A Survey on Deep Recurrent Q Networks

M. V. K. Gayatri Shivani(✉), S. P. V. Subba Rao, and C. N. Sujatha

Sreenidhi Institute of Science and Technology, Yamnapet, Hyderabad, India
mvkgayatrishivani@gmail.com, {spvsubbarao, cnsujatha}@sreenidhi.edu.in

Abstract. Reinforcement learning (RL), one of the branches of machine learning, enables a system to learn through trial and error. RL helps in solving control and decision-making tasks. Applying Deep Learning to Reinforcement learning has made it much better at solving many problems. Deep Reinforcement learning, a combination of deep learning and Reinforcement Learning is gaining a lot of interest and application in solving real-world problems. Among Deep Reinforcement learning, Deep Q networks emerged as a popular algorithm. While Deep Q networks have been successfully applied to a lot of scenarios, their application is based on the notion that the agent can completely perceive the environment. In real-time applications, this notion has a fallacy as complete observability is a difficult and sometimes impossible endeavor in real-time and the real world, therefore the use of recurrent networks along with Deep Q networks has been suggested for application to partially observable environments. This paper provides a literature review on various deep recurrent Q network applications. The paper first provides a brief introduction to the concept behind Deep Recurrent Q networks, and the various modifications to improve its performance and then proceeds to review its various applications in different fields.

Keywords: Deep Q networks · Deep Recurrent Q network · POMDPs · Reinforcement Learning

1 Introduction

With the ever-growing demand for various kinds of technological services, the need for intelligent systems is at an all-time rise. An important objective of Artificial Intelligence is to design systems that could interact with the environment, and learn and take decisions that could maximize the outputs or profits of the task assigned. Reinforcement Learning (RL) provides a method to achieve the aforementioned goal. Therefore, Reinforcement learning (RL) [1] is a significant research domain in Artificial intelligence. Reinforcement learning is an artificial intelligence-based learning and training method wherein an agent is trained to take its own decisions, the results thus obtained are observed, and then actions are selected or adjusted based on observations to obtain an optimum policy that maximizes long-term results. However, the previous applications based solely on RL were limited to low dimension problems and lacked scalability as even though

S. Nandan Mohanty et al. (Eds.): ICISML 2022, LNICST 470, pp. 251–261, 2023.
https://doi.org/10.1007/978-3-031-35078-8_21

the method was shown to converge, a lot of time was taken to find an optimal policy, thus making it unsuitable for large-scale systems that need to process large amounts of data. But the recent developments in Deep Learning (DL) [2] have made a significant impact on various areas of machine learning including RL. Deep neural networks (DNN) are very useful because they are designed to efficiently and on their own find compact low-dimensional representations (features) of high-dimensional data, which is one of their most important properties. The above-mentioned property has been applied to reinforcement learning, giving birth to Deep Reinforcement Learning (DRL). DRL uses the advantages of Deep neural networks (DNNs) in the training process leading to an increase in the speed and performance of the algorithm.

DRL has been increasingly used to solve problems that can be modelled as a Markov decision process (MDP) [3]. MDP is a mathematical analysis tool that helps in control and decision tasks. There are several different methods under DRL but one of the most popular methods is Q learning. When Deep Neural Networks are applied to Q learning, it gives rise to a structure called Deep Q networks (DQN). DQNs have been effectively applied to many real-world applications in research and have shown promising results. A lot of variations of DQN have also been developed like Double DQN, Duelling DQN, and DQN with experience play, to mitigate the issues found in the original DQN also called a vanilla DQN. But one of the major drawbacks of DQN is that it is assumed that the complete environment is always perceivable. This assumption fails in a real-world environment, where there are multiple agents and the observation of the environment is also partial. This converts the system model from an MDP to POMDP- Partially Observable Markov Decision Process. It has been observed that for a POMDP model a DQN network cannot successfully find an optimal policy. In recent literature, several Deep RL models have been developed to solve issues arising out of multi-agent applications and partial observability is one such issue, which has also been studied and several alternate models are being researched to solve the problem of partial observability.

1.1 Related Work

A no. of surveys has been conducted on DRL and its applications. For example, [5] and [6] are general surveys on DRL and the various methods or programs under it. While [7] focuses completely on the issues that arise in a multi-agent Deep RL system. One of the issues is that of partial observability and one of the methods use to mitigate it is Deep recurrent Q networks. To the best of our knowledge, we haven't found any literature that solely focuses on a survey on Deep Recurrent Q networks that have been used to mitigate the partial observability issue.

2 Background

2.1 Reinforcement Learning

In Reinforcement learning, the learning takes place through trial and error. The learner decides to act, the action taken alters the state of the environment and the learner receives feedback. It can be negative feedback or a positive feedback. Positive feedback can also

be called a reward (short term) that increases the value of the action taken, while negative feedback or no reward gained, diminishes the value of the action. The learner stores the actions that give rewards and try to find a policy that could help it determine rewardable actions and then uses this information in the future to find a similar policy to maximize its results on a similar problem that it encounters. The action to be taken at a given time t is represented as at, while the state of the environment before the action is taken is represented as st, and the next state to which transits after an action is taken is represented as st + 1 the feedback provided is represented as rt. Each interaction of the agent with the environment can be represented in a succession of steps of a feedback loop of actions, states, and rewards. This series can tend to infinity but it can also be restricted to be finite by limiting n by providing a terminal state. The series of actions from an initial state to a terminal state is termed an episode. The policy which maps an action to a state can be deterministic or stochastic. The policy can be defined as follows:-

$$\pi = \Psi(s) = \left\{ p(a_i|s) | \forall a_i \in \Delta_\pi \Sigma_i p(a_i|s) = 1 \right. \tag{1}$$

Here p(ai|s) is the probability of picking an action for a given state s, $\Delta\pi$ is the representation of all the actions available for a policy. Initially, a random policy $\pi 0$ is assigned to the agent, the agent then starts to explore by trial and error which policy is better until it can no longer find a policy that is better than the optimal policy π^*. To determine which policy is better among all the different available policies, we compare the policies based on a function called the value function. A value function determines which state or a state-action pair is better. The Bellman Equation is used to determine the output of the value function. The value function of a state-action pair or a state is stored in a table during calculations This renders them ineffective for handling large amounts of data.

2.2 Deep Q Learning

Q learning is one of the methods of reinforcement learning that is used for finding out the optimal policy. But in practical scenarios when the system model is complicated and the data to process is large, the Q-learning algorithm fails to or struggles to determine an optimal policy. To overcome this drawback, Deep Q learning is presented. In Deep Q learning, instead of a table, a deep neural network is used to obtain the approximate value of Q(s, a). But, as stated in [8], if a non-linear approximator is used with reinforcement learning algorithms, the average reward obtained may not be stable, or even worse, would not converge, as even a small change in the value of Q(s, a) would greatly affect the policy. To mitigate this issue, two methods are used namely Experience Replay and Target Q-network [4] (Fig. 1).

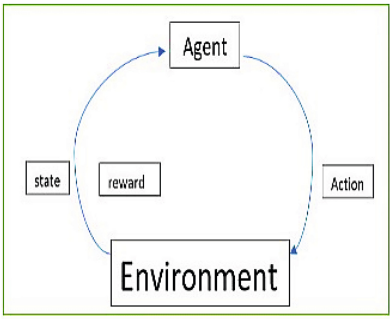

Fig. 1. Model representation of Reinforcement Learning

2.3 POMDPs

Deep Q learning algorithms like DQN, Double DQN, etc., are modeled as an MDP, when a system is modeled as an MDP we assume that the system follows the Markov property and that the system can obtain complete state information, but this assumption fails in practical scenarios because accessing complete state information is difficult in real-world application because of the limited accuracy of the data collecting systems, other agents and unwanted noise. This leads to only the collection of partial information. To mitigate this issue, the system can be designed as a POMDP-Partially Observable Markov Decision Process. A POMDP is a generalized case of an MDP. It takes into account the partial observability factor. A POMDP is represented using a 6-tuple (S; A; P; R; O; Z) where the first four elements are similar to MDP, O is the observation space, and Z: $S \times A \times O \rightarrow [0,1]$ represents the distribution of probability of the observations given a state $s \in S$ and an action $a \in A$.

3 Deep Recurrent Q Networks

To make the DQN networks model deal with partial observability, it is suggested in [9, 10] to add recurrency to the existing vanilla DQN structure. The authors modified the architecture of DQN marginally by substituting the first fully connected layer with a recurrent LSTM layer of similar size. In DRQN, the function $Q(o, a)$ is approximated by using a recurrent neural network, Unlike DQN in a partially perceivable environment of DRQN the state s_t is hidden and only receives an observation o_t that is correlated with s_t. Here o is the observation and a is the action. DRQN treats the network's hidden state h_{t-1} as an internal state, therefore DRQN is represented by $Q(o_t, h_{t-1}, a_t, \theta_i)$ where i is the network parameter at the i^{th} training step. For finding the optimum policy, that is robust to partial observability the model proposed recurrently integrates an arbitrarily long sequence of histories of the observations. When the authors [9] trained the network and used it along with DQN for Atari games. An LSTM layer is usually added instead of the first fully connected layer to incorporate the previous states. This inclusion of past states helps in understanding the environment and making a decision. It is observed that it performs on par with DQN with complete observations, and performs better than DQN when trained with partial observations but evaluated using full observation, showcasing

that it can generalize its policies for the full observation case. While parallelly authors in [10] used the same idea in a different format of games and showcased that recurrency helps in text-based fantasy games. But this method suffers from a drawback that it only considers the history of observations without explicitly considering the actions, which leads to a degradation in performance [11]. Different techniques have been developed since then that have utilized recurrent networks with different modifications to address Partial observability as well as other drawbacks.

3.1 Deep Distributed Recurrent Q Network (DDRQN)

The authors in [12] propose a method to extend the ability of DRQN to perform in a multi-agent scenario. DRQN is used to address partial observability with only a single RL agent, while DDRQN is designed to address partial observability in a multi-agent scenario, where more than one RL agent has to perform and they all need to communicate with each other. The method introduces three key modifications over the previous method that was used to extend DRQN to multi-agent scenarios: i) last-action input: in this, for every agent, its last action acts as input to its next time step. ii) inter-agent weight sharing: in this, every agent's weight is tied with every other agent, weight sharing reduces the number of parameters that an agent needs to learn, which leads to improvement in learning speeds iii) disabling experience replay: a common feature that is used with every DQN network, disabling experience replay ensures that the learning is not obsolete and misleading in a multi-agent scenario. These modifications change the Q function in DDRQN to $Q(o_t^m, h_{t-1}^m, m, a_{t-1}^m, a_t^m, \theta_i)$

3.2 Action-Based Deep Recurrent Q Network (ADRQN)

ADRQN is a model-free RL learning method that is used to work on POMDP problems [11]. The authors use actions as well as observations paired as inputs. They designed this model to incorporate the actions taken also into history as an action performed is important for belief estimation and this idea is not incorporated in the DRQN model, DRQN only considers the convolved features of the observations and actions. They combine the action performed and the observation obtained as input for the Q network. The authors reason that for a finite observation space and with a fixed policy the information of actions taken can be extrapolated and so providing the action information could be redundant and that's specifically the reason why the DRQN algorithm doesn't consider it but the observation space does get truncated when employing recurrent models and in an infinite observation space for complex real-world problems, it becomes even more difficult to extrapolate the action information from the truncated observation history. Thus explicit information about the action information would help a lot during the training process. For their proposed model the authors make use of a fully connected layer to encode the actions and then couple them with an observation associated with them. The pairs formed by the action observation are fed as an input to the LSTM layer in the form of a time series. The LSTM layer learns about the latent states. The learning is used by the fully connected layer to calculate Q values. The model has been found to perform better than DRQN when tested on certain games.

3.3 Deep Attention Recurrent Q Network (DARQN)

In [13] the authors incorporate attention mechanisms into Deep Q networks. The attention mechanism provides insights into the logic used by the agent when the agent is making decisions. The attention mechanism can also be considered as an additional filter gate incorporated into LSTM that can be used to process the structured visual data that CNN produces for an entire image. The architecture consists of three networks:- Convolutional, attention, and recurrent. The convolutional network is used to produce feature maps at every time step t, the feature maps are transformed into a set of vectors by the attention network, which produces the linear combination of the set of vectors into a context vector that is given as an input to the recurrent network. The attention network uses two methods to produce the context vector, one is titled soft and the other is titled hard. The authors tested the performance of the model on certain Atari games along with DQN and DRQN. While the model performed better than DQN and DRQN on certain games. It performed worst on others. The authors express that using attention mechanism is advantageous as it help us observe and understand the actions of agents better (Fig. 2).

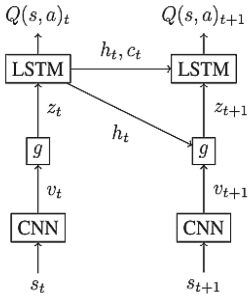

Fig. 2. Representation of DARQN proposed by authors in [13].

3.4 Deep Recurrent Double Q Learning

The authors combine the concept of Double deep Q learning with Deep Recurrent Q learning by introducing an LSTM layer in a Double Deep Q learning network [14]. They implemented the CNN network proposed by Chen [17] by introducing some changes in the last layers and utilizing Adam error, initially they used Q learning algorithm with a simple CNN with 3 Conv 2D layers but the process of learning was slow so they modified the design to include Dense 512 and 256 networks with Relu and linear activation in the last layer and added a LSTM layer with a tanh activation function. The authors tested the model on a certain set of Atari games and found that the algorithm is better for a certain specific set of games that share certain similarities (Fig. 3).

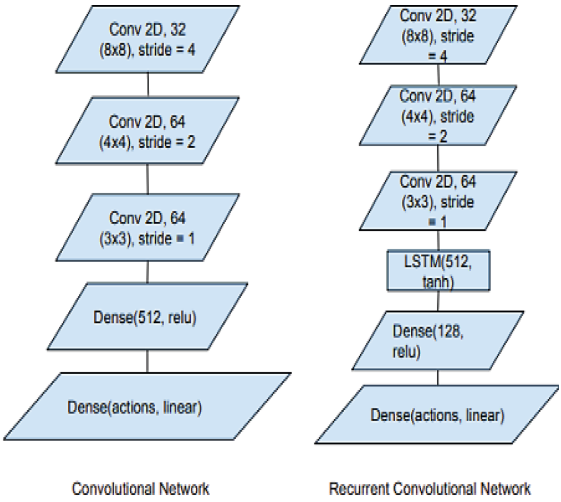

Fig. 3. Convolutional network proposed by authors for their model in paper [14].

3.5 Deep Recurrent Policy Inference Q-network (DRPIQN)

In order to make DQN networks more robust in multi-agent systems, the authors in [15] suggest modifications to the existing DQN system such that the agent learns about other agents' policies through auxiliary tasks that it is made to carry out. This gave birth to the Deep Policy Interference Q Network (DPIQN). The authors explain a scenario where there are two agents a control agent and a target agent in a multi-agent system. The DPIQN enables the control agent to learn other agents' policy features by using auxiliary tasks and this enhances the hidden representations which enable the controllable agent to exploit other agents' actions in a multiple-agent scenario. The policy features can be defined as a hidden representation that can be used by the controllable agent to deduce the target agent's policy and this is represented as a vector. The spatial-temporal behavior of the target agent's policy is encoded by this approximate representation and it can be obtained by observing the target agent's behaviour for a series of timesteps. By incorporating this policy vector, the controllable agent has an opportunity to learn a better q-function than when compared to traditional methodologies. The DPIQN can adapt to dynamic environments in which the target agents' policy can change over time. The authors further make the system more adaptable by introducing the LSTM layer and converting the network into a Deep Recurrent Policy Interference Network to deal with partial observability. The authors found that DRPIQN and DPIQN both perform better than DRQN and DQN in soccer field scenarios in various settings.

3.6 Duelling Double-Deep Recurrent Q-learning

A Duelling Double Deep Recurrent Q-learning network combines the advantages of a Duelling Q network, Double Deep Q network, and a DRQN. The authors proposed to design a model for the autonomous quadrotor for obstacle avoidance [16]. The idea of

Double Q learning was described by Hado et al. [18] to reduce the overestimation issue in normal DQN networks.

$$\text{DQN Model}: \quad Y_t = R_{t+1} + \gamma \max Q(S_{t+1};\ a_t;\ \theta) \tag{2}$$

$$\text{DDQN Model}: \quad Y_t = R_{t+1} + \gamma Q(S_{t+1};\ \arg\max\ Q(S_{t+1};\ a_t;\ \theta_t);\ \theta_t^1) \tag{3}$$

Here at represents action, R is the reward, Y is the updated target value, γ is the discount factor, S represents state, θ is the parameters of the network.

A duelling DQN is a representation of non-sequential Deep learning architecture. The model is split into two different subnetworks each of which has its own output layer and fully connected layer after the initial convolutional layers. One of the subnetworks estimates the value function and its output network consists of a single node The other subnetwork known as an "Advantage" network computes the advantage of taking a particular action [19]. The authors found that their suggested model learns more efficiency and performs better during testing than DQN, DRQN and Double DRQN (Table 1 and Fig. 4).

Table 1. Comparison of the Algorithms with Standard DRQN

Algorithm	Improvement Over the original DRQN
DDRQN	Can be used in a multi-agent scenario
ADRQN	Includes past actions as well as observation for better decision making
DARQN	Can observe and understand how the agent is taking action in real-time
DRPIQN	Designed to perform in Multi-agent scenarios as opposed to a single agent

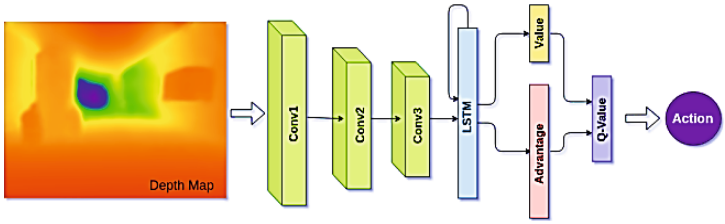

Fig. 4. Model of Duelling DRQN as proposed in the paper [16].

4 Application of Deep Recurrent Q Networks

Deep Recurrent Q networks find its application to numerous different fields. Several real-life applications in the field of robotics, Autonomous Vehicles, Signal Processing, Image Processing, Wireless Communication, etc. have used the concept of DRQN to

design intelligent systems that can take decisions on their own and optimize the end result. Only some of the algorithms mentioned above have been used for real-time applications, wherever notable applications have been found, they have been mentioned in the paper (Table 2).

Table 2. List of Different applications of Deep Recurrent Reinforcement Learning

S.no	Reinforcement Learning Method Used	Application
1.	DDRQN	Uses DDRQN to improve radar performance by using it to help the radar change its bandwidth and central frequency (in LFM waveforms) to improve target detection performance by mitigating interference [20]
2.	DRQN	DRQN is used to deal with the issue of Dynamic Spectrum Access in Cognitive radio networks [21]
3.	DRQN	In capacity-constrained fog nodes, a Uni-node is incapable of performing computing-intensive applications. Therefore, offloading of tasks is required. The paper studies the use of DRQN for joint task loading and resource sharing in Multi fog networks. The problem is modeled as POMDP and solved using DRQN [22]
4.	DDRQN	For managing bandwidth in storage servers, a dynamic storage allocation system is designed using DDRQN The agent decides the allowable bandwidth to each filer thus improving throughput and reducing load [23]
5.	DRQN	DRQN is used in multi-agent mode to reduce Age of Information (AoI) in wireless ADHOC networks [24]
6.	DRQN	Uses DRQN for Defense decision making for cyber security. The method enable network defence strategies to learn an optimal defence strategy [25]
7.	DRQN with bidirectional RNN	In cognitive radio network framework for DSA (Dynamic Spectrum Access) [26]
8.	DRQN	Intelligent Task scheduling and Partial offloading are developed for SDN-based fog networks using DRQN [27]

5 Conclusion

The paper surveys every available DRQN algorithm and its variations to the best of the authors' ability, it reviews the methods and all its applications, DRL is a very promising technology with application to a wide variety of fields. Partial observability is one of the hurdles that need to be overcome to make DRL methods more robust for practical scenarios. Especially in the field of wireless communication where the environment is extremely chaotic and unpredictable, and modeling the environment as completely observable may lead to erroneous results or less than satisfactory results when testing in real-time. In such a situation working by using algorithms that address partial observability would lead to more promising results. In the future, one of the algorithms will be applied for the selection of modulation and coding schemes and compared against DQN.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT Press, Cambridge (1998)
2. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep Learning. MIT Press, Cambridge (2016)
3. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons (2010)
4. Li, Y.: Deep reinforcement learning: an overview. arXiv preprint arXiv:1701.07274 (2017)
5. Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: A brief survey of deep reinforcement learning. IEEE Signal Process. Mag. **34**(6), 26–38 (2017)
6. Lee, Y.L., Qin, D.: A survey on applications of deep reinforcement learning in resource management for 5G heterogeneous networks. In: 2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), pp. 1856–1862 (2019). https://doi.org/10.1109/APSIPAASC47483.2019.9023331
7. Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications
8. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
9. Hausknecht, M., Stone, P.: Deep recurrent q-learning for partially observable mdps. In: 2015 AAAI Fall Symposium Series (2015)
10. Narasimhan, K., Kulkarni, T., Barzilay, R.: Language understanding for text-based games using deep reinforcement learning. CoRR abs/1506.08941 (2015)
11. Zhu, P., Li, X., Poupart, P., Miao, G.: On improving deep reinforcement learning for pomdps. arXiv preprint arXiv:1704.07978 (2017)
12. Foerster, J.N., Assael, Y.M., de Freitas, N., Whiteson, S.: Learning to communicate to solve riddles with deep distributed recurrent q-networks. arXiv preprint arXiv:1602.02672 (2016)
13. Sorokin, I., Seleznev, A., Pavlov, M., Fedorov, A., Ignateva, A.: Deep attention recurrent Q-network. arXiv preprint arXiv:1512.01693 (2015)
14. Moreno-Vera, F.: Performing deep recurrent double Q-learning for Atari games. In: 2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI), pp. 1–4 (2019). https://doi.org/10.1109/LA-CCI47412.2019.9036763
15. Hong, Z.-W., Su, S.-Y., Shann, T.-Y., Chang, Y.-H., Lee, C.-Y.: A deep policy inference q-network for multi-agent systems. arXiv preprint arXiv:1712.07893 (2017)

16. Jiajun, Ou., Guo, X., Zhu, M., Lou, W.: Autonomous quadrotor obstacle avoidance based on dueling double deep recurrent Q-learning with monocular vision. Neurocomputing **441**, 300–310 (2021). https://doi.org/10.1016/j.neucom.2021.02.017
17. Chen, C., Ying, V., Laird, D.: Deep Q-Learning with Recurrent Neural Networks
18. Volodymyr, M., et al.: Playing Atari with deep reinforcement learning. In: NIPS Deep Learning Workshop (2013)
19. Sewak, M.: Deep Q Network (DQN), Double DQN, and Dueling DQN: a step towards general artificial intelligence. In: Sewak, M. (ed.) Deep Reinforcement Learning: Frontiers of Artificial Intelligence, pp. 95–108. Springer Singapore, Singapore (2019). https://doi.org/10.1007/978-981-13-8285-7_8
20. Thornton, C.E., Kozy, M.A., Buehrer, R.M., Martone, A.F., Sherbondy, K.D.: Deep reinforcement learning control for radar detection and tracking in congested spectral environments. IEEE Trans. Cognitive Commun. Netw. **6**(4), 1335–1349 (2020). https://doi.org/10.1109/TCCN.2020.3019605
21. Xu, Y., Yu, J., Buehrer, R.M.: The application of deep reinforcement learning to distributed spectrum access in dynamic heterogeneous environments with partial observations. IEEE Trans. Wireless Commun. **19**(7), 4494–4506 (2020). https://doi.org/10.1109/TWC.2020.2984227
22. Baek, J., Kaddoum, G.: Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multifog networks. IEEE Internet of Things J. **8**(2), 1041–1056 (2021). https://doi.org/10.1109/JIOT.2020.3009540
23. Xue, S., Luo, B., Liu, D., Li, Y.: Event-triggered adaptive dynamic programming for continuous-time nonlinear two-player zero-sum game. In: Cheng, L., Leung, A.C.S., Ozawa, S. (eds.) ICONIP 2018. LNCS, vol. 11307, pp. 15–25. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-04239-4_2
24. Leng, S., Yener, A.: Age of information minimization for wireless ad hoc networks: a deep reinforcement learning approach. In: 2019 IEEE Global Communications Conference (GLOBECOM), pp. 1–6 (2019). https://doi.org/10.1109/GLOBECOM38437.2019.9013454
25. Liu, X., Zhang, H., Dong, S., Zhang, Y.: Network defense decision-making based on a stochastic game system and a deep recurrent Q-network. Comput. Secur. **111**, 102480 (2021)
26. Chen, P., Guo, S., Gao, Y.: Deep reinforcement learning with bidirectional recurrent neural networks for dynamic spectrum access. In: 2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall), pp. 1–5 (2021). https://doi.org/10.1109/VTC2021-Fall52928.2021.9625359
27. Baek, J., Kaddoum, G.: Online partial offloading and task scheduling in SDN-fog networks with deep recurrent reinforcement learning. IEEE Internet of Things J. **9**, 11578–11589 (2021). https://doi.org/10.1109/JIOT.2021.3130474