and changes to a new parameter φ' at time step 250000, where $\varphi, \varphi' \in (0.02, 0.1)$. According to [46], there is difficulty in learning the opponent's strategy as a result of the randomly sampled spacing between two consecutive flippings; that is, τ_t^1 causes substantial perturbation to the deep Q-learning algorithm because of its high randomness, and thus only LM_t^0 is chosen as the input of the Q-network similar to [46]. The parameter settings of the three methods are as follows.

- The Q-learning algorithm. The probability of choosing not flipping when the Q-values are the same is 0.9; ϵ is initialized as 0.8, and the decay parameter for ϵ is 0.0001.
- The classic deep Q-learning algorithm and the adaptive DRL method. The settings of these two algorithms are the same as that in the non-stationary periodic strategy.

Figures 7 and 8 show the average rewards against a non-stationary exponential opponent who changes strategy slightly and significantly, respectively. Figures 7(a) and 8(a) correspond to the classic deep Q-learning algorithm, Figures 7(b) and 8(b) correspond to the Q-learning algorithm, and Figures 7(c) and 8(c) correspond to the adaptive DRL method, with Figures 7(d) and 8(d) providing the corresponding predictive variance.

For the opponent's non-stationary exponential strategy, when the parameter changes slightly from $\varphi = 0.02$ to $\varphi' = 0.06$, as shown in Figure 7, a similar conclusion can be drawn as in Figure 5. When the parameter changes significantly from $\varphi = 0.02$ to $\varphi' = 0.1$, only our method can converge to a new optimal strategy with the other two methods failing, as shown in Figure 8.

3.3.3 *Experimental results analysis*

Comparing the predictive variance in Figures 5–8 shows some fluctuations for the exponential strategy but not for the periodic strategy because the period between the opponent's flippings is sampled randomly

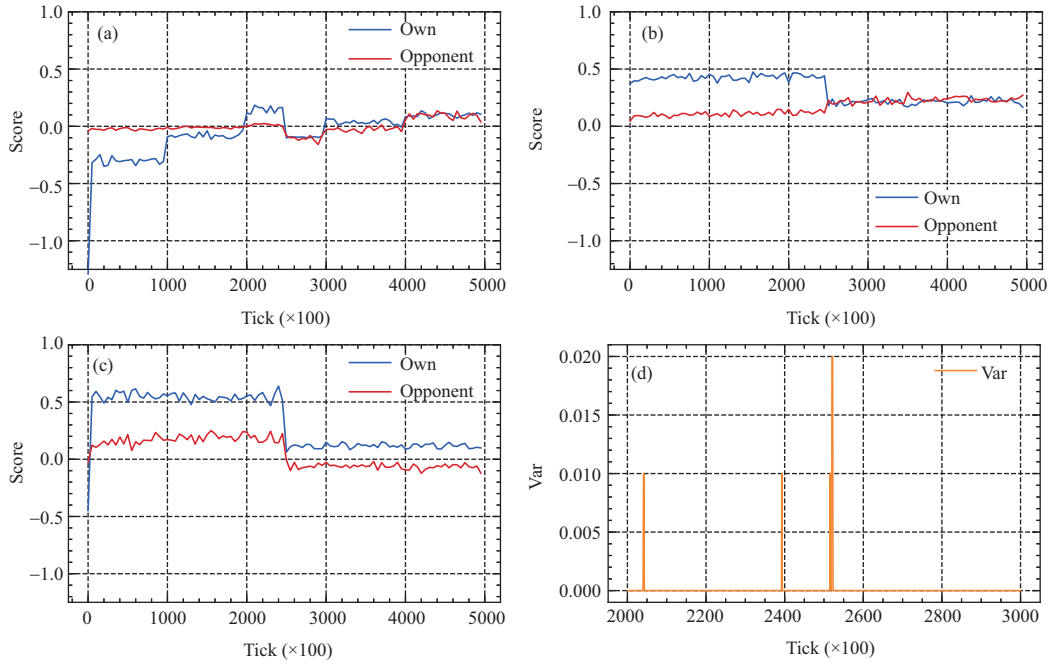

Figure 8 (Color online) Average rewards and predictive variance against a non-stationary exponential opponent ($\varphi = 0.02 \to \varphi' = 0.1$). (a) Classic deep Q-learning algorithm; (b) Q-learning algorithm; (c) adaptive DRL method; (d) predictive variance.

Table 2 Against a non-stationary opponent with different algorithms

Non-stationary opponent		Score of our agent			Average improvement of adaptive DRL (%)	
		Deep Q-learning	Q-learning	Adaptive DRL	vs. Deep Q-learning	vs. Q-learning
Periodic strategy	$\delta: 100 \to 50$	0.46252	0.43824	0.66844	185.08	79.20
	$\delta: 100 \to 10$	0.01927	0.03984	0.08202		
Exponential strategy	$\varphi: 0.02 \to 0.06$	0.24109	0.22082	0.27298	14.28	16.79
	$\varphi: 0.02 \to 0.1$	0.10034	0.10524	0.11572		

for the exponential strategy. Furthermore, it can be seen from Figures 5(d) and 6(d) that at the exact time point 250000, there is a jump of predictive variance reflecting the changing of the opponent's periodic strategy. As for the exponential strategy, things are complicated and there are fluctuations in the predictive variance curve. The occurrence of these fluctuations might interfere with the learning process of the Q-network. For the case in which the opponent changes strategy significantly, the peaks around time step 250000 are the highest, as shown in Figure 8(d), leading us to regard the opponent as changing strategy at that time point. However, if the opponent's strategy changes slightly from 0.02 to 0.06, it is difficult to determine exactly when the opponent changes strategy, as shown in Figure 7(d).

For a quantitative investigation of Q-learning, deep Q-learning, and the adaptive DRL method, we list the scores of our agent in Table 2. It can be seen that compared with the classic deep Q-learning algorithm and the Q-learning algorithm, the proposed adaptive DRL method improves the performance by at least 14.28%. As to periodic strategy, there is also a significant improvement because only our method can converge while the other two algorithms fail.

From an investigation of these experimental results in a non-stationary environment, the following conclusion can be drawn.

• When the opponent's strategy changes significantly, only our method can converge to the new optimal strategy, while the classic deep Q-learning algorithm and the Q-learning algorithm cannot, because the EE ratio of the deep Q-learning algorithm and Q-learning algorithm gradually decreases as the game progresses, and these two algorithms can neither perceive changes in the opponent's strategy nor adjust the EE ratio automatically as the opponent's strategy changes, i.e., as the environment varies. Meanwhile, regardless of how the opponent changes strategy, the score of our agent remains higher than the opponent's, illustrating that our method is more effective than the classic deep Q-learning algorithm in a non-stationary environment.

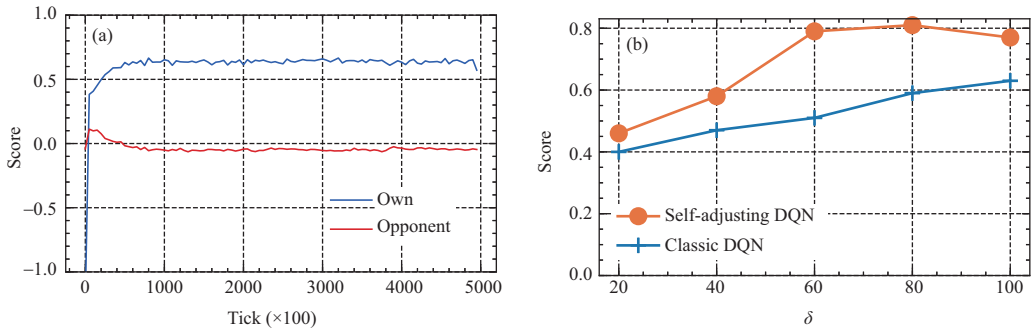

Figure 9 (Color online) Average rewards against a stationary periodic opponent. (a) $\delta = 40$; (b) $\delta = 20, 40, 60, 80, 100$.

• Owing to the EE problem, the convergence speed of the classic deep Q-learning algorithm is limited by the settings of ϵ and its decay parameter, while our method can instantly adjust the EE ratio in accordance with the learning progress of the opponent's strategy, guaranteeing fast convergence when the opponent's strategy changes slightly.

• According to Algorithm 2, the time point at which the predictive variance changes most remarkably corresponds to the moment when the opponent changes strategy, illustrating that model uncertainty provides us a visual understanding of the training process in such a manner that we can perceive the change in the opponent's strategy by observing the predictive variance.

3.4 Our agent vs. a stationary strategy opponent

In this part, we examine the improvements in the performance of our method compared with the classic deep Q-learning algorithm against a stationary strategy opponent, by which we mean that the opponent's strategy parameters remain at a fixed value throughout the game. Here, the periodic and exponential strategies are investigated.

3.4.1 *Our agent vs. a stationary periodic opponent*

The opponent adopts the periodic strategy with parameter $\delta \in (20, 100)$, and the parameter settings of the two algorithms are the same as in the case of a non-stationary periodic opponent. The scores against a stationary periodic opponent in FlipIt are presented in Figure 9, where Figure 9(a) shows the evolution of scores with $\delta = 40$, and Figure 9(b) shows the average rewards with $\delta = 20, 40, 60, 80, 100$.

As shown in Figure 9(a) where $\delta = 40$, the score of our method soon catches up with the opponent and maintains a steady advantage in the remaining time steps, by which we mean that after collecting interaction data with the opponent in the FlipIt environment and training the Q-network in the first few time steps, the prediction of the Q-value becomes accurate and sufficiently stable to guide the action selection, obtaining a stable optimal strategy in the following time steps. Figure 9(b) presents the corresponding scores of our method and the classic deep Q-learning algorithm, when $\delta = 20, 40, 60, 80, 100$. For different parameters δ, the performance of our method always exceeds that of the classic deep Q-learning algorithm, indicating that our method can enhance the learning effect of the classic deep Q-learning algorithm. From Figure 9(b), it can be seen that the average rewards of our method decrease when the parameter increases from 80 to 100, because when δ increases from 80 to 100, i.e., when the opponent's action frequency increases, we can obtain less training data relating to the opponent, thereby resulting in a decrease in the average rewards.

3.4.2 *Our agent vs. a stationary exponential opponent*

The opponent adopts the exponential strategy with parameter $\varphi \in (0.02, 0.1)$, and the parameter settings of the two algorithms are the same as for a non-stationary exponential opponent. The scores against a stationary exponential opponent in FlipIt are presented in Figure 10, where Figure 10(a) shows the evolution of scores with $\varphi = 0.04$, and Figure 10(b) shows the average rewards with $\varphi = 0.02, 0.04, 0.06, 0.08, 0.1$.

As shown in Figure 10(a) where $\varphi = 0.04$, our method earns higher scores than the classic deep Q-learning algorithm. Meanwhile, for different parameter values $\varphi = 0.02, 0.04, 0.06, 0.08, 0.1$, our method always has a better learning effect than the classic deep Q-learning algorithm.

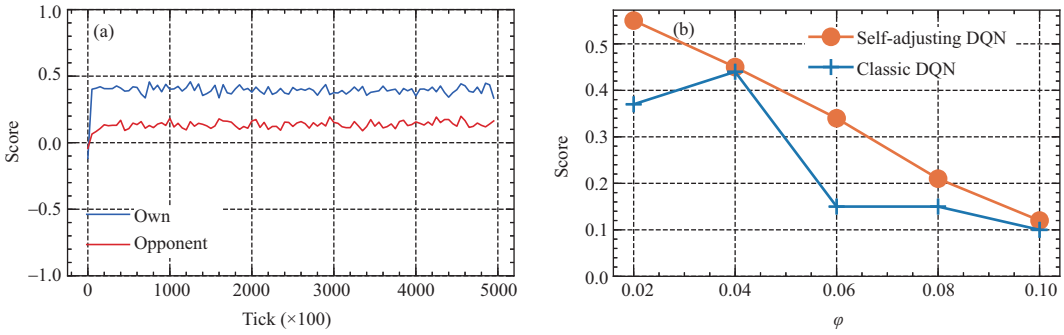

Figure 10 (Color online) Average rewards against a stationary exponential opponent. (a) $\varphi = 0.04$; (b) $\varphi = 0.02, 0.04, 0.06, 0.08, 0.1$.

Table 3 Against a stationary opponent with different algorithms

Stationary opponent		Score of our agent		Average improvement of adaptive DRL (%)
		Deep Q-learning	Adaptive DRL	
Periodic strategy	$\delta = 20$	0.4	0.46	30.56
	$\delta = 40$	0.47	0.58	
	$\delta = 60$	0.51	0.79	
	$\delta = 80$	0.59	0.81	
	$\delta = 100$	0.63	0.77	
Exponential strategy	$\varphi = 0.02$	0.37	0.55	47.52
	$\varphi = 0.04$	0.44	0.45	
	$\varphi = 0.06$	0.15	0.34	
	$\varphi = 0.08$	0.15	0.21	
	$\varphi = 0.1$	0.1	0.12	

Similarly, we examine classic deep Q-learning and the proposed adaptive DRL method for a stationary opponent with different parameters, and we list the scores of our agent in Table 3. It can be seen that compared with the classic deep Q-learning algorithm, the proposed adaptive DRL method improves the performance by at least 30.56% against the stationary strategy opponent.

It can be seen from Table 3 that our method has advantages over the classic deep Q-learning algorithm in a stationary environment. Together with the experiment results in a non-stationary environment obtained above, this leads to the conclusion that our method can achieve a better balance of EE than the classic deep Q-learning algorithm and has more satisfying performance in both non-stationary and stationary environments.

4 Conclusion

An adaptive DRL method was proposed in this paper for general MDP problems in non-stationary environments. This method consists of two parts: the self-adjusting deep Q-learning algorithm and a criterion of inappropriate network settings. With this method, the MDP problem in a non-stationary environment can achieve an automatic balance of exploration and exploitation as well as saving substantial costs in the parameter selection stage. Verified in experiments, this adaptive DRL method guarantees that the strategy keeps up with environmental changes and improves the performance by at least 14.28% and 30.56% in non-stationary and stationary environments compared with the classic deep Q-learning algorithms.

Acknowledgements This work was supported by National Key Research and Development Project (Grant No. 2018AAA0100802).

References

1 Puterman M L. Markov Decision Processes: Discrete Stochastic Dynamic Programming. Hoboken: John Wiley & Sons, 2014
2 Jia Q S, Wu J J. On distributed event-based optimization for shared economy in cyber-physical energy systems. Sci China Inf Sci, 2018, 61: 110203
3 Zhao Q C, Geirhofer S, Tong L, et al. Opportunistic spectrum access via periodic channel sensing. IEEE Trans Signal Process, 2008, 56: 785–796
4 Zhang R R, Guo L. Controllability of stochastic game-based control systems. SIAM J Control Optim, 2019, 57: 3799–3826

5 Tian F K, Yang C C. Deep belief network-hidden Markov model based nonlinear equalizer for VCSEL based optical interconnect. Sci China Inf Sci, 2020, 63: 160406
6 Dit-Yan S C, Choi S P, Yeung D Y, et al. Hidden-mode Markov decision processes. In: Proceedings of IJCAI Workshop on Neural, Symbolic, and Reinforcement Methods for Sequence Learning, 1999
7 Chen C L, Dong D Y, Li H X, et al. Hybrid MDP based integrated hierarchical Q-learning. Sci China Inf Sci, 2011, 54: 2279–2294
8 Ayoub A, Jia Z Y, Szepesvari C, et al. Model-based reinforcement learning with value-targeted regression. In: Proceedings of International Conference on Machine Learning, 2020. 463–474
9 Chebotar Y, Hausman K, Zhang M, et al. Combining model-based and model-free updates for trajectory-centric reinforcement learning. In: Proceedings of International Conference on Machine Learning, Sydney, 2017. 703–711
10 Sutton R S, Barto A G. Reinforcement Learning: An Introduction. Cambridge: MIT Press, 2018
11 Igl M, Zintgraf L, Le T A, et al. Deep variational reinforcement learning for POMDPs. In: Proceedings of International Conference on Machine Learning, Stockholm, 2018. 2117–2126
12 Perkins T J. Reinforcement learning for POMDPs based on action values and stochastic optimization. In: Proceedings of the AAAI Conference on Artificial Intelligence, Edmonton, 2002. 199–204
13 Zhang C J, Lesser V. Coordinated multi-agent reinforcement learning in networked distributed POMDPs. In: Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, 2011. 764–770
14 Bui V H, Hussain A, Kim H M. Double deep Q-learning-based distributed operation of battery energy storage system considering uncertainties. IEEE Trans Smart Grid, 2020, 11: 457–469
15 Morozs N, Clarke T, Grace D. Distributed heuristically accelerated Q-learning for robust cognitive spectrum management in LTE cellular systems. IEEE Trans Mobile Comput, 2016, 15: 817–825
16 Gimelfarb M, Sanner S, Lee C G. ϵ-BMC: a Bayesian ensemble approach to epsilon-greedy exploration in model-free reinforcement learning. In: Proceedings of the 35th Uncertainty in Artificial Intelligence Conference, Tel Aviv, 2019. 476–485
17 Hasselt H V, Guez A, Silver D. Deep reinforcement learning with double Q-learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, 2016. 2094–2100
18 Xiao L, Sheng G Y, Liu S C, et al. Deep reinforcement learning-enabled secure visible light communication against eavesdropping. IEEE Trans Commun, 2019, 67: 6994–7005
19 Hester T, Vecerik M, Pietquin O, et al. Deep Q-learning from demonstrations. In: Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, 2018. 3223–3230
20 Chen C L, Dong D Y, Li H X, et al. Fidelity-based probabilistic Q-learning for control of quantum systems. IEEE Trans Neural Netw Learn Syst, 2014, 25: 920–933
21 Khadka S, Majumdar S, Nassar T, et al. Collaborative evolutionary reinforcement learning. In: Proceedings of International Conference on Machine Learning, Long Beach, 2019. 3341–3350
22 Wiering M, Otterlo M V. Reinforcement Learning. In: Adaptation, Learning, and Optimization. Berlin: Springer, 2012
23 Dong D Y, Chen C L, Li H X, et al. Quantum reinforcement learning. IEEE Trans Syst Man Cybern B, 2008, 38: 1207–1220
24 Pathak D, Agrawal P, Efros A A, et al. Curiosity-driven exploration by self-supervised prediction. In: Proceedings of International Conference on Machine Learning, Sydney, 2017. 2778–2787
25 Liu Y, Liu Q, Zhao H K, et al. Adaptive quantitative trading: an imitative deep reinforcement learning approach. In: Proceedings of the AAAI Conference on Artificial Intelligence, New York, 2020. 2128–2135
26 Liu Z, Li X, Kang B, et al. Regularization matters in policy optimization — an empirical study on continuous control. In: Proceedings of International Conference on Learning Representations, 2021. 1–44
27 Sung T T, Kim D, Park S J, et al. Dropout acts as auxiliary exploration. Int J Applied Eng Res, 2018, 13: 7977–7982
28 Xie S R, Huang J N, Liu C X, et al. NADPEx: an on-policy temporally consistent exploration method for deep reinforcement learning. In: Proceedings of International Conference on Learning Representations, New Orleans, 2019. 1–18
29 Wang X S, Gu Y, Cheng Y H, et al. Approximate policy-based accelerated deep reinforcement learning. IEEE Trans Neural Netw Learn Syst, 2020, 31: 1820–1830
30 Garcelon E, Ghavamzadeh M, Lazaric A, et al. Conservative exploration in reinforcement learning. In: Proceedings of International Conference on Artificial Intelligence and Statistics, 2020. 1431–1441
31 Xie S, Girshick R, Dollár P, et al. Aggregated residual transformations for deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Hawaii, 2017. 1492–1500
32 Gal Y, Ghahramani Z. Dropout as a Bayesian approximation: representing model uncertainty in deep learning. In: Proceedings of International Conference on Machine Learning, New York, 2016. 1050–1059
33 Arulkumaran K, Deisenroth M P, Brundage M, et al. Deep reinforcement learning: a brief survey. IEEE Signal Process Mag, 2017, 34: 26–38
34 Kendall A, Gal Y. What uncertainties do we need in Bayesian deep learning for computer vision? In: Proceedings of International Conference on Neural Information Processing Systems, Long Beach, 2017. 5580–5590
35 Gal Y, Islam R, Ghahramani Z. Deep Bayesian active learning with image data. In: Proceedings of International Conference on Machine Learning, Sydney, 2017. 1183–1192
36 Rutkowska A. Properties of the cox-stuart test for trend in application to hydrological series: the simulation study. Commun Stat-Simul Comput, 2015, 44: 565–579
37 Canonaco G, Restelli M, Roveri M. Model-free non-stationarity detection and adaptation in reinforcement learning. In: Proceedings of the 24th European Conference on Artificial Intelligence, Santiago de Compostela, 2020. 1047–1054
38 Lei Y G, Li N P, Lin J. A new method based on stochastic process models for machine remaining useful life prediction. IEEE Trans Instrum Meas, 2016, 65: 2671–2684
39 Bernardo J, Smith A F. Bayesian Theory. Hoboken: John Wiley & Sons, 2009
40 Alliney S, Ruzinsky S A. An algorithm for the minimization of mixed l_1 and l_2 norms with application to Bayesian estimation. IEEE Trans Signal Process, 1994, 42: 618–627
41 Sadowsky J S, Bucklew J A. On large deviations theory and asymptotically efficient Monte Carlo estimation. IEEE Trans Inform Theor, 1990, 36: 579–588
42 Touchette H. The large deviation approach to statistical mechanics. Phys Rep, 2009, 478: 1–69
43 Hu P F, Li H X, Fu H, et al. Dynamic defense strategy against advanced persistent threat with insiders. In: Proceedings of IEEE Conference on Computer Communications, Hong Kong, 2015. 747–755
44 Laszka A, Horvath G, Felegyhazi M, et al. FlipThem: modeling targeted attacks with FlipIt for multiple resources. In: Proceedings of International Conference on Decision and Game Theory for Security, Los Angeles, 2014. 175–194
45 Mu Y, Guo L. How cooperation arises from rational players? In: Proceedings of IEEE Conference on Decision and Control, Atlanta, 2010. 6149–6154
46 Greige L, Chin P. Reinforcement learning in FlipIt. 2020. ArXiv:2002.12909