Accelerating Model-Free Reinforcement Learning With Imperfect Model Knowledge in Dynamic Spectrum Access

Lianjun Li, Lingjia Liu, *Senior Member, IEEE*, Jianan Bai, Hao-Hsuan Chang, *Graduate Student Member, IEEE*, Hao Chen, Jonathan D. Ashdown, *Member, IEEE*, Jianzhong Zhang, *Fellow, IEEE*, and Yang Yi, *Senior Member, IEEE*

***Abstract*—Current studies that apply reinforcement learning (RL) to dynamic spectrum access (DSA) problems in wireless communications systems mainly focus on model-free RL (MFRL). However, in practice, MFRL requires a large number of samples to achieve good performance making it impractical in real-time applications such as DSA. Combining model-free and model-based RL can potentially reduce the sample complexity while achieving a similar level of performance as MFRL as long as the learned model is accurate enough. However, in a complex environment, the learned model is never perfect. In this article, we combine model-free and model-based RL, and introduce an algorithm that can work with an imperfectly learned model to accelerate the MFRL. Results show our algorithm achieves higher sample efficiency than the standard MFRL algorithm and the Dyna algorithm (a standard algorithm integrating model-based RL and MFRL) with much lower computation complexity than the Dyna algorithm. For the extreme case where the learned model is highly inaccurate, the Dyna algorithm performs even worse than the MFRL algorithm while our algorithm can still outperform the MFRL algorithm.**

***Index Terms*—Dynamic spectrum access (DSA), imperfect model, reinforcement learning (RL), training acceleration, wireless communications systems.**

I. Introduction

A. Reinforcement Learning and Its Variations

DEEP model-free reinforcement learning (RL) is a powerful tool and has been successfully applied in many areas from playing video games to robotic control. However, model-free RL (MFRL) is also known for the high sampling complexity [1], i.e., it requires a large amount of samples to learn a good policy. In applications where it is easy to interact with the environment to acquire samples, the high sampling complexity may not be an issue. However, for many real-time applications, high sample complexity is not desirable: 1) a larger amount of samples requires longer acquisition time resulting in a slow convergence rate and 2) interacting with the environment is expensive in real-time applications, especially in the initial stage, the trial and error behavior of the RL agent may impact the behaviors of other agents. The high sampling complexity prevents the wide adoption of MFRL in real-time applications.

On the other hand, model-based RL, in general, achieves low sampling complexity [2] through learning a model of the underlying environment. Once an appropriate model is learned, the RL agent can use it to plan actions through heuristic search [3] or optimal control [4] without interacting with the real environment. However, the performance of the model-based approach is limited by the accuracy of the learned model where the learned model is usually imperfect in complex environments. Direct utilization of the model-based approach may harm the system performance.

In [5], Dyna introduces a new class of RL algorithms that use models to accelerate MFRL. Many works have been done to extend the Dyna algorithm to reduce the sampling complexity: Nagabandi *et al.* [6] used model-based RL to obtain an initial policy, and used it as a starting point for the MFRL. Gu *et al.* [7] fitted a linear model to generate imaginary samples for MFRL training. Racanière *et al.* [8] used the information gained from imaginary rollouts as inputs to improve the policy. Feinberg *et al.* [9] used a learned model to perform H steps target value update instead of the standard one-step update. Buckman *et al.* [10] also performed H steps target value update, but their method trains multiple models and uses the ensemble of models' outputs to improve the target value accuracy. However, all of them are designed for robotic control applications thus may not be suitable for other applications for the following reasons: 1) the deterministic dynamic assumption does not hold in many applications with the stochastic environment and 2) algorithms can work with the imperfect model are inspired by ensemble learning that train multiple models and value approximation networks, the high computation complexity is not desirable as the RL agent is usually implemented with limited power and computation resources.

Manuscript received March 3, 2020; revised March 24, 2020; accepted March 30, 2020. Date of publication April 16, 2020; date of current version August 12, 2020. The work of Lianjun Li, Lingjia Liu, Jianan Bai, Hao-Hsuan Chang, and Yang Yi was supported in part by the U.S. National Science Foundation under Grant ECCS-1811497 and Grant CCF-1937487. *(Corresponding author: Lingjia Liu.)*

Lianjun Li, Lingjia Liu, Jianan Bai, Hao-Hsuan Chang, and Yang Yi are with the Electrical and Computer Engineering Department, Virginia Tech, Blacksburg, VA 24061 USA (e-mail: ljliu@ieee.org).

Hao Chen and Jianzhong Zhang are with the Standards and Mobility Innovation Lab, Samsung Research America, Plano, TX 75023 USA.

Jonathan D. Ashdown is with the Information Directorate, Air Force Research Lab, Rome, NY 13441 USA.

Digital Object Identifier 10.1109/JIOT.2020.2988268

B. Dynamic Spectrum Access in 5G Networks

To support the fast-growing wireless traffic volume and the massive number of Internet-of-Things (IoT) devices, 5G networks are required to provide 1000 times higher mobile data volume per area, 10–100 times higher number of connected devices, and 10–100 times higher user data rate as compared to 4G networks [11]. The main difficulty in meeting those requirements is the scarcity of the spectrum caused by the spectrum segmentation and fixed allocation policy. Motivated by the fact that most licensed spectrum is under utilized in temporal and spatial domains, dynamic spectrum access (DSA) is viewed by various vendors and operators across the globe as a key enabling technology for 5G [12]. Various DSA models have been introduced to improve spectrum utilization under different system settings [13]. In this article, we mainly focus on the *spectrum overlay* model, where secondary users (SUs) are allowed to opportunistically access the radio spectrum that is not used by the licensed primary users (PUs).

In this article, we introduce a lightweight architecture that only one model and one value approximation network need to be trained. Based on the derivation of the relationship between model inaccuracy and RL performance, an uncertainty-aware (UA) algorithm is introduced to accelerate MFRL with an imperfect model by automatically filtering out inaccurate samples generated by the learned model. We apply the algorithm to the important application use case of DSA in 5G networks and evaluate its performance in relevant scenarios. Since it has been demonstrated that MFRL can achieve near-optimal reward performance in DSA-related problems [14]–[17], this article focuses on improving the sampling efficiency of the MFRL algorithm.

The main contributions of this article are as follows.

1) It provides a lightweight algorithm that can accelerate MFRL while keeping the computation cost low. Therefore, it significantly extends the application scenarios of RL to low-cost devices.
2) By deriving the relationship between the model inaccuracy and the RL performance, we mathematically show the performance guarantee of the algorithm.
3) The first study that combines model based and MFRL for the DSA problem in the wireless communications system. Various DSA settings, such as multiple SUs and imperfect sensing, are considered.

The remainder of this article is organized as follows. Section II introduces model-free and model-based RL. Section III describes the DSA problem formulation and the RL agent architecture. Section IV introduces the UA algorithm. Section V extends the algorithm to the multiple-SU case. The experimental results are discussed in Section VI. Section VII concludes this article.

II. Model-Free and Model-Based Reinforcement Learning

RL is one paradigm of machine learning, where an agent learns what are the best actions to take in the environment. Unlike supervised learning, the RL agent does not need labeled training data; instead, it interacts with the environment to acquire data and learns from it. Depending on whether a model of the environment is needed, there are two types of RL: 1) model-based RL and 2) MFRL.

MFRL learns a value function or policy directly through interacting with the environment. Fig. 1 column 1 illustrates the iterative procedure of MFRL, the RL agent improves the value function or policy based on the experience data acquired from the environment and then, uses the updated policy to interact with the environment to acquire more experience. Based on whether explicitly learns a policy, there are three types of MFRL: 1) value based, which learns a value function and derives a policy from it, e.g., deep Q-learning [18]; 2) policy based, which learns a policy directly, e.g., policy gradient [19]; and 3) actor–critic (AC), which is a combination of value-based and policy-based MFRL, where a critic learns a value function and uses it to guide actor's policy learning [20].

Model-based RL (MBRL), as depicted in Fig. 1 column 2, on the other hand, learns a model to estimate the transition probabilities $p(s', r|s, a)$ of the environment and then, improves the value function or policy by interacting with the model instead of the real environment; this process is called planning. There are several ways of planning, such as dynamic programming (DP) [21], heuristic search [3], and optimal control [4].

MFRL is capable of learning a wide range of complex tasks at the cost of a large number of samples, whereas MBRL is sample efficient but suffers from model inaccuracy. In [5], Dyna introduces a new class of RL algorithms that use models to accelerate MFRL. The procedure of the Dyna algorithm is depicted in Fig. 1 column 3. The real experience sampled from the environment has two roles: one is to directly improve the value function or policy, i.e., direct RL; another is to learn a model. It uses the model to generate imaginary experience to improve the value function or policy, i.e., indirect RL.

III. Network Setup

Our goal is to improve the sampling efficiency of the MFRL with imperfect model knowledge while keeping the computation cost low. The particular use case we are interested in is the DSA scenario of 5G networks. In this section, we will provide a detailed description of the network setup of the DSA as well as the architecture of the RL agent we are developing.

A. Dynamic Spectrum Access

In a DSA network, there are totally N PUs, each PU occupies a particular wireless channel. DSA SU co-exists with PUs and tries to access one of N channels if the channel is not used by PU at each time slot.

The PU's activity follows its own incoming traffic distribution, e.g., Markovian, Poisson, etc. At any time, each PU can be in one of the two states: *Inactive* meaning the underlying wireless channel is idle (1); and *Active* meaning the underlying wireless channel is busy (0). Accordingly, the status of N PUs (channels) at time t can be represented as an N-dimensional vector

$$\boldsymbol{Z_t} = \left[z_t^1, z_t^2, \ldots, z_t^N\right]^T, \; z_t^n \in \{0, 1\}, \; n = 1, 2, \ldots, N.$$

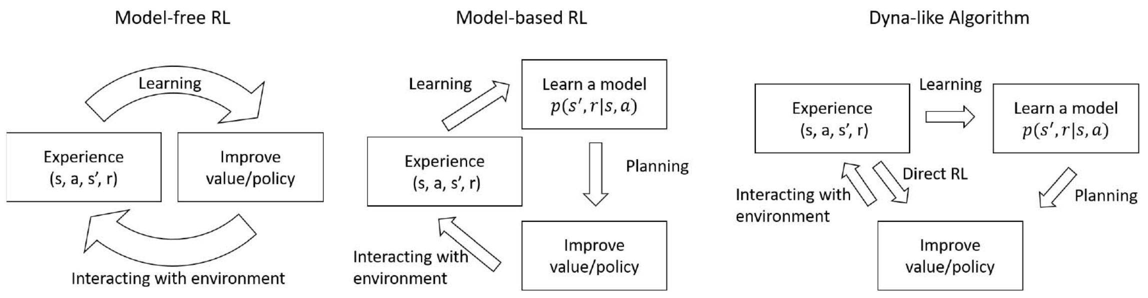

Fig. 1. RL procedure.

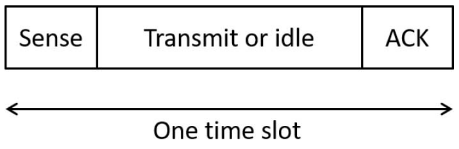

Fig. 2. Time slot composed of three periods.

The corresponding DSA network is assumed to adopt a slot structure in the time domain, which is termed as frame-based equipment (FBE) by the European Telecommunications Standards Institute (ETSI) [22]. This setting is in line with most modern mobile wireless standards and DSA networks [14], [15], [23] in which PU's activity changes according to the underlying traffic pattern and the scheduling of the base station at the beginning of each time slot. At SU, each time slot is further divided into three periods: 1) sensing period; 2) transmitting period; and 3) acknowledgment period, shown in Fig. 2.

Sensing: At the beginning of each time slot, SU chooses a set of channels to sense. The sensing result of channel n at time t is denoted as $e_t^n \in \{0(\text{Active}), 1(\text{Inactive})\}$. If sensing is perfect, then $e_t^n = z_t^n$. However, in general, this equality does not hold due to the sensing error. More detail about the sensing error will be discussed in Section III-B.

Transmitting: If the sensing result is *Inactive*, SU accesses the channel and transmits data during the transmitting period. Otherwise, SU stays idle.

Acknowledgment: At the end of each time slot, the SU receiver[1] acknowledges channel access result to the SU transmitter. The access result of channel n at time t is

$$k_t^n = \begin{cases} 1(\text{Succeed}), & e_t^n = 1 \text{ and } z_t^n = 1 \\ 0(\text{Fail}), & \text{otherwise.} \end{cases} \tag{1}$$

Throughout this article, we assume acknowledgment can be received by the SU transmitter without error.

B. Sensing Error

Sensing error can be characterized by two probabilities: 1) false alarm probability p_f and 2) detection probability p_d, defined as

$$p_f = P(e = 0|z = 1) \tag{2}$$
$$p_d = P(e = 0|z = 0).^2 \tag{3}$$

[1]The SU receiver knows the channel access is succeed when it decodes the received message and identifies its unique ID. We assume the SU receiver can decode ID information correctly, unless stated otherwise.

The false alarm occurs when the channel is *Inactive* ($z = 1$) but sensing result is *Active* ($e = 0$), in this case, an access opportunity will be overlooked. Detection occurs when the channel is *Active* ($z = 0$) and the sensing result is consistent with the channel state ($e = 0$). If perfect sensing is assumed, then $p_f = 0$ and $p_d = 1$.

With the false alarm probability p_f and detection probability p_d, the channel access succeed/fail probability $P(k)$ and the channel active/inactive probability $P(z)$ can be related as

$$P(k = 1) = P(z = 1) \times \left(1 - p_f\right) \tag{4}$$
$$\begin{aligned} P(k = 0) &= P(z = 1) \times p_f + P(z = 0) \times p_d \\ &\quad + P(z = 0) \times (1 - p_d) \\ &= P(z = 1) \times p_f + P(z = 0) \end{aligned} \tag{5}$$

meaning channel access succeed only when the channel is *Inactive* ($z = 1$) and sensing false alarm does not happen, otherwise channel access fails. In case of perfect sensing, it can be easily seen that

$$P(k = 1) = P(z = 1) \tag{6}$$
$$P(k = 0) = P(z = 0). \tag{7}$$

C. Architecture of the RL Agent

In our design, an RL agent is introduced at SU to assist the spectrum access. To be specific, the RL agent suggests what channels SU should sense at the beginning of each time slot; such suggestion is based on SU's history observations, and the goal is to maximize the data transmission opportunity of SU.

The RL agent in SU consists of one MFRL algorithm using deep Q network (DQN) [18], one model that learns the environment through a neural network, and one UA algorithm that can incorporate the imperfect model knowledge.

1) Model-Free RL Algorithm: The MFRL algorithm is implemented using DQN with one evaluation network and one target network. It is trained through supervised learning by minimizing the loss function

$$\mathcal{L}(\boldsymbol{w}) = \left(\mathcal{T}\left(s, a, \boldsymbol{w}^-\right) - Q(s, a, \boldsymbol{w})\right)^2 \tag{8}$$

with target $\mathcal{T}(s, a, \boldsymbol{w}^-) = r + \gamma \max_{a'} Q^-(s', a', \boldsymbol{w}^-)$, where s and a are the current state and action, s' and a' are the next state and action, r is the current reward, $\gamma \in (0, 1)$ is the discount factor of the reward, $\boldsymbol{w}$ are weights of the evaluation

[2]For notation simplification, here we omit t and n, and represent z_t^n and e_t^n as z and e.

network $Q(s, a, \boldsymbol{w})$, and $\boldsymbol{w}^-$ are weights of the target network $Q^-(s', a', \boldsymbol{w}^-)$ periodically copied from $\boldsymbol{w}$. The standard ϵ-greedy policy is adopted by the MFRL algorithm to balance exploration and exploitation.

The input state of the MFRL is historical observations of channel access result

$$\boldsymbol{S_t} = [\boldsymbol{O_{t-1}}, \boldsymbol{O_{t-2}}, \dots, \boldsymbol{O_{t-M}}] \tag{9}$$

where M is the number of history time slots the agent considers. $\boldsymbol{O_t}$ is the observation at time slot t.

It is very important to note that in reality there are practical hardware and power constraints preventing from SU to sense all available N channels continuously [15], [23]. This is especially true for the application of DSA technologies to 5G massive machine-type communications (mMTC) where SUs are low-cost devices. Accordingly, to make our study more meaningful, we assume SU can only sense a subset of the N channels. This partial observation assumption makes our analysis much more relevant to scenarios that are important to 5G networks.

Without loss of generality, we assume each SU can only sense one channel at each time slot, so $\boldsymbol{O_t}$ contains which channel has been sensed and the channel access result: $\boldsymbol{O_t} = [n, k_t^n]$. The output of MFRL is the suggested action, denoted as

$$a_t \in \{0, 1, 2, \dots, N\} \tag{10}$$

where $a_t = n,\ 0 < n \le N$ means at time t, the RL agent suggests SU to sense channel n, and $a_t = 0$ means the SU does not sense or access any channel at time t.

The goal of the MFRL is to maximize the total reward $\sum_{t=0}^{\infty} r_t$, r_t is defined as

$$r_t = \begin{cases} 1, & a_t = n, k_t^n = 1,\ 0 < n \le N \\ -1, & a_t = n, k_t^n = 0,\ 0 < n \le N \\ 0, & a_t = 0. \end{cases} \tag{11}$$

That is, if SU successfully accessed the channel suggested by the RL agent, the corresponding reward is $+1$; if the access failed, the corresponding reward is -1; otherwise, if SU did not sense or access any channels, the corresponding reward is 0.

2) Underlying Model for Environment: A model in the RL agent learns the transition probability of the environment and generates imaginary samples to accelerate the MFRL training. It is implemented as a deep neural network parameterized by $\boldsymbol{\theta}$, the input of the model is the historical observations of channel access result $\boldsymbol{S_t}$ concatenated with the action a_t suggested by the RL agent, the output of the model is $\hat{\boldsymbol{S}}_{\boldsymbol{t+1}}$,[3,4] the predicted state at time $t+1$, i.e.,

$$\hat{\boldsymbol{S}}_{\boldsymbol{t+1}} = \text{model}(\boldsymbol{S_t}, a_t, \boldsymbol{\theta}). \tag{12}$$

[3]In general, the model should predict both the next state and the corresponding reward. However, in our problem setting, predicting the next state is sufficient as the corresponding reward is solely determined by the next state as shown in (11).

[4]More specifically, in our implementation, the model predicts $P(k_t^n = 1)$, the probability of successfully accessing the channel n suggested by the action a_t. Then, a sample of k_t^n will be drawn from $P(k_t^n)$. Based on the sample of k_t^n, we can generate $\hat{\boldsymbol{O}}_{\boldsymbol{t}}$ and $\hat{\boldsymbol{S}}_{\boldsymbol{t+1}}$.

Algorithm 1 Dyna With DQN and the Neural Network Model for Environmental Learning

```
1:  Initialize evaluate network $Q(\boldsymbol{S}, a, \boldsymbol{w})$ and target network
    $Q^-(\boldsymbol{S}, a, \boldsymbol{w}^-)$
2:  Initialize model network model$(\boldsymbol{S}, a, \boldsymbol{\theta})$
3:  while Not exhausted do
4:      $a \leftarrow$ action based on $\epsilon$-greedy policy and current
        state $\boldsymbol{S}$
5:      Receive reward $r$, next state $\boldsymbol{S'}$ from environment
6:      Update weights $\boldsymbol{w}$ of the evaluate network with loss
        function defined by (8)
7:      Update weights $\boldsymbol{\theta}$ of model network with loss function
        defined by (13)
8:      repeat
9:          $a \leftarrow$ action based on $\epsilon$-greedy policy and current
            state $\boldsymbol{S}$
10:         Generate imaginary sample from model $\hat{\boldsymbol{S}}', \hat{r} \leftarrow$
            model$(\boldsymbol{S}, a, \boldsymbol{\theta})$
11:         Update weights $\boldsymbol{w}$ of the evaluate network with loss
            function defined by (8)
12:     until $n$ times
13:     for every m steps do
14:         replace target network weights $w^-$ with evaluate
            network weights $w$
```

The model is trained by supervised learning, the training data are samples from the real environment $(\boldsymbol{S_t}, a_t, \boldsymbol{S_{t+1}})$ and the loss function is defined as

$$\mathcal{L}(\boldsymbol{\theta}) = \left\| \hat{\boldsymbol{S}}_{\boldsymbol{t+1}} - \boldsymbol{S_{t+1}} \right\|_2^2. \tag{13}$$

IV. Uncertainty-Aware Algorithm With Imperfect Model Knowledge

If the learned model is accurate enough, a standard Dyna algorithm using our RL architecture (Algorithm 1) is sufficient to improve the sampling efficiency. However, if the learned model is biased, the RL agent will be trained by biased imaginary samples and converges to suboptimal policy. In reality, the learned model is usually imperfect, especially in the case of partial observation like the DSA scenario discussed in this article. Therefore, an algorithm that can work with imperfect model knowledge is critical.

A. Key Idea and the Toy Example

If we can identify the model uncertainty, i.e., how confident we are for each imaginary sample generated by the model, we can discard bad (high uncertainty) samples and keep the good ones for the underlying RL training. This is the key idea of our UA algorithm, which can be illustrated by a toy example[5] in Fig. 3. In this example, the average reward versus real RL step is plotted for three algorithms: 1) baseline (BL) is the standard MFRL algorithm (DQN); 2) Dyna with fixed imaginary step (FS) is the implementation of Algorithm 1 where $n = 5$ imaginary steps are generated for each real step; and 3) UA is our uncertainty-aware algorithm. To illustrate the

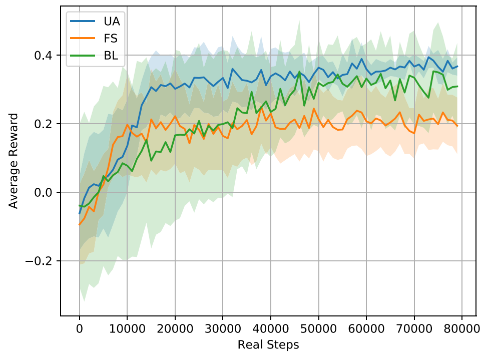

Fig. 3. Toy example with overfitted model.

negative effect of an inaccurate model, we show the extreme case where the model used for both FS and UA is intentionally trained with a small amount of data and then fixed, i.e., the model is overfitted. We can see that with an inaccurate model, the performance of the Dyna algorithm is even worse than the model-free algorithm, while our UA algorithm can achieve 5× sample efficiency compare to the model-free algorithm.

In the rest of this section, we will study how the model imperfectness affects the MFRL and introduces an algorithm that can automatically select good imaginary samples to train the underlying MFRL agent.

B. Imperfect Model Knowledge

Recall that when the RL agent is interacting with the real environment, it is updated by the real target value $\mathcal{T} = r+\gamma \max_{a'} Q^-(s', a')$ as in (8). When we use the model generated imaginary samples to train the RL agent, it is updated by the imaginary target value $\hat{\mathcal{T}} = \hat{r} + \gamma \max_{a'} Q^-(\hat{s}', a')$, where $\hat{r}$ and $\hat{s}'$ are the reward and the next state predicted by the model. Ideally, we want the average target value generated by the model to be the same as that sampled in the real environment, i.e., $E[\hat{\mathcal{T}}] = E[\mathcal{T}]$, meaning the model perfectly learned the real environment. Unfortunately, the model is imperfect in most cases. Let us denote the actual transition probability that governs the Markov decision process of the environment as P and the transition probability learned by model as G. It is important to characterize the relationship between the model imperfectness (distance between P and G) and the target value error $|E[\mathcal{T}] - E[\hat{\mathcal{T}}]|$.

We adopt the following notations as follows.

1) The total variation distance between two distributions P and G is defined as

$$\delta(P, G) = \frac{1}{2}\|P - G\|_1 \tag{14}$$

where $\|\cdot\|_1$ is the L_1 norm.

2) Conditioned on current state s, the actual probabilities of accessing channel n to be *Succeed* and *Fail* are $P(k_t^n = 1|s)$ and $P(k_t^n = 0|s)$, respectively. To simplify notation, we denote them as $p^1_{n|s}$ and $p^0_{n|s}$, and let $\boldsymbol{P_{n|s}} = [p^1_{n|s}, p^0_{n|s}]$. Similarly, we define the model learned probability to be $\boldsymbol{G_{n|s}} = [g^1_{n|s}, g^0_{n|s}]$. The total variation distance between $\boldsymbol{P_{n|s}}$ and $\boldsymbol{G_{n|s}}$, conditioned on s, can be expressed as

$$\delta(\boldsymbol{P_n}, \boldsymbol{G_n}|s) = \frac{1}{2}\left\|\boldsymbol{P_{n|s}} - \boldsymbol{G_{n|s}}\right\|_1. \tag{15}$$

3) For simplicity, we write $\max_{a'} Q^-(s', a') = V(s')$, so

$$\mathcal{T} = r + \gamma V(s'). \tag{16}$$

1) Relation Between $\delta(\boldsymbol{P_n}, \boldsymbol{G_n}|s)$ and $|E[\mathcal{T}] - E[\hat{\mathcal{T}}]|$: We can show (see the Appendix)

$$\left|E[r] - E\left[\hat{r}\right]\right| \le 2\delta(\boldsymbol{P_n}, \boldsymbol{G_n}|s) \tag{17}$$

$$\left|E\left[V(s')\right] - E\left[V\left(\hat{s}'\right)\right]\right| \le 2\delta(\boldsymbol{P_n}, \boldsymbol{G_n}|s) \max_{s'} |V(s')|. \tag{18}$$

Accordingly, we have

$$\begin{aligned}\left|E[\mathcal{T}] - E\left[\hat{\mathcal{T}}\right]\right| &= \left|E\left[r + \gamma V(s')\right] - E\left[\hat{r} + \gamma V\left(\hat{s}'\right)\right]\right| \\ &\le \left|E[r] - E\left[\hat{r}\right]\right| + \gamma\left|E\left[V(s')\right] - E\left[V\left(\hat{s}'\right)\right]\right| \\ &\le 2\delta(\boldsymbol{P_n}, \boldsymbol{G_n}|s)\left(1 + \gamma \max_{s'} |V(s')|\right). \end{aligned} \tag{19}$$

This suggests that the average target value error can be bounded by the model imperfectness and consequently, $\delta(\boldsymbol{P_n}, \boldsymbol{G_n}|s)$ is a good metric to identify the accuracy of the learned model. To simplify notation, we will omit n and s and denote $\delta(\boldsymbol{P_n}, \boldsymbol{G_n}|s)$ as $\delta(P, G)$ for the rest of this article.

2) Relation Between $\delta(P, G)$ and $\delta(G(t-1), G(t))$: In practice, P is unknown and $\delta(P, G)$ cannot be calculated directly. Let $G(t)$ denotes the distribution learned by the model at time t. We show that the time difference of distributions learned by the model $\delta(G(t-1), G(t))$ can be used to bound $\delta(P, G(t))$ (see the Appendix)

$$\delta(G(t), P) \le \frac{1}{1-\lambda}\delta(G(t-1), G(t)) + \epsilon. \tag{20}$$

Therefore, it can be used as a measure of model accuracy.

C. Uncertainty-Aware Algorithm

As shown in the previous section, $\delta(G(t-1), G(t))$ can be used to measure model accuracy. Our algorithm uses $\delta(G(t-1), G(t))$ as an indicator to select good samples from all generated samples. More specifically, besides the up-to-date model, we also save an older version of the model in the system for calculating $\delta(G(t-1), G(t))$. For each sample generated by the up-to-date model, $\delta(G(t-1), G(t))$ will be calculated, if it is larger than a threshold η, the sample will not be selected for the MFRL training. This is because when the two consecutive models have a large disagreement with a specific sample, we know the underlying model has large uncertainties about the sample accuracy, therefore, it should not be used to train the underlying RL. The UA algorithm is described in Algorithm 2.

V. Extension to Multiple SUs

When there are multiple SUs present in the network, the underlying problem can be formulated as a multiagent RL (MARL) problem. Depending on the level of cooperation

[5]In this experiment, we set the number of channel $N = 4$; no sensing error, i.e., $k_t^n = e_t^n = z_t^n$; each algorithm was simulated five times with randomly initialized neural network weights. The solid line is the mean value, and the shaded area represents the standard deviation among the five simulation runs.

Algorithm 2 UA Algorithm

```
1: Initialize evaluate network $Q(\boldsymbol{S}, a, \boldsymbol{w})$ and target network $Q^{-}(\boldsymbol{S}, a, \boldsymbol{w}^{-})$
2: Initialize model network model$(\boldsymbol{S}, a, \boldsymbol{\theta})$, old model model$_{\text{old}}(\boldsymbol{S}, a, \boldsymbol{\theta}^{-})$
3: **while** Not exhausted **do**
4:     $a \leftarrow$ action based on $\epsilon$-greedy policy and current state $\boldsymbol{S}$
5:     Receive reward $r$, next state $\boldsymbol{S'}$ from environment
6:     Update weights $\boldsymbol{w}$ of the evaluate network with loss function defined by (8)
7:     Replace the old model with current model, $\boldsymbol{\theta}^{-} \leftarrow \boldsymbol{\theta}$
8:     Update weights $\boldsymbol{\theta}$ of model network with loss function defined by (13)
9:     **repeat**
10:        $a \leftarrow$ action based on $\epsilon$-greedy policy and current state $\boldsymbol{S}$
11:        Generate imaginary sample from model $\hat{\boldsymbol{S}}', \hat{r} \leftarrow$ model$(\boldsymbol{S}, a, \boldsymbol{\theta})$
12:        **if** $\delta(G(t-1), G(t)) \leq \eta$ **then**
13:            Update weights $\boldsymbol{w}$ of the evaluate network with loss function defined by (8)
14:    **until** $n$ times
15:    **for** every m steps **do**
16:        replace target network weights $w^{-}$ with evaluate network weights $w$
```

needed among these RL agents during the training and the execution, MARL can be categorized into three major types: 1) fully centralized; 2) centralized training with distributed execution; and 3) fully distributed.

1) *Fully Centralized:* MARL is implemented as a centralized mega agent RL, the input state of the mega agent is the concatenated states of all agents and the outputs are actions for all agents. We call it fully centralized because this type of MARL requires gathering information from all agents in both training and execution phases, i.e., centralized training and execution.
2) *Centralized Training With Distributed Execution:* Each RL agent selects action based on its local observation, meaning no information exchange during execution, i.e., distributed execution. While in the training phase, the reward used for all agents is a single team reward (e.g., the sum reward of all agents). This requires collecting information from all agents, so the training is centralized.
3) *Fully Distributed:* Each RL agent selects action based on its local observation and trained based on its individual reward. Meaning no information exchange is needed in both the training and execution phase. Therefore, it is fully distributed.

There are many studies focus on the reward performance of different types of MARL in DSA [24]–[26]. However, it is beyond the scope of this article as we are focusing on improving the sampling efficiency. The UA algorithm introduced in Section IV can be applied to all MARL types. Without loss of generality, we choose the fully distributed MARL to apply and show the performance of our algorithm. The system architecture is depicted in Fig. 4. Each SU is equipped with an independent RL agent, which includes a model-free DQN, a model, and the UA algorithm. The reward for each SU in MARL is the same as defined in (11). One exception is that when multiple SUs interfere with each other by accessing the same channel at the same time slot, the reward is -1 for each of those interfering agents.

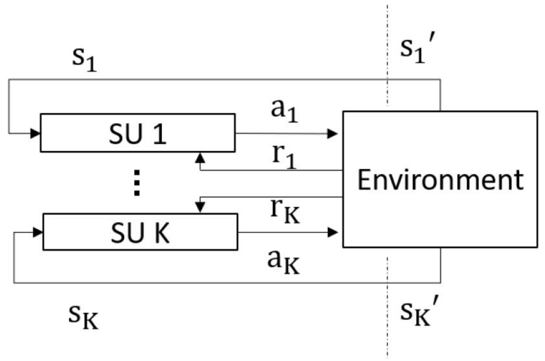

Fig. 4. Fully distributed MARL.

VI. Experiments

Having the UA algorithm introduced, we are going to evaluate its performance under the DSA scenario. We first present details of the DQN and the model network, then talk about the DSA scenario setup, and finally, show the performance of the algorithm.

A. DQN and Model Network

The DQN is implemented as a double-hidden-layer multilayer perceptron (MLP) with the ReLU activation function. Its output layer does not have the activation function due to the output Q values being continuous real numbers. The input size is equal to the length of state $\boldsymbol{S}$, which depends on the length of observation vector $\boldsymbol{O}$ and the number of history time slots M under consideration. The observation $\boldsymbol{O}$ is a sparse vector with $N-1$ zero elements and one nonzero element at the nth position. The nonzero element (1 or -1) represents the access of channel n was *Succeed* or *Fail*, correspondingly. One exception is when SU did not sense any channel, $\boldsymbol{O}$ will be a vector of all zeros. Here, we set $M = N$,[6] so the input size of DQN is N^2. The output size is equal to the number of actions $N+1$. The number of neurons of two hidden layers is between the input layer size and output layer size, so they can extract compact features from the sparse input. For example, when $N = 16$, the input size is 256, the first hidden layer has 100 neurons, the second hidden layer has 50 neurons, and the output layer size is 17. The network is trained by mean square error loss with the Adam optimizer.

[6]Generally speaking, the larger M will lead to better reward performance since the RL agent can learn from more history information. However, this article focuses on sampling efficiency; investigating how M affects reward performance is beyond the scope of this article. Without loss of generality, we make an arbitrary choice of M in our experiment to demonstrate the improvement on sampling efficiency.

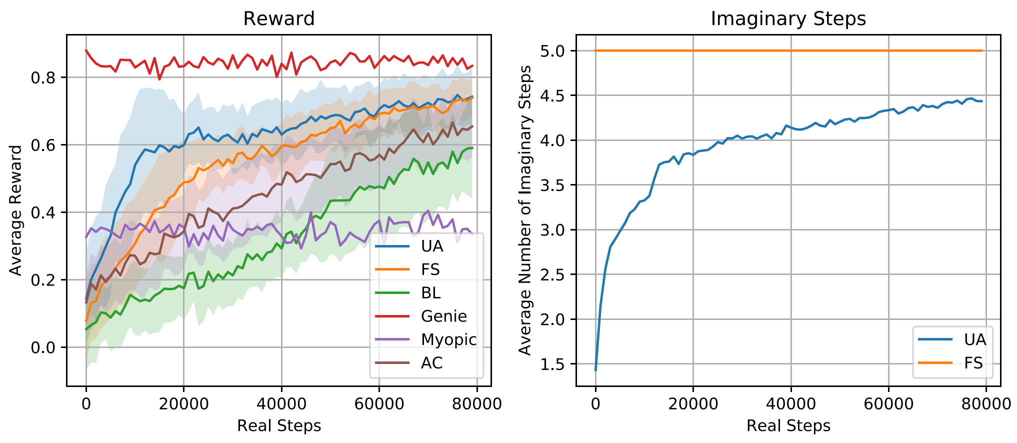

Fig. 5. Performance of genie added (Genie), model-free DQN (BL), Dyna (FS), AC, myopic, and introduced UA algorithms.

The model network is a double-hidden-layer MLP with tanh as the activation function. The input size is N^2. The output layer size is N, and we choose sigmoid as the activation function, because the output is the probability of the channel being *Inactive* thus should be a real number between 0 and 1. When $N = 16$, the input size is 256, the first and second hidden layers have 100 and 50 neurons, and the output layer size is 16. The network training loss is mean-square error and it is optimized by the Adam optimizer.

B. Experiment With Markov Traffic Pattern

In the following experiments, we set the number of channels $N = 16$, and each channel is assigned to a PU, the traffic pattern of each PU is independent and follows a two-state Markov chain. The transition probability of the Markov chain on the nth channel is

$$P_n = \begin{bmatrix} p_{00}^n & p_{01}^n \\ p_{10}^n & p_{11}^n \end{bmatrix} \tag{21}$$

where p_{ij} represents the probability of changing from current state i to next state j, $i, j \in \{0(Active), 1(Inactive)\}$. The transition probability P_n is randomly and independently generated from a uniform distribution over [0, 1].

1) Single SU (Perfect Sensing): In this experiment, one SU tries to access one of the N channels and perfect sensing is assumed.

Fig. 5 column 1 shows the average reward[7] of the introduced UA algorithm with $\eta = 0.02$ and the maximum number of imaginary step $n = 5$, Dyna (Algorithm 1) with fixed $n = 5$ imaginary steps (FS), the model-free DQN BL, the actor–critic (AC) algorithm,[8] as well as a belief state-based algorithm introduced in [23], because this algorithm requires state transition probabilities and sensing error probabilities as prior knowledge. We call it genie-added algorithm (Genie) and treat its performance as an upper bound. The performance of the myopic algorithm introduced in [27] is also plotted. This algorithm does not need the full information of transition probabilities, instead, it only requires the order of p_{11}^n and p_{01}^n. Myopic is a simple robust round-robin-based algorithm, however, it is optimal only when all channels follow the same two-state Markov transition probability and $p_{11}^n \geq p_{01}^n, \forall n$. Solid lines in the figure are the mean values over ten simulation runs with randomly initialized neural network weights, and shaded areas show the standard deviation. It can be seen that at 80 000 step the BL reaches a reward value of 0.6. To reach the same reward value, AC needs 60 000 steps, FS needs around 45 000 steps, while UA only needs 15 000 steps. The introduced algorithm achieves 5×, 4×, and 3× sampling efficiency over BL, AC, and FS. Another observation is that the performance gap between UA and FS is prominent at the initial RL stage. The reason is the model is overfitted at the beginning due to the limited amount of training samples. FS uses all imaginary samples generated by the model to train the RL agent, while UA only selects good imaginary samples. At later RL stages, when the model becomes more accurate, the performance gap between UA and FS decreases. Also, both UA and FS can achieve close-to-Genie reward performance. Because the Markov transition probabilities are randomly generated, which does not meet the optimal condition of the myopic algorithm, it only achieves around 0.35 average reward.

The average number of imaginary steps selected by UA is plotted in Fig. 5 column 2, the fixed number of five imaginary steps taken by FS is also plotted for comparison. We can see the number of imaginary samples selected by UA is much smaller than that of FS, especially at the initial RL stage, when the model is not well trained. In addition to high sampling efficiency, this result indicates another advantage of the UA algorithm—the low computation complexity. UA trains the RL agent with much fewer imaginary samples than FS. Although UA needs one more forward pass

[7]For better presenting the simulation result, each point on a curve is the averaged value over 1000 real steps, this setting is consistent among all figures in this article.

[8]The AC algorithm follows [20, Ch. 13.5]. It is implemented by two neural networks. The critic network learns the state value function $V(s)$, which is used to facilitate the actor network to learn the policy $\pi(a|s)$. For a fair comparison, both networks have the same NN architecture and input as the DQN BL described in Section VI-A, except for the output layer of critic network, which only needs one neuron to predict state value $V(s)$, and the actor network uses the Softmax activation function on the output layer to produce the probability of taking each action $\pi(a|s)$.

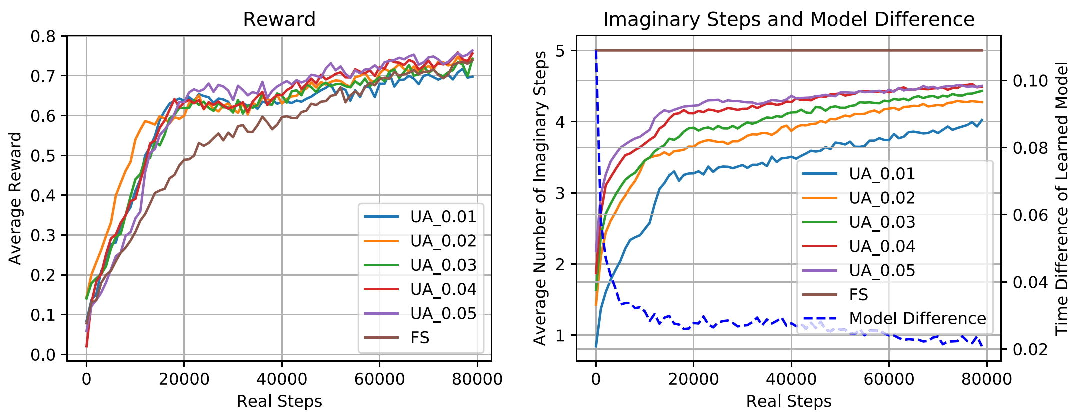

Fig. 6. UA algorithm with different threshold η.

to inference $G(t-1)$, the amount of computation is negligible compared with the unnecessary backward propagation computations needed for FS.

The threshold η is an important parameter in the UA algorithm, because it decides whether an imaginary sample generated by the model network will be used for RL training. Intuitively, if the threshold is too relaxed, an inaccurate sample will be adopted to train the RL agent, which confuses the RL agent and degrades the performance. On the other hand, if the threshold is too strict, many good samples will be filtered out instead of being adopted for RL training, which slows down the training process. When the model network training converges (the environment is well learned), the value of $\delta(G(t-1), G(t))$ should be a good candidate for the threshold η. In order to find this value, we conduct model network training alone (actions are chosen randomly, so the RL agent is not needed) under random environment realizations and plot average $\delta(G(t-1), G(t))$ in Fig. 6 column 2 (dashed line on the right y-axis). It can be seen that the convergence value is 0.02. So we believe potential candidates for η should be around this value, therefore we select the value from 0.01 to 0.05 with step size 0.01 for this experiment. The experiment reward results are shown in Fig. 6 column 1. The average number of imaginary steps selected by UA with different threshold η can be found in Fig. 6 column 2. The result of the FS algorithm with fixed imaginary steps is also shown for comparison. Each curve shows the average value of ten simulation runs. Clearly, the UA algorithm with any of the five η values outperforms the FS algorithm. The more interesting observation comes from the comparison among UA algorithms with different η values. From Fig. 6 column 1, we can see that the major differences are within the initial stage of RL (before 20 000 steps), and $\eta = 0.02$ has the highest convergence speed. This result verifies our intuition that too strict threshold (e.g., $\eta = 0.01$) filters out too many good imaginary samples, while too relaxed threshold (e.g., $\eta = 0.05$) adopts inaccurate imaginary samples to train the RL agent. Both situations degrade the RL performance.

Imperfect Sensing: In this experiment, sensing error is considered. To be specific, the false alarm probability is set to be $p_f = 0.1$. Because the channel access result does not depend on detection probability p_d, as shown by (4) and (5), it can be set to any value between 0 and 1 in the experiment. The simulation result is shown in Fig. 7 row 1 column 1. We can see with sensing error that UA can still achieve reward performance close to the genie-aided algorithm, while having better sampling efficiency than FS, AC, and BL. On the other hand, the final reward achieved by all algorithms is lower than the perfect sensing case in Fig. 5, which is reasonable due to sensing error.

2) Multiple SUs: As discussed in Section V, we apply the UA algorithm to the fully distributed MARL architecture[9] and show its performance under both perfect and imperfect sensing scenarios. In this experiment, multiple SUs present in the network, therefore the reward plotted is the total reward of all SUs.

The perfect sensing case with three SUs is depicted in Fig. 7 row 1 column 2, while the case with the false alarm probability $p_f = 0.1$ is depict in row 1 column 3. We can see in multiple-SU scenarios that UA still outperforms FS, AC, and BL in terms of sampling efficiency and achieves close to Genie reward performance. Note the Genie here is a centralized algorithm, it combines observations of all SUs as its input and outputs nonoverlapping actions to make sure SUs do not interfere with each other, while the UA is a fully distributed algorithm with no message exchange among SUs. The myopic algorithm still cannot achieve good performance, due to the same reason as the single SU case.

For investigating the effects of more SUs present in the network, the results of 4, 5, and 6 SUs are shown in row 2 of Fig. 7. It can be seen that as the number of SU increases, the incremental reward achieved by the optimal algorithm (Genie) is decreasing, meaning the reward is saturating. This is reasonable because as the number of SU increases, the maximum opportunistic capacity (spectrum white space) of the system is about to be fully utilized. This maximum capacity is determined by the number of channels and the underlying traffic pattern of PUs in the system. Once the maximum capacity is reached, further increasing the SU number will not improve the reward. Another observation is that the learning problem becomes harder when the number of SUs increases because

[9]Note the training is distributed, so each SU trains its own DQN and model network weights independently without parameter sharing. Furthermore, the number of SUs should be consistent at the training and deployment stage.

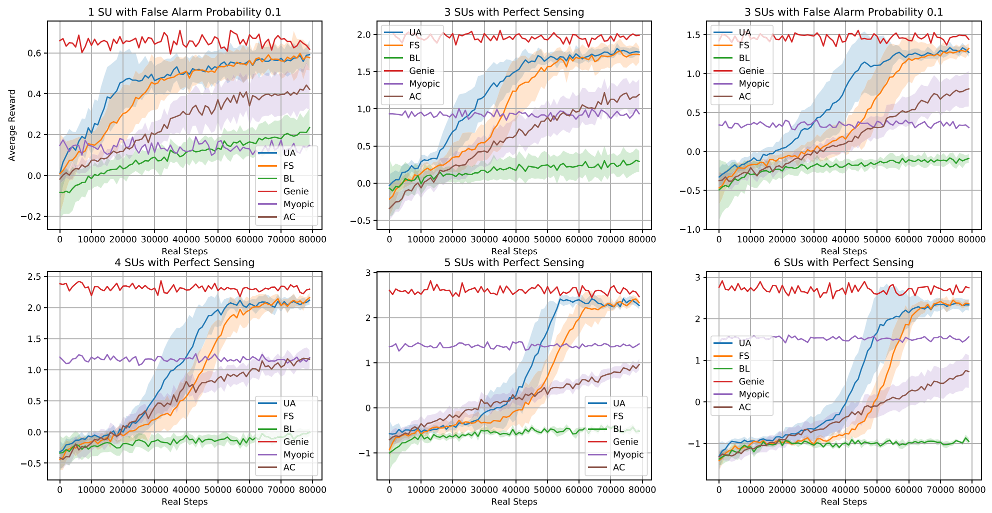

Fig. 7. Performance with sensing error and multiple SUs.

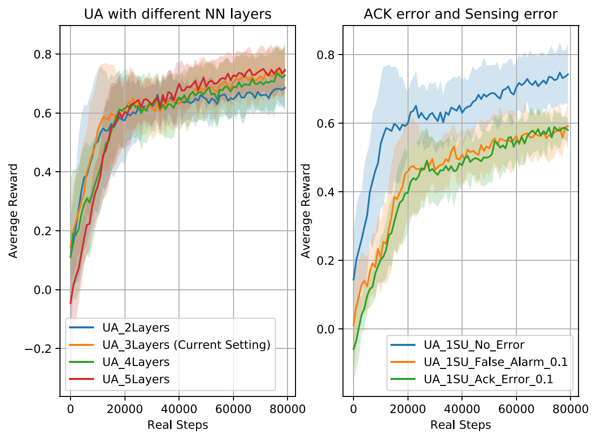

Fig. 8. Different NN structure and ACK error.

more SUs are interfering with each other, therefore, it takes a longer time for SU agents to converge to a good policy. Nevertheless, UA still has the highest sampling efficiency and achieves close to optimal reward performance.

3) Different Neural Network Structure: In the previous experiments, both the DQN and model networks are three-layer MLP (two hidden layers plus one output layer). It is beneficial to study how network structure change affects UA performance. In this experiment, we set the layer number of both neural networks to two, four, and five, and compare the performance with the current setting (three layers). More specifically, two-layer MLP has one hidden layer of 100 neurons. One more hidden layer of 50 neurons is added for three-layer MLP. Similarly, two and three more hidden layers of 50 neurons are added for four-layer and five-layer MLP, respectively. The result is depicted in Fig. 8 column 1. It can be seen except for the two-layer MLP which has slightly lower final reward performance, and the rest structures have almost the same final reward performance. This indicates that as long as the DQN and model networks are deep enough to learn the Q value and the environment, adding more layers will not improve the performance. The drawback of a deeper network is longer training time, which is also reflected in the figure: the deeper network structure has a slower start on the reward curve.

4) Acknowledgment Error: Previously, it was assumed the channel access result can be identified by the SU receiver perfectly. However, it is possible that the SU receiver is unable to decode ID information due to the transmission error and then sends back the wrong result to the SU transmitter, therefore it is important to study the effect of acknowledgment error. When error occurs, the SU receiver mistakes a succeed access as a failed one,[10] i.e., channel is idle $z_t^n = 1$, but access failed $k_t^n = 0$. Recall when there is a false alarm sensing error, the channel is available $z_t^n = 1$, but the access result is fail $k_t^n = 0$. So the acknowledgment error has the same effect as the false alarm sensing error. To verify this point, the performance of one SU with 10% false alarm probability and one SU with 10% acknowledgment error probability is plotted in Fig. 8 column 2. It can be seen they have almost the same performance curve. Comparing with the one SU no error result, the acknowledgment error degrades the performance.

VII. Conclusion

In this article, we demonstrated that the introduced lightweight UA algorithm outperforms the standard deep Q learning algorithm as well as the Dyna algorithm in terms of sampling efficiency, while achieving reward performance close to the genie-aided algorithm. This new algorithm significantly extends the application scenarios of RL to real-time applications and low-cost devices. For subsequent studies, we will incorporate a multistep target value update [9], [10] in the UA algorithm to further improve the sampling efficiency of MFRL. As AC also belongs to MFRL, another future direction is applying our algorithm to accelerate it.

[10]We believe it is highly unlikely the case $z_t^n = 0, k_t^n = 1$ can happen. Because it requires the SU receiver to incorrectly decode its unique ID from a message that is not sent to it.

Appendix A
Proof of (17)

The current state is s. The current action is to access channel n. If channel access succeed, we have reward $+1$, otherwise, reward is -1, so the average reward error is

$$\begin{aligned}\left|E[r]-E[\hat{r}]\right| &= \left|p^1_{n|s}\cdot 1+p^0_{n|s}\cdot(-1)\right. \\ &\quad \left.- g^1_{n|s}\cdot 1-g^0_{n|s}\cdot(-1)\right| \\ &\le \left|p^1_{n|s}-g^1_{n|s}\right|+\left|g^0_{n|s}-p^0_{n|s}\right| \\ &= \left\|\boldsymbol{P_{n|s}}-\boldsymbol{G_{n|s}}\right\|_1 \\ &= 2\delta(\boldsymbol{P_n},\boldsymbol{G_n}|s).\end{aligned}$$

Appendix B
Proof of (18)

Given the current state s, the RL agent suggests SU to access channel n, then, there will be only two possible observations: access channel n is *Succeed* or *Fail*. Because the input state of the RL agent is defined as the stack of history observations, there will be two possible states at the next time step s'_0 and s'_1 corresponding to the two observations *Fail* and *Succeed*.

$E[V(s')]$ is the actual average value function at next time step. It should be averaged over two possible states $s'\in\{s'_0,s'_1\}$. The probability to be in states s'_0 and s'_1, as defined in Section IV-B, is $p^0_{n|s}$ and $p^1_{n|s}$.

For the learned model, note our policy is deterministic (given the current state, the agent suggests the same action regardless it is in the real environment or the simulated environment), so the two possible next states predicted by the model $\{\hat{s}'_0,\hat{s}'_1\}$ will be exactly the same as the two states possibly seen in the real environment $\{s'_0,s'_1\}$, i.e., $\{\hat{s}'_0,\hat{s}'_1\}=\{s'_0,s'_1\}$. The difference is the probability to be in those states; the probability to be in states $\hat{s}'_0$ and $\hat{s}'_1$ predicted by the model is $g^0_{n|s}$ and $g^1_{n|s}$. Recall as defined earlier that the probability to be in states s'_0 and s'_1 in real environment is $p^0_{n|s}$ and $p^1_{n|s}$.

With the above notations, we have

$$\begin{aligned}E\big[V(s')\big] &= \sum_{i=0}^{1}p^i_{n|s}V\big(s'_i\big) \\ E\big[V(\hat{s}')\big] &= \sum_{i=0}^{1}g^i_{n|s}V\big(\hat{s}'_i\big)=\sum_{i=0}^{1}g^i_{n|s}V\big(s'_i\big) \\ \big|E\big[V(s')\big]-E\big[V(\hat{s}')\big]\big| &= \left|\sum_{i=0}^{1}\Big(p^i_{n|s}V\big(s'_i\big)-g^i_{n|s}V\big(s'_i\big)\Big)\right| \\ &\le \sum_{i=0}^{1}\Big|p^i_{n|s}V\big(s'_i\big)-g^i_{n|s}V\big(s'_i\big)\Big| \\ &\le \sum_{i=0}^{1}\Big|p^i_{n|s}-g^i_{n|s}\Big|\max_{s'}|V(s')| \\ &= 2\delta(\boldsymbol{P_n},\boldsymbol{G_n}|s)\max_{s'}|V(s')|.\end{aligned}$$

Appendix C
Proof of (20)

Assumptions and inequality needed for the proof are as follows.

1) The learned model converges to the real distribution
$$\|G(\infty)-P\|_1\le\epsilon. \tag{22}$$
2) Minkowski inequality
$$\|x+y\|_1\le\|x\|_1+\|y\|_1. \tag{23}$$
3) Assume $f(t)=\delta(G(t-1),G(t))$ is a decreasing function of t, and
$$f(t+1)=\lambda f(t),\ \ 0<\lambda<1 \tag{24}$$
this assumption is based on the convergence property of the gradient descent method and two approximations. It is also verified by our experiment result shown in Fig. 6 column 2. See Appendix D for details.

Proof:

$$\begin{aligned}2\delta(G(t),P) &= \|G(t)-P\|_1 \\ &= \|G(t)-G(t+1)+G(t+1)-G(t+2)+\cdots \\ &\quad +G(\infty)-P\|_1 \\ &\le \sum_{\tau=t}^{\infty}\|G(\tau)-G(\tau+1)\|_1+\epsilon \qquad (25) \\ &\le \sum_{\tau=t-1}^{\infty}\|G(\tau)-G(\tau+1)\|_1+\epsilon \\ &= \sum_{\tau=t-1}^{\infty}\lambda^{\tau-t+1}\|G(t-1)-G(t)\|_1+\epsilon \\ &= \frac{1}{1-\lambda}\|G(t-1)-G(t)\|_1+\epsilon \\ &= \frac{2}{1-\lambda}\delta(G(t-1),G(t))+\epsilon \qquad (26)\end{aligned}$$

where (25) is derived by applying (22) and (23), and (26) is derived by applying (24). ■

Appendix D
Assumption (24)

From [28, Ch. 9.3], we know that the convergence of the gradient descent method follows:

$$h(x(t))-h^*\le\lambda^t\big(h(x(0))-h^*\big) \tag{27}$$

where $h(x(t))$ is the function value at iteration t, h^* is the optimal function value, and $\lambda\in(0,1)$ is a constant, meaning that the function value reaches the optimal value at an exponential rate. In our case, the model network learns the real distribution P by the gradient descent method, so $h(x(t))\triangleq\delta(G(t),P)$, $h^*=0$, and (27) can be expressed as

$$\delta(G(t),P)\le\lambda^t\delta(G(0),P).$$

Here, we make an approximation that

$$\delta(G(t),P)\approx\lambda^t\delta(G(0),P). \tag{28}$$

For $\delta(G(t-1), G(t))$, we know that

$$\begin{aligned}\delta(G(t-1), G(t)) &= \frac{1}{2}\|G(t-1) - G(t)\|_1 \\ &= \frac{1}{2}\|G(t-1) - P + P - G(t)\|_1 \\ &\geq \frac{1}{2}\|G(t-1) - P\|_1 - \frac{1}{2}\|G(t) - P\|_1 \\ &= \delta(G(t-1), P) - \delta(G(t), P).\end{aligned}$$

Here, we make another approximation

$$\delta(G(t-1), G(t)) \approx \delta(G(t-1), P) - \delta(G(t), P). \quad (29)$$

Combining (28) and (29), we have

$$\delta(G(t-1), G(t)) \approx \lambda^{t-1}(1-\lambda)\delta(G(0), P). \quad (30)$$

Accordingly

$$\frac{\delta(G(t), G(t+1))}{\delta(G(t-1), G(t))} \approx \frac{\lambda^{t}(1-\lambda)\delta(G(0), P)}{\lambda^{t-1}(1-\lambda)\delta(G(0), P)} = \lambda \quad (31)$$

therefore, we have assumption (24), which is also verified by the experimental result shown in Fig. 6 column 2; the dashed line represents $\delta(G(t-1), G(t))$, which decreases exponentially.

REFERENCES

[1] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1238–1274, 2013.

[2] M. P. Deisenroth, G. Neumann, J. Peters, "A survey on policy search for robotics," *Found. Trends Robot.*, vol. 2, nos. 1–2, pp. 1–142, 2013.

[3] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[4] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Vilamoura, Portugal, 2012, pp. 4906–4913.

[5] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM SIGART Bull.*, vol. 2, no. 4, pp. 160–163, 1991.

[6] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Brisbane, QLD, Australia, 2018, pp. 7559–7566.

[7] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-learning with model-based acceleration," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 2829–2838.

[8] S. Racanière *et al.*, "Imagination-augmented agents for deep reinforcement learning," in *Proc. 31st Int. Conf. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5694–5705.

[9] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine, "Model-based value estimation for efficient model-free reinforcement learning," 2018. [Online]. Available: arXiv:1803.00101.

[10] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, "Sample-efficient reinforcement learning with stochastic ensemble value expansion," in *Proc. 32nd Int. Conf. Adv. Neural Inf. Process. Syst.*, 2018, pp. 8234–8244.

[11] A. Osseiran *et al.*, "Scenarios for 5G mobile and wireless communications: The vision of the METIS project," *IEEE Commun. Mag.*, vol. 52, no. 5, pp. 26–35, May 2014.

[12] S. Marek, *Marek's Take: Dynamic Spectrum Sharing May Change the 5G Deployment Game*, Fierce Wireless, Framingham, MA, USA, Apr. 2019.

[13] Q. Zhao and B. M. Sadler, "A survey of dynamic spectrum access," *IEEE Signal Process. Mag.*, vol. 24, no. 3, pp. 79–89, May 2007.

[14] H. Chang, H. Song, Y. Yi, J. Zhang, H. He, and L. Liu, "Distributive dynamic spectrum access through deep reinforcement learning: A reservoir computing-based approach," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1938–1948, Apr. 2019.

[15] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, "Deep reinforcement learning for dynamic multichannel access in wireless networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 4, no. 2, pp. 257–265, Jun. 2018.

[16] N. Morozs, D. Grace, and T. Clarke, "Distributed Q-learning based dynamic spectrum access in high capacity density cognitive cellular systems using secondary LTE spectrum sharing," in *Proc. Int. Symp. Wireless Pers. Multimedia Commun. (WPMC)*, Sydney, NSW, Australia, 2014, pp. 462–467.

[17] O. Naparstek and K. Cohen, "Deep multi-user reinforcement learning for dynamic spectrum access in multichannel wireless networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Singapore, 2017, pp. 1–7.

[18] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[19] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[21] D. P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1987.

[22] "Broadband radio access networks (BRAN); 5 GHz high performance RLAN; harmonized EN covering the essential requirements of article 3.2 of the R&TTE directive, v1.7.2" ETSI, Sophia Antipolis, France, Rep. EN.301.893, 2014.

[23] Q. Zhao, L. Tong, A. Swami, and Y. Chen, "Decentralized cognitive MAC for opportunistic spectrum access in ad hoc networks: A POMDP framework," Dept. Elect. Comput. Eng., Univ. California, Davis, CA, USA, Rep. CA 95616, 2007.

[24] O. Naparstek and K. Cohen, "Deep multi-user reinforcement learning for distributed dynamic spectrum access," *IEEE Trans. Wireless Commun.*, vol. 18, no. 1, pp. 310–323, Jan. 2019.

[25] L. Liang, H. Ye, and G. Y. Li, "Spectrum sharing in vehicular networks based on multi-agent reinforcement learning," 2019. [Online]. Available: arXiv:1905.02910.

[26] P. Venkatraman and B. Hamdaoui, "Cooperative Q-learning for multiple secondary users in dynamic spectrum access," in *Proc. 7th Int. Wireless Commun. Mobile Comput. Conf.*, Istanbul, Turkey, 2011, pp. 238–242.

[27] Q. Zhao, B. Krishnamachari, and K. Liu, "On myopic sensing for multichannel opportunistic access: Structure, optimality, and performance," *IEEE Trans. Wireless Commun.*, vol. 7, no. 12, pp. 5431–5440, Dec. 2008.

[28] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

Lianjun Li received the B.S. degree in telecommunications engineering from Zhejiang University, Hangzhou, China, and the M.S. degree in electrical engineering from the University of Texas at Dallas, Richardson, TX, USA. He is currently pursuing the Ph.D. degree with the Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA, USA.

He joined Ericsson, Beijing, China, as a wireless network optimization engineer seven years ago. His research interest is applying reinforcement learning and deep learning techniques to wireless communications.

Lingjia Liu (Senior Member, IEEE) received the B.S. degree in electronic engineering from Shanghai Jiao Tong University, Shanghai, China, and the Ph.D. degree in electrical and computer engineering from Texas A&M University, College Station, TX, USA.

He was an Associate Professor with the EECS Department, University of Kansas (KU), Lawrence, KS, USA. He is currently an Associate Professor with the ECE Department, Virginia Tech (VT), Blacksburg, VA, USA, and is also the Associate Director of Wireless@VT. He spent more than four years working in the Mitsubishi Electric Research Laboratory and the Standards and Mobility Innovation Lab, Samsung Research America, Plano, TX, USA, where he received the Global Samsung Best Paper Award in 2008 and 2010. He was leading Samsung's efforts on multiuser MIMO, CoMP, and HetNets in LTE/LTE-Advanced standards. His general research interests mainly lie in emerging technologies for Beyond 5G cellular networks, including machine learning for wireless networks, massive MIMO, massive MTC communications, and mmWave communications.

Dr. Liu received the Air Force Summer Faculty Fellow from 2013 to 2017, the Miller Scholar at KU in 2014, the Miller Professional Development Award for Distinguished Research at KU in 2015, the 2016 IEEE GLOBECOM Best Paper Award, the 2018 IEEE ISQED Best Paper Award, the 2018 IEEE TAOS Best Paper Award, and the 2018 IEEE TCGCC Best Conference Paper Award.

Jianan Bai received the B.S. degree in communications engineering from Yingcai Honors College, University of Electronic Science and Technology of China, Chengdu, China, in 2018. He is currently pursuing the Ph.D. degree with the Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA, USA.

His research interest is applying tools from optimization, stochastic geometry, and machine learning to solve emerging problems in device-to-device, Internet of Things, and machine-type communications networks.

Hao-Hsuan Chang (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering and the M.S. degree in communication engineering from National Taiwan University, Taipei, Taiwan, in 2014 and 2015, respectively. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Virginia Tech, Blacksburg, VA, USA.

From 2016 to 2017, he worked as a Research Assistant on super resolution with the Computer Vision Laboratory, Academia Sinica, Taipei, Taiwan. His research interests include dynamic spectrum access, deep learning, and optimization for wireless communications.

Hao Chen received the B.S. and M.S. degrees in information engineering from Xi'an Jiaotong University, Xi'an, China, in 2010 and 2013, respectively, and the Ph.D. degree in electrical engineering from the University of Kansas, Lawrence, KS, USA, in 2017.

Since 2016, he has been a Research Engineer with the Standards and Mobility Innovation Laboratory, Samsung Research America, Plano, TX, USA. His research interests include dynamic spectrum access, network optimization, machine learning, and 5G cellular systems.

Jonathan D. Ashdown (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Rensselaer Polytechnic Institute, Troy, NY, USA, in 2006, 2008, and 2012, respectively. His doctoral dissertation was on a high-rate ultrasonic through-wall communication system using MIMO-OFDM in conjunction with interference mitigation techniques.

From 2012 to 2015, he worked as a Civilian Research Scientist with the Department of Defense (DoD), Naval Information Warfare Center, Charleston, SC, USA, where he was involved in several basic and applied research projects for the U.S. Navy. In 2015, he transferred within DoD to the Air Force Research Laboratory, Rome, NY, USA, where he works on several advanced emerging technologies for the U.S. Air Force.

Dr. Ashdown was a recipient of the Best Unclassified Paper Award at the IEEE Military Communications Conference in 2012, the 2016 IEEE GLOBECOM Best Paper Award, the 2018 IEEE TAOS Best Paper Award, and the 2018 IEEE TCGCC Best Paper Award.

Jianzhong (Charlie) Zhang (Fellow, IEEE) received the Ph.D. degree from the University of Wisconsin, Madison, WI, USA.

From 2009 to 2013, he served as the Vice Chairman for the 3GPP RAN1 Working Group and led the development of LTE and LTE-Advanced technologies, such as 3-D channel modeling, UL-MIMO, CoMP, and carrier aggregation for TD-LTE. He is an SVP and the Head of the Standards and Mobility Innovation Laboratory, Samsung Research America, Plano, TX, USA, where he leads research, prototyping, and standards for 5G and future multimedia networks.

Yang Yi (Senior Member, IEEE) received the B.S. and M.S. degrees in electronic engineering from Shanghai Jiao Tong University, Shanghai, China, and the Ph.D. degree in electrical and computer engineering from Texas A&M University, College Station, TX, USA.

She is an Associate Professor with the Bradley Department of ECE, Virginia Tech, Blacksburg, VA, USA. Her research interests include very large-scale integrated circuits and systems, computer-aided design, and neuromorphic computing.