Deep Reinforcement Learning for Dynamic Spectrum Access in Wireless Networks

Y. Xu, J. Yu, W.C. Headley, and R.M. Buehrer
Wireless @ Virginia Tech
email:xuyue24@vt.edu, jianyuan@vt.edu, cheadley@vt.edu, buehrer@vt.edu*

Abstract—**This paper investigates the use of deep reinforcement learning (DRL) to solve the dynamic spectrum access problem. Specifically, we examine the scenario where multiple discrete channels are shared by different types of nodes which lack the ability to communicate (with other node types) and do not have *a priori* knowledge of the other nodes' behaviors. Each node's objective is to maximize its own long-term expected number of successful transmissions. The problem is formulated as a Markov Decision Process (MDP) with unknown system dynamics. In order to overcome the challenge of an unknown environment combined with a prohibitively large transition matrix, we apply two specific DRL approaches: The Deep Q Network (DQN) and The Double Deep Q Network (DDQN). Additionally, we also introduce techniques to improve DQNs including an eligibility trace, prior experience and the "guess process". We first study the proposed DQN approach in a simple environment. The simulations show that both DQN and DDQN can effectively learn different nodes' communication patterns and achieve near optimal performance without prior knowledge. Then, we examine these techniques in a more complex environment and conclude that although different implementation variations obtain different performance, our proposed DQN nodes can still learn to avoid collisions and achieve near optimal performance even in the more complex scenario.**

Index Terms—**dynamic spectrum access, Deep Reinforcement Learning, MDP, DQN, DDQN, Implementations**

I. Introduction

It is generally understood that the current static spectrum access mechanisms can not effectively support the increasing demand for spectrum from a growing number of wireless (both mobile and fixed) devices. Dynamic spectrum access (DSA) [1][2] is an emerging concept for improving spectrum utilization efficiency in wireless networks and meeting the need for more capacity. Generally, DSA assumes that secondary users (SU) search and use idle channels which are not being used by their primary users (PU).

This work considers an uncoordinated multichannel access problem with discrete channels. Each channel has two possible states (occupied and idle) meaning that the overall spectrum is modeled as a 2^N-state Markov chain. The channels are also assumed to have different channel qualities due to SNR. Each node in the environment selects one channel at each time step to transmit a packet. If the selected channel is idle (and not selected by other nodes), the transmission is successful; otherwise, there is a collision. Every node is able to detect the channel state and corresponding quality using spectrum sensing (and channel-probing). The nodes are assumed to be of various types based on their spectrum behavior which will be described in more detail shortly. The overall goal of the nodes is to optimize their individual performance.

Generally speaking, in order to exploit idle channels and improve spectrum utilization efficiency without knowing the various node types, reinforcement learning (RL), especially MDP, has emerged as a potential solution [3]. The advantage of RL is that exploration actions are selected in situations when the knowledge about the environment is uncertain resulting in learning through experience. During the learning process, the more information the node obtains (i.e., experience), the higher the probability that it will select the optimal action. However, when applying MDPs, if the number of channels is even modest, it will be difficult to compute and manipulate the transition matrix (and other associated matrices) due to the exponential relationship between the number of channels and the number of states.

As a key point to overcoming the prohibitive computational requirements of a high-dimensional transition matrix and learning an unknown environment, we investigate the use of Deep Reinforcement Learning [4], and in particular, Deep Q learning. By combining a deep neural network with Q-learning(also called a Deep Q Network), DQN can estimate the Q values by inputting states and rewards. DQN, Double DQN and other implementations are applied in order to learn the optimal channel access policy via online learning. These approaches are able to deal with large systems, and find a good suboptimal or even optimal policy directly from historical experience without any prior information about the system dynamics [5].

The remainder of this paper is organized as follows. Section II describes the related previous work. In Section III, the system model including the problem formulation and the communication environment assumed in this paper is presented. Section IV describes the details of the DQN, the Double-DQN and various implementations that can potentially improve performance. This is followed by the simulation and numerical results in Section V. Finally, we conclude this paper in Section VI.

II. Related Work

Dynamic spectrum access has been widely studied [1]. When channels are independent and identically distributed (i.i.d.), there are many algorithms, such as vertical handoff and power allocation[6][7], that have been shown to be near-optimal under certain conditions. However, these methods are based on prior knowledge of the communication systems' protocols, inevitably need to calculate transition matrices and are not adaptable in more complex communication environments.

In recent years, the development of machine learning has attracted people's attention in the area of wireless communications (and many other fields), as it can provide good solutions and is adaptable to the many challenges facing wireless systems [8]. Many works have begun to focus on the more practical and complex problem where the system dynamics are unknown. The most popular algorithm for accomplishing is Q-learning since it is a model-free method and can learn the policy directly via online learning [9]. Although these algorithms are widely used, they still require substantial resources in order to build the Q-value table (essentially, it is table look-up approach). In 2013, Google DeepMind used a deep neural network, called a DQN, to approximate Q-values in Q-learning in order to overcome the limitations of traditional Q-learning [4]. Based on this development, the authors of [10] proposed a new approach implementing DQN in dynamic spectrum access under correlated channels without prior knowledge. This work provided nice insights into the combination of DQN and wireless communications. However, it was claimed that the model was a partially observable MDP (POMDP), although it still used 'states' instead of observations. Technically, this is still a fully-observed channel. Similarly, our current approach also assumes a fully observed channel.

III. System Model

A. Problem Formulation

In this work, we consider a dynamic spectrum access environment where N nodes dynamically choose to transmit on one of K channels. At the beginning of each time step, some fraction of the nodes sense all channels including their quality (e.g., this could represent SNR or SIR) and chooses to either transmit on one of the K channels or not to transmit at all. If the node transmits and the corresponding channel is truly idle, the transmission succeeds and the user receives a positive reward. Because of channel noise (or possibly fading), each channel has a different quality. Thus, the rewards range from 0 to 100 for a successful transmission. If the channel state is occupied, the user transmission fails and there is a negative reward (-10). As mentioned, nodes can also choose to wait (i.e., not transmit) in order to receive a higher future reward. Because we do not want these nodes to wait too often, the reward for waiting is -1. The goal is to learn a policy that maximizes the expected long-term reward.

As shown in Figure 1, an intelligent agent interacts with an environment over a sequence of discrete times to accomplish a task. At time t, the agent observes the state of the environment, $s_t \in S$, where S is the set of possible states. It then takes an action, $a_t \in A_{s_t}$ where A_{s_t} is the set of possible actions at state s_t. As a result of the state-action

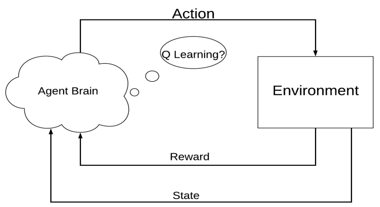


Fig. 1: The agent-environment interaction process

pair, (s_t, a_t), the agent receives a reward r_{t+1}, and the environment moves to a new state s_{t+1} at time $t+1$. The goal of the agent is to maximize some function of that reward. For example, the performance criterion to be maximized at time t could be $R_t \triangleq \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_{\tau+1}$, where $\gamma \in (0,1]$ is a discount factor for weighting future rewards. In general, the agent takes actions according to some decision policy π and, with sufficient experience, the agent can learn an optimal decision policy π^* which will maximize its performance criterion (typically long-term accumulated reward).

Q-learning [3] is one of the most popular RL methods. The iteration approach could be policy gradient based on the state transition probabilities, or a value-based one. In a value-based approach, a Q-learning RL agent learns an action-value function $Q^{\pi}(s,a)$ corresponding to the expected accumulated reward when an action a is taken in the environmental state s under the decision policy π:

$$Q^{\pi}(s,a) \triangleq \mathbb{E}[R_t | s_t = s, a_t = a, \pi]. \tag{1}$$

The optimal action-value function $Q^*(s,a) \triangleq \max_{\pi} Q^{\pi}(s,a)$ obeys the well-known Bellman equation [11]:

$$Q^*(s,a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q^*(s', a') | s_t = s, a_t = a] \tag{2}$$

For online learning, we must balance exploitation (i.e, using the best known action) and exploration (learning new, possibly better actions). The ε-greedy policy is often adopted to accomplish this tradeoff. For the ε-greedy policy, the agent selects the greedy action $a_t = \arg\max_a Q(s_t, a)$ with probability $1-\varepsilon$, and selects a random action with probability ε. The reason for randomly selecting an action some percentage of the time is to avoid getting stuck with a $Q(s,a)$ function that has not yet converged to $Q^*(s,a)$ (i.e., learning new, better actions).

However, using traditional Q-learning is inefficient (perhaps even impractical) for solving our online-learning problem. As mentioned above, the primary reason is that Q-learning needs a large amount of resources to compute and store Q-values resulting in an unacceptable amount of necessary computation and memory resources, especially when the number of channels is large. Thus, taking into account the advantages of deep neural networks and combining them with reinforcement learning results in Deep Reinforcement Learning. In DRL, deep neural networks are used to approximate the state-action value function, $q(s,a;\theta) \approx Q^*(s,a)$, where the parameter vector, θ, is the vector of weights in the deep neural network. Rather than using the table-update approach of traditional Q-learning, DQN updates $q(s,a;\theta)$ by training the neural network (i.e., it updates the weights θ). More details concerning DRL (DQN) will be provided in the following section.

B. Communication Environment

We assume a mesh network of $2N$ nodes, and among them are N nodes acting as transmitters and another N nodes acting as receivers. There are K orthogonal channels. We assume that there are no duplex collisions (i.e., collisions between up/down transmissions of the same communications link. Collisions only occur when two different transmitter nodes attempt to occupy the same channel simultaneously.

In the communication system, there are several types of nodes assumed: *Legacy node*: Occupies one channel constantly; *Hopping node*: Dynamically occupies the channels under a regular pattern; *Intermittent node*: Periodically occupies one specific channel (this could be a TDMA node or a bursty node); *Greedy node*: Occupies the first available channels they observed. It can be regarded as a type of DSA node; *DQN node*: Some variation of the proposed learning node. Note that each of these nodes can apply ACK/NAK information to determine whether or not the transmission was successful.

IV. Deep Reinforcement Learning

A. DQN and Double-DQN

As mentioned before, deep neural networks are used to approximate the state-action value function, $q(s,a;\theta) \approx Q^*(s,a)$. Thus, the DQN can be trained by minimizing the prediction error of $q(s,a;\theta)$.

Supposing that at time step t, the state is s_t and the DQN agent takes an action a_t, and the resulting reward is r_{t+1} and the state moves to s_{t+1}. Then, $(s_t, a_t, r_{t+1}, s_{t+1})$ constitutes an experience-tuple that will be used to train the neural network.

In particular, the prediction error of the neural network at time step t is given by the loss function:

$$L(\theta) = \left(y^{DQN} - q(s_t, a_t; \theta)\right)^2 \tag{3}$$

where $q(s_t, a_t; \theta)$ is the approximated Q-function given by the neural network with weight parameter θ and y^{DQN} is the target output of the neural network based on the new experience $(s_t, a_t, r_{t+1}, s_{t+1})$:

$$y_t^{DQN} = r_{t+1} + \gamma \max_{a'} q(s_{t+1}, a'; \theta) \tag{4}$$

The loss function $L(\theta)$ here can be minimized by the Stochastic Gradient Descent algorithm. After an iteration of the DQN using the specified loss function, the weights θ are updated to θ_{t+1}. The updated DQN with the new approximated Q-function $q(s,a;\theta_{t+1})$is used in decision-making by the DRL agent at time step $t+1$. Note that unlike (2), where only one entry in $q(s,a)$ with a particular (s,a) gets updated, all entries in $q(s,a;\theta_{t+1})$ with different (s,a) are affected by the new parameter θ_{t+1}. Note also that the training of a DQN is different from the training of traditional neural networks in supervised learning, where the network weights are tuned offline. More specifically, the weights of the DQN are updated using previous experience in an online manner.

Although the Deep Q Learning algorithm can be effective, it still has some known problems including overestimating action values under certain conditions. Therefore, [5] provides an advanced algorithm termed Double Deep Q-learning. Since the max operator in Q-learning (2) and DQN (3) uses the same values both to select and evaluate an action, it is more likely to select overestimated values, resulting in overly optimistic value estimates. Thus, the idea of Double Deep Q-learning is to reduce overestimation by decomposing the max operation in the target into action selection and action evaluation.

Compared to DQN, Double DQN is only different in the step of updating the Q-value:

$$y_t^{DDQN} = r_{t+1} + \gamma q\left(s_{t+1}, \arg\max_a q(s_{t+1}, a; \theta_t), \theta_t^-\right) \tag{5}$$

where θ_t and θ_t^- are different sets of parameters in the Double Deep Q network. In the following subsection, we will introduce some implementation variations to help DQN and Double DQN networks converge faster and obtain lower collision rates and higher success rate.

B. Implementation Variations

In this section, we mainly focus on three different implementation variations: Eligibility Trace, Prioritized Experience Replay and Guessing Process.

Eligibility Trace(ET) is described in [12]. The idea behind e-trace is answering the question "What causes the following state? The most frequent states or the most recent states". There are two options: (1) a frequency heuristic (assign credit to the most frequent states) and (2) a recency heuristic (assign credit to the most recent states). E-trace combines these two options in:

$$E_0(s) = 0 \tag{6}$$

$$E_t(s) = \gamma\lambda E_{t-1}(s) + 1(S_t = s) \tag{7}$$

where, λ is a tradeoff parameter and $1(x)$ is the indicator function. The term E_t is then input into DRL's update step:

$$\delta_t = r_{t+1} + \gamma q\left(s_{t+1}, a'; \theta^-\right) - q(s,a;\theta) \tag{8}$$

$$q(s,a;\theta) \equiv q(s,a;\theta) + \partial\delta_t E_t(s) \tag{9}$$

where ∂ is the learning rate. Thus, the memory-tuple which is stored in the memory becomes $\langle s_t, a_t, r_{t+1}, t, s_{t+1} \rangle$.

In DQN and DDQN, experience stored in memory helps to break the temporal correlations by mixing more and less recent experience into the updates. Since some experiences will be used for more than just a single update, experience replay is more efficient and effective. However, each memory has a different influence on the learning process which means the importance of memory can be different. Thus, [13] introduces a variation to measure the importance of memories - Prioritized Experience Replay (PER). The key idea of PER is that an RL agent can learn more effectively from some experiences than from others. In general, experiences may be more or less surprising, redundant, or task-relevant. In this paper, we use the value of the loss function as a criterion to order experiences.

We also introduce another implementation variation that is Guessing Process(GP). GP is based on the idea of a human being's "associative memory". Because DQN and DDQN nodes have already sensed the channels' quality, they can utilize this information to estimate some hypothetical behaviors and put them into the minibatch. Thus, the hypothetical memory tuple is $\langle s_{t-rel}, a_{est}, r_{est}, t_{t-rel}, s_{t+1-est} \rangle$. The process has two main functions: first, since these hypothetical experience-tuples aren't real, they can help the deep network avoid over-fitting; second, since these tuples are based on a channels' quality, the agent has a better chance of gaining a high reward.

Based on two types of DRL and three implementation variations, we examine six types of nodes *DQN0*: DQN without ET and GP(these are the same as the DQN nodes in [10]); *DQN1*: DQN with GP; *DQN2*: DQN with ET and GP; *DQN3*: DDQN with ET and GP; *DQN4*: DDQN with ET; *DQN5*: DDQN with ET and PER.

In the following section, we will simulate the performance of these proposed DQN nodes and compare them.

V. Simulation Results

This section investigates the performance of the proposed DQN nodes via simulation. In the all communication environments, we assume that there exist $K = 10$ discrete frequency channels. For these sets of experiments, we designed our neural network structure to be a fully connected neural network with 4 hidden layers and each layer containing 30, 50, 60 and 40 neurons respectively. The activation function used for the neurons is the ReLU (Rectified Linear Unit) function. When updating the weights of the network, a minibatch of 32 experience-tuples are randomly selected from a memory box of 400 prior experiences for the computation of the loss function. To the minibatch we also add three "guessing" memory-tuples. The RMSProp algorithm is used to conduct a Stochastic Gradient Descent for updating. In order to avoid getting stuck with a suboptimal decision policy before sufficient learning experience, we apply an adaptive ε greedy algorithm which is initially set to 0.1 and is decreased by 0.0001 every time step until its value reaches 0.05.

A. Simple Communication Environment

In this set of simulations, we want to simply verify that the proposed DQN nodes can learn the primary nodes' patterns without prior knowledge. To that end, we use two simple communication environments to illustrate that DQN nodes can learn the pattern of the primary nodes (i.e., legacy nodes, intermittent nodes and greedy nodes).

In the first environment, 6 of 10 channels(#1 to #6) are always occupied by legacy (fixed) nodes. There are also three hopping nodes and their dynamic pattern is:#7#8#9 $\rightarrow$ #8#9#10 $\rightarrow$ #9#10#7 $\rightarrow$ #10#7#8 $\rightarrow$ #7#8#9. Thus, in this system, there is always an available channel that can be used by the DQN node. In this initial set of simulations, we arbitrarily choose to use a DQN3 node (i.e., Double DQN with e-trace and Guessing Process).

In Figure 2, we present the results of the first simulation.

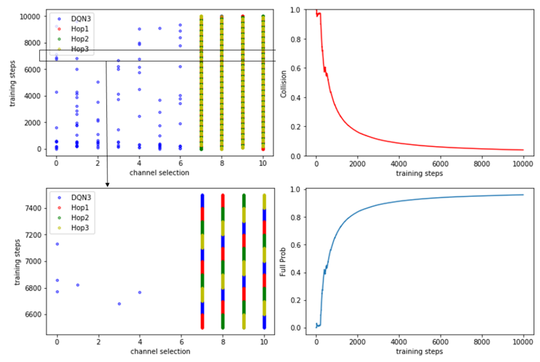


Fig. 2: Performance in Communication Environment 1: Plots on the left show channel occupancy versus time while plots on the right show collision rate (top) and the Probability of Full Spectrum Utilization (bottom).

Specifically, we plot the channels occupied by the DQN node over time (left two plots) as well as the channels occupied by the hopping nodes. Note that for clarity, we do not plot the channels occupied by the legacy nodes. On the right we plot the collisions over time and the probability of full spectrum utilization. We can see that the DQN node learns the pattern of the legacy nodes and hopping nodes and avoids them by successfully predicting the next occupied state. Because ε is a non-zero parameter, DQN nodes still have some probability to select an occupied channel (due to exploration). Thus, the number of collisions will never be zero and probability of full spectrum utilization will never be 1. However, this can be controlled through the exploration rate.

In the second environment, 7 of 10 channels (#1 to #7) are always occupied by legacy nodes. There are also intermittent nodes on three channels: #8 (node occupies this channel for 50 consecutive steps and is idle 100 consecutive steps), #9 (occupies the channel for 100 consecutive steps and is idle for 50 consecutive steps) and #10 (occupies the channel for 149 steps and is idle for 1 step). Meanwhile, there is also a greedy node in this simulation. Thus, our proposed DQN nodes can only use the idle periods of the Intermittent nodes. Here, we again use DQN3 node for testing.

From Figure 3, we can conclude that the DQN nodes

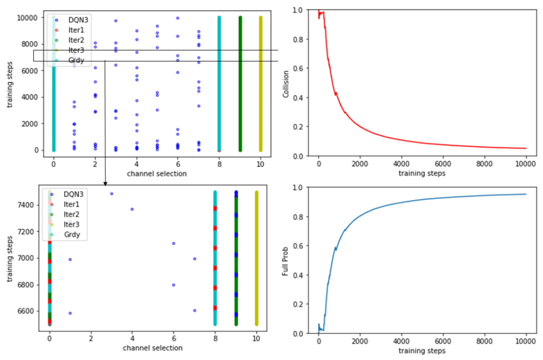


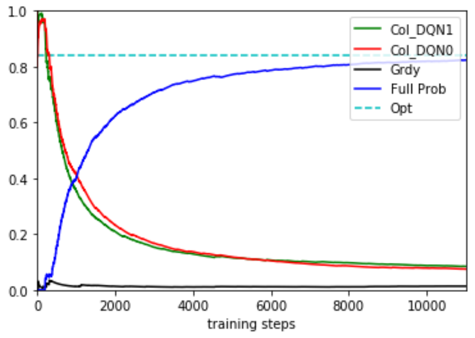
Fig. 3: Performance in Simple Communication Environment 2: Plots on the left show channel occupancy versus time while plots on the right show collision rate (top) and Probability of Full Spectrum Utilization (bottom).

can effectively learn the behavior of the legacy nodes, the greedy node and the intermittent nodes. Because the system does not always have idle channels for the DQN node, the proposed DQN nodes learn to wait until the intermittent nodes are not transmitting. Again, because the exploration rate ε is a non-zero parameter, the DQN nodes still have some probability to select a random channel. Thus, the collision rate does not go to zero and the probability of full spectrum utilization does not go to 1. Note that the primary difference between spectrum utilization and the collision rate is the impact of a node waiting to transmit.

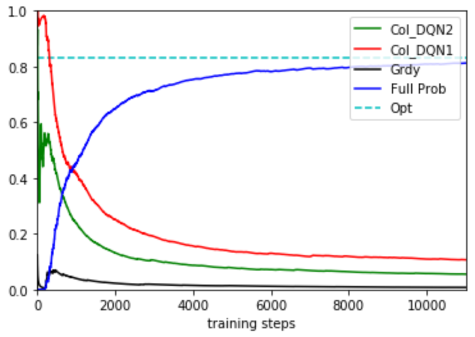
B. Complex Communication Environment

Based on the simulation results of the previous subsection, we can conclude that DQN nodes can learn the different nodes' patterns/behaviors. Thus, in this section, we mainly focus on the each node's collision rate (Col) and probability of full spectrum utilization (Full Prob) performance of the entire system in a more complex communication environment and compare the various implementation variations of the DQN and DDQN using these metrics.

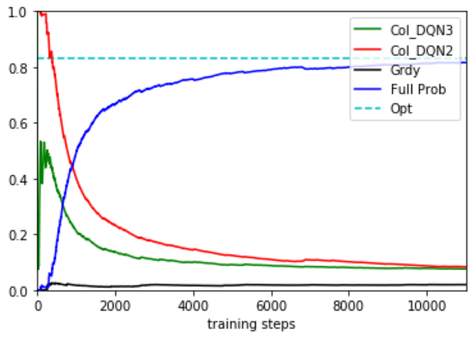
There are five types of nodes in this environment: 3 Legacy nodes(#1 to #3), 3 hopping nodes (dynamic pattern: #4#5#6 $\rightarrow$ #5#6#7 $\rightarrow$ #6#7#8 $\rightarrow$ #7#8#9 $\rightarrow$ #8#9#4 $\rightarrow$ #9#4#5 $\rightarrow$ #4#5#6), 1 greedy node (DSA), two DQN nodes and one intermittent node. The intermittent node uses channel #10 and occupies the channel for 230 consecutive steps and is idle 20 steps. Technically, since DQN nodes have a 5% possibility of random selection and the intermittent nodes are bursty, the system's optimal probability ('Opt' in Figure 4) is approximately 85% (approximately 8% loss from the



(a) DQN0 and DQN1



(b) DQN1 and DQN2

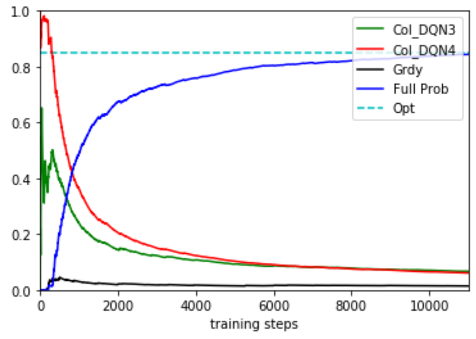


(c) DQN2 and DQN3

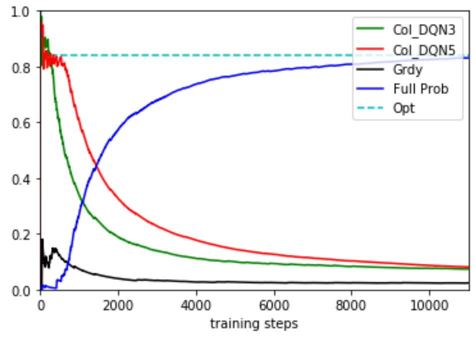
Fig. 4: Performance in Complex Environment System with Different Pairs of Secondary Nodes ('Col' = collisions, 'Full Prob' = probability of full spectrum utilization, 'Grdy' = greedy node collisions)

intermittent node and 8% from DQN's random action). All nodes have perfect observations and take actions at same time.

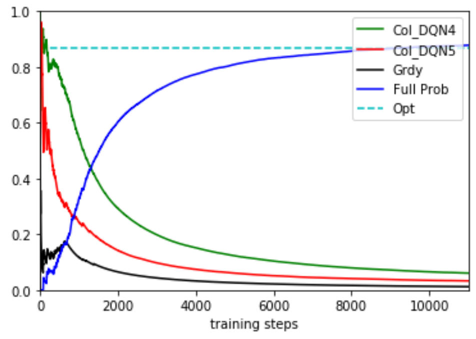
Compared with the previous simulations, due to the introduction of the greedy node (a DSA node) combined with the various types of nodes, the entire system is more complicated, so the learning rate of the DQN node is slower, but still reasonable. From the results shown in Figures 4 and 5, we find that all of the DQN nodes can learn the DSA node's behavior and avoid collisions with it (in addition to the other node types). However, due to the different implementation variations employed, the learning performance of each node is different. Figure 4(a) and (b)



(a) DQN3 and DQN4



(b) DQN3 vs DQN5



(c) DQN4 and DQN5

Fig. 5: Performance in Complex Environment System with Different Pairs of Secondary Nodes ('Col' = collisions, 'Full Prob' = probability of full spectrum utilization, 'Grdy' = greedy node collisions)

illustrate that the Eligibility Trace and Guessing Process can accelerate the rate of convergence and improve system performance. Figure 4(c) illustrates that, using the same implementation variants, a Double DQN performs better than a DQN in this environment. Figures 5 (a) and (b) illustrate that, under the same DRL structure, DQN3 (a DDQN with the trace and guess process) converge faster than others and different implementation variations have different (albeit similar) performance. Meanwhile, each DQN node also learns the other's behavior, but the introduced DSA node slows the learning performance of the DQN nodes to some extent. Thus, we can conclude that: First, different strategies have different effects on learning effectiveness; Second, collisions (after convergence) mainly come from two effects: The DQN node's random actions (i.e., exploration) and the inability to fully avoid the DSA node; Third, the reason the full utilization probability $< 100\%$ is due to the DQN's random action and the intermittent nodes; Fourth, DQN nodes take a little longer time to converge in the more complex communication environment; Fifth, all the proposed techniques can eventually achieve optimal performance. In the end, Double Deep Q learning with trace and guess process provides the best performance among all variations.

VI. Conclusion

In this paper, we have considered the dynamic spectrum access problem in a general and complex scenario where the system has several channels and the statistics of the interfering nodes is unknown. We have applied a DQN approach, a Double DQN approach and several implementation variations to find the optimal access policy via online learning. Through simulations, we have shown that our proposed DRL nodes are able to learn different nodes' patterns without any prior knowledge. We have also shown that our DRL can achieve near-optimal performance even in more complex scenarios.

References

[1] Qing Zhao and Brian M Sadler. A survey of dynamic spectrum access. *IEEE signal processing magazine*, 24(3):79–89, 2007.

[2] Ian F Akyildiz, Won-Yeol Lee, Mehmet C Vuran, and Shantidev Mohanty. Next generation/dynamic spectrum access/cognitive radio wireless networks: A survey. *Computer networks*, 50(13):2127–2159, 2006.

[3] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[5] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 16, pages 2094–2100, 2016.

[6] Pranav Sakulkar and Bhaskar Krishnamachari. Online learning of power allocation policies in energy harvesting communications. In *Signal Processing and Communications (SPCOM), 2016 International Conference on*, pages 1–5. IEEE, 2016.

[7] Enrique Stevens-Navarro, Yuxia Lin, and Vincent WS Wong. An mdp-based vertical handoff decision algorithm for heterogeneous wireless networks. *IEEE Transactions on Vehicular Technology*, 57(2):1243–1254, 2008.

[8] Mingzhe Chen, Ursula Challita, Walid Saad, Changchuan Yin, and Mérouane Debbah. Machine learning for wireless networks with artificial intelligence: A tutorial on neural networks. *arXiv preprint arXiv:1710.02913*, 2017.

[9] Mehdi Bennis and Dusit Niyato. A q-learning based approach to interference avoidance in self-organized femtocell networks. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 706–710. IEEE, 2010.

[10] Shangxing Wang, Hanpeng Liu, Pedro Henrique Gomes, and Bhaskar Krishnamachari. Deep reinforcement learning for dynamic multichannel access in wireless networks. *IEEE Transactions on Cognitive Communications and Networking*, 2018.

[11] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

[12] Doina Precup, Richard S Sutton, and Satinder P Singh. Eligibility traces for off-policy policy evaluation. In *ICML*, pages 759–766. Citeseer, 2000.

[13] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.