Deep Recurrent Q Networks for Dynamic Spectrum Access in Dynamic Heterogeneous Environments with Partial Observations

Yue Xu

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Electrical Engineering

Buehrer, R. Michael, Chair
Dhillon, Harpreet Singh
Headley, William C
Liu, Lingjia
Wang, Yue J

Aug 11, 2022
Blacksburg, Virginia

Keywords: Dynamic Spectrum Access, Partial Knowledge, Deep Recurrent Neural Networks, Parallel Learning, Transfer Learning, Meta-Learning, Sensing Prediction, Imperfect System Feedback, Multi-Rate and Multi-Agent, Dynamic environments, Cache

Deep Recurrent Q Networks for Dynamic Spectrum Access in Dynamic Heterogeneous Environments with Partial Observations

Yue Xu

(ABSTRACT)

Dynamic Spectrum Access (DSA) has strong potential to address the need for improved spectrum efficiency. Unfortunately, traditional DSA approaches such as simple "sense-and-avoid" fail to provide sufficient performance in many scenarios. Thus, the combination of sensing with deep reinforcement learning (DRL) has been shown to be a promising alternative to previously proposed simplistic approaches. DRL does not require the explicit estimation of transition probability matrices and prohibitively large matrix computations as compared to traditional reinforcement learning methods. Further, since many learning approaches cannot solve the resulting online Partially-Observable Markov Decision Process (POMDP), Deep Recurrent Q-Networks (DRQN) have been proposed to determine the optimal channel access policy via online learning. The fundamental goal of this dissertation is to develop DRL-based solutions to address this POMDP-DSA problem. We mainly consider three aspects in this work: (1) optimal transmission strategies, (2) combined intelligent sensing and transmission strategies, and (c) learning efficiency or online convergence speed. Four key challenges in this problem are (1) the proposed DRQN-based node does not know the other nodes' behavior patterns a priori and must to predict the future channel state based on previous observations; (2) the impact to primary user throughput during learning and even after learning must be limited; (3) resources can be wasted during sensing/observation and want to limit this waste; and (4) convergence speed must be improved without impacting performance. We demonstrate in this dissertation, that the proposed DRQN-based agent

can learn: (1) the optimal transmission strategy in a variety of environments with partial observations; (2) a sensing strategy that provides near-optimal throughput in different environments while dramatically reducing the needed sensing resources; (3) a strategy that is robust to imperfect observations; (4) a sufficiently flexible approach that can accommodate dynamic environments, multi-channel transmission and the presence of multiple agents; (5) in an accelerated fashion utilizing one of three different approaches.

Deep Recurrent Q Networks for Dynamic Spectrum Access in Dynamic Heterogeneous Environments with Partial Observations

Yue Xu

(GENERAL AUDIENCE ABSTRACT)

With the development of wireless communication, such as 5G, global mobile data traffic has experienced tremendous growth, which makes spectrum resources even more critical for future networks. However, the spectrum is an exorbitant and scarce resource. Dynamic Spectrum Access (DSA) has strong potential to address the need for improved spectrum efficiency. Unfortunately, traditional DSA approaches such as simple "sense-and-avoid" fail to provide sufficient performance in many scenarios. Thus, the combination of sensing with deep reinforcement learning (DRL) has been shown to be a promising alternative to previously proposed simplistic approaches. Compared with traditional reinforcement learning methods, DRL does not require explicit estimation of transition probability matrices and extensive matrix computations. Furthermore, since many learning methods cannot solve the resulting online Partially Observable Markov Decision Process (POMDP), a deep recurrent Q-network (DRQN) is proposed to determine the optimal channel access policy through online learning. The basic goal of this paper is to develop a DRL-based solution to this POMDP-DSA problem. This paper mainly focuses on improving performance from three directions. 1. Find the optimal (or sub-optimal) channel access strategy based on fixed partial observation mode; 2. Based on work 1, propose a more intelligent way to dynamically and efficiently find more reasonable (higher efficiency) sensing/observation policy and corresponding channel access strategy; 3. On the premise of ensuring performance, use different machine learning algorithms or structures to improve learning efficiency and avoid

users waiting too long for expected performance. Through the research in these three main directions, we have found an efficient and diverse solution, namely DRQN-based technology.

Dedication

To all who I love, and who loves me

Acknowledgments

My deepest gratitude goes first and foremost to Dr. Mike Buehrer, my advisor, for his constant encouragement and guidance. He has walked me through all the stages of my Ph.D. study. Without his consistent and illuminating instruction, time management talent, and immense knowledge and experience, my Ph.D. study and life could not have reached their present form. I appreciate the enormous amount of time and effort, ideas, and funding you put into earning my Ph.D. journey. I also respect his charming teaching personality, being a father of six, and contributing to the local community. Meanwhile, I would like to thank the members of my Ph.D. advisory committee, Dr. Dhillon, Harpreet Singh, Dr. Headley, William C, Dr. Lingjia Liu, and Dr. Wang, Yue J for their valuable comments which have helped me to substantially improve the quality of this dissertation. Second, I would like to express my heartfelt gratitude to my parents. Without my parents, who gave me life, nurtured, supported, and cared about me unconditionally, I would not have had the opportunity to complete my studies here. Meanwhile, I am grateful for all the professors who have taught me. Without their careful teaching and strict requirements, I would not have been able to acquire a solid foundation in wireless communication, machine learning, and mathematics, which are my pursuit of a Ph.D. necessary conditions. I would like to thank my lab colleagues, Don, Charlie, Will, Gaurav, Chris, Mark, Daniel, Yifei, Hao-Hsuan, Hao Song, Ye Hu, and many others in our office for their help and support at Wireless@VT. Special thanks to Jet for offering me a lot of help in research and life. Additional gratitude for my amazing friends at Blacksburg: Jinyang Li, Shengyuan Niu, Zhengdao Jiao, Liurui Li, Pang Zi and Jing Zhao. Without their endless encouragement, support, unconditional love and company, my educational journey would have been eclipsed. Last but not least,

words cannot express how grateful I am to Miss Visa, my beloved and soul mate—with you, looking ahead and working together.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Aificial Intelligence

ANN Artificial Neural Network

CDF Cumulative Distribution Function

DNN/ RNN/ CNN Deep/ Recurrent/ Convolutional Neural Networks

DQN Deep Q Networks

DRL Deep Reinforcement Learning

DRQN Deep Recurrent Q Networks

DSA Dynamic spectrum access

i.i.d idependent and identically distributed

IoT Internet of Things

kNN k-Nearest Neighbor

LSTM Long Short-Term Memory

MAML Model-Agnostic Meta-Learning

ML Machine learning

MSE Mean Square Error

PDF Probability Density Function

RF Radio Frequency

SNR Signal-to-Noise Ratio

Chapter 1

Motivation, Background and Contributions

Over the past ten years, global mobile data traffic has experienced tremendous growth. Smartphone-centric mobile networks are gradually evolving into a massive Internet of Things(IoT) ecosystem that integrates everything from smartphones to drones, connected vehicles, wearable sensors, and virtual reality apparatuses[1, 2, 3]. This unprecedented shift will not only continue to drive exponential growth in wireless traffic, straining spectrum resources but will also lead to the emergence of new wireless service use cases that differ from traditional multimedia or voice-based services. Meanwhile, beyond the need for higher data rates, the next-generation wireless networks must have multiple performances required to achieve ultra-reliable, low-latency communication in real-time to a rich and dynamic environment [4, 5, 6, 7, 8, 9]. Further, the future wireless network system must also support cloud-based gaming, immersive virtual reality services, real-time HD streaming, and conventional multimedia services [10, 11, 12]. To cope with the ongoing and rapid evolution of wireless services, basic concepts from machine learning are being integrated into the wireless infrastructure and at the terminal.

Machine learning is expected to play several roles in the next generation of wireless communication systems. First, the most natural application of machine learning is to leverage big data to enhance situational awareness, and overall network operations [11, 13, 14]. Second,

machine learning will be the main driver for intelligent and data-driven wireless network optimization [15, 16]. It needs to be emphasized here that, compared to traditional distributed optimization techniques, which are usually done iteratively in an offline or semi-offline manner, the machine learning-guided resource management mechanism will be able to operate in a fully online manner by learning, in real-time, wireless environment state and network users[17]. Third, in addition to system-level functions, machine learning can play a key role in the physical layer of wireless networks.[18, 19]. Thus, we believe that machine learning has the potential to solve many of the challenges associated with next-generation wireless communications, particularly related to DSA, which we will discuss next.

1.1 Dynamic Spectrum Access

Dynamic spectrum access (DSA)is a highlight research topic and has been widely studied over the past decade [20, 21, 22]. DSA is a combination of research and development techniques based on theoretical concepts in network information theory and game theory to improve the spectral efficiency of multi-communication networks. More recently, the concept of DSA has drawn on principles from cross-layer optimization, artificial intelligence, machine learning, etc[23, 24]. It has been made more feasible by the availability of software radio and the development of sufficiently fast processors both at servers and terminals. Different kinds of software developments on wireless commutation and AI are techniques for cooperative optimization. And the related optimization techniques can also be compared or related to the optimization of one link in the network because of losing performance on many links negatively affected by this single optimization. DSA provides sensing and dynamic reconfiguration capabilities that leverage spectrum holes (also called white spaces). Exploiting these

white spaces allows opportunistic transmission without requiring extra spectrum bandwidth. When channels are independent and identically distributed (i.i.d.), there are many existing works, such as [25, 26, 27, 28], which examine the optimal/near-optimal policy for channel access. The Myopic policy [25] and the Whittle Index policy [26] are two classic and practical approaches. According to [25], the Myopic policy focuses on the immediate reward obtained from action and ignores its effects in the future. Thus, the user always tries to select a channel that will maximize the expected immediate reward. However, the Myopic policy is not optimal in general. When the channel state transitions are positively correlated or slightly negatively correlated, the approach is nearly optimal[25]. However, when the channel state transitions are negative, or the agent doesn't know the transitions, the myopic may fail. The Whittle Index is another algorithm that can not guarantee optimal performance. Based on [26], when the two-state Markov chain matrix is known and all channels are independent, the Whittle Index policy can be represented as a closed-form solution and has the same optimal result as the Myopic policy. Due to their similarity, we only implement the Myopic policy in the study presented in this dissertation.

Although the Myopic policy and the Whittle Index policy are not difficult to implement, the two algorithms need to know the system statistics, which are unknown *apriori*. Moreover, even if the statistical knowledge is obtained via online estimation, the accuracy of this knowledge cannot be guaranteed. Further, if there are some channels that a Markov transition matrix cannot fully model, even if its probability characteristics are estimated, the performance can suffer dramatically. Thus, new approaches are needed.

At this point, it is beneficial to explain why the Markov decision process (MDP) is assumed in our study. In mathematics, an MDP is a discrete-time stochastic control process. It provides a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision maker. The model is used in many disciplines, including communication, manufacturing, automatic control, and

economics. In wireless communication, especially in our DSA issue, future channel states (conditional on both past and present values) depend only upon the present state; given the present, the future does not depend on the past. And the process conforms to the Markov property, so this is an important reason why we introduce MDP - everything starts from practical problems. The basic idea of an MDP is that at each time step, the process is in some state s. The process at the next time step will randomly move into a new state $s\_$, which depends only on the previous state and not the history of states. Additionally, when a decision maker is involved and takes action a based on state s, the decision maker receives a corresponding reward r. Mathematically we represent the probability of transitioning from state s to state $s\_$ as

$$P_{ss\_} = P\left[S_{t+1} = s\_ \,|S_t = s\right] \tag{1.1}$$

while the expected reward received is written as

$$R_s = E\left[R_{t+1}\,|S_t = s, A_t = a\right] \tag{1.2}$$

1.2 Machine Learning

Machine learning (ML) was introduced in the late 1950's as a technique for artificial intelligence (AI) [29, 30]. It is the scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead. Over time, its focus evolved and shifted to computationally viable and robust algorithms. Over the past decade, machine learning techniques have been widely used for various tasks, including classification, regression, and density estimation in application domains such as bioinformatics, speech recognition, spam

detection, computer vision, fraud detection, and advertising networks.[31]. The algorithms and techniques come from diverse fields, including statistics, mathematics, neuroscience, and computer science. The following two classical definitions capture the essence of machine learning: 1) The development of computer models for learning processes that provide solutions to the problem of knowledge acquisition and enhance the performance of developed systems. 2) The adoption of computational methods for improving machine performance by detecting and describing consistencies and patterns in training data [32].

Existing machine learning algorithms: supervised, unsupervised, and reinforcement learning [33]. Machine learning algorithms are provided with a labeled training data set in the first category. This set is used to build the system model representing the learned relationship between the input, output, and system parameters. In contrast to supervised learning, unsupervised learning algorithms are not provided with labels (i.e., there is no output vector). An unsupervised learning algorithm aims to classify the sample sets into different groups (i.e., clusters) by investigating the similarity between the input samples. Thus, in unsupervised learning, the same input may have different outputs. The third category includes reinforcement learning algorithms, in which the agent learns by interacting with its environment (i.e., online learning). However, they can also be pre-trained by simulating the environment. Finally, some machine learning algorithms do not naturally fit into this classification since they share characteristics of both supervised and unsupervised learning methods. These hybrid algorithms (often termed semi-supervised learning) aim to inherit the strengths of these main categories while minimizing their weaknesses [34].

1.3 Deep Neural Networks

Before defining a Deep Neural Network, we must fist explain the Neural Network.

A Neural Network (a.k.a Artificial Neural Network, ANN) [35, 36, 37] is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It comprises many highly interconnected processing units (neurons) working in unison to solve specific problems. ANNs, like people, learn by experience. ANNs are configured for specific applications, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves tuning the synaptic connections between neurons. The development of artificial neural networks has been key to teaching computers to think and understand the world while retaining their inherent advantages over us, such as speed, accuracy, and unbiasedness. A trained neural network can be thought of as an 'expert' in the category of information it has been given to analyze. This expert can then be used to provide projections showing new situations of interest and answer 'what if' questions [38, 39].

Deep neural networks are a set of algorithms that have set new standards in accuracy for many significant problems, such as image recognition, sound recognition, recommender systems, natural language processing, etc. For example, deep learning is a part of DeepMind's well-known AlphaGo algorithm, which beat the former world champion Lee Sedol at Go in early 2016, and the current world champion Ke Jie in early 2017. A complete explanation of the neural works follows. First, depth is a technical term. It refers to the number of layers in a neural network. Shallow nets have a so-called *hidden layer*, while deep nets have more than one. Multiple hidden layers allow deep neural networks to learn data features in so-called feature hierarchies. Simple features are recombined from one layer to the next to form more complex features. A network with multiple layers passes the input data (attributes) through

more mathematical operations and thus is more computationally intensive to train than a network with few layers. Computational intensity is one of the hallmarks of deep learning, which is why a new type of chip called a GPU (general-purpose processing unit) is needed to train deep learning models.

1.4 Deep Recurrent Q-Learning

1.4.1 Deep Q Learning

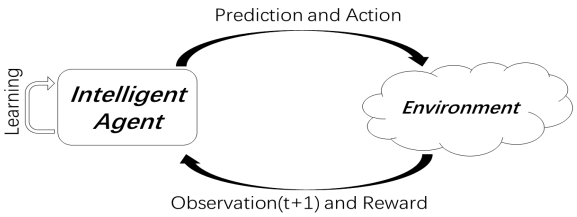


Figure 1.1: The agent-environment interaction process

Q-learning is one of the model-free reinforcement learning algorithms. Q-learning aims to learn a policy that tells an agent what action to take under what circumstances. It does not require a model (hence the connotation "model-free") of the environment and can handle problems with stochastic transitions and rewards without requiring adaptations. As shown in Figure 1.1, an intelligent agent interacts with the environment over a sequence of discrete times to accomplish a task. At the time t, the agent observes the environment to obtain $o_t \in O$, where O is the set of possible observations. It then takes an action, $a_t \in A_{o_t}$ where A_{o_t} is the set of possible actions given observation o_t. As a result of the observation-action pair, (o_t, a_t), the agent receives a reward $R(o_t, a_t)$ (in order to simplify the representation, usually we also use r_t), and the environment provides a new observation o_{t+1} at time $t+1$.

The goal of the agent is to maximize some function of the reward. 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 reward). The Q-learning algorithm process has the following steps: (1) Build and initialize the Q-Table. The table has n columns, where n is a number of actions. There are m rows, where m is a number of states. This table represents the value of a particular action while in a particular state. (2) Choose and act. The chosen action in the state is based on the Q-Table. Before the agent has sufficient training, it occasionally takes a random action (known as 'exploring') to learn/update the Q-table. After training, the agent takes action with the maximum Q-value based on the table (known as 'exploiting'). In order to balance the exploring period and exploiting period, a greedy epsilon strategy is often applied in Q-learning [40]. After each action, the agent will evaluate and update the function represented by the table $Q(s,a)$. The function is:

$$Q(s,a)_{new} = Q(s,a)_{old} + \alpha \left[R(s,a) + \gamma \max Q'(s',a') - Q(s,a)_{old} \right] \tag{1.3}$$

where, α is learning rate, γ is discount rate, s, a is current state and action pair, and s', a' is future state and action pair.

Deep Q-learning or a Deep Q-Network (DQN) is a recently developed technique within Reinforcement Learning[41]. It overcomes the main defect of traditional Q-Learning [42]: the requirement of a large amount of resources to compute and store the state-action value function, i.e., the Q-table. These requirements result in an unacceptable amount of computation and memory resources, especially when the number of states (related to channels in our problem) is large. In DQN, deep neural networks are used to approximate the state-action value function, $q(s,a;\theta) \approx Q(s,a)$, where, $Q(\cdot)$ is the true Q-value, and $q(\cdot)$ is estimated Q-value, (s,a) is the state-action pair and the estimated parameter vector θ is the vector of weights in

the deep neural network. Thus, the DQN can be trained by minimizing the prediction error of $q(s, a; \theta)$. Meanwhile, during 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 $a_t = \arg\max_a q(s, a; \theta)$ 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 sub-optimal $q(s, a; \theta)$ function that has not yet converged to better $q^*(s, a; \theta)$ (i.e., learning better actions).

With the development of deep neural networks, many research fields have begun to utilize it to improve their performance. For example, in signal processing, [43] utilizes a DNN to rebuild time-domain signal, [44] does antenna selection via deep learning, [45] focuses on channel estimation and direction-of-arrival (DOA) estimation in massive MIMO based on deep learning and so on. As is known, language processing and image processing are two branches of signal processing. In language processing, or more specifically natural language processing (NLP), deep neural networks have already been widely used. In NLP, researchers combine recurrent neural network with deep neural networks to deal with Speech Recognition [46, 47, 48]. In image processing, researchers combine convolutional neural networks with deep neural networks to perform image reconstruction (compression and decompression) and recognition [49, 50, 51, 52]

However, a DQN cannot completely solve a POMDP problem. This is because, in a DQN, the Q-value should be calculated based on the current state which requires a full (and perfect) channel state observation. Thus, if the agent is unable to observe the full state, a DQN-based approach may perform poorly. Therefore, we need to introduce another approach, viz., a recurrent neural network, to accommodate partial observations. We will introduce deep recurrent Q-networks (DRQN) in the following section.

1.4.2 Recurrent Neural Networks and LSTM

The recurrent neural network (RNN) [53, 54] is one class of artificial neural networks that has the capability of modeling the dynamic temporal behavior of a sequence of events. The idea behind an RNN is to utilize sequential information. In a traditional neural network, it is assumed that all inputs (and outputs) are independent of each other. However, if the network wants to predict the next state, it needs to know which observations came before it. Another way to think about an RNN is that it has 'memory' which captures information about what has been calculated so far. In theory, an RNN can make use of information in an arbitrarily long sequence, but in practice they are limited to looking back a finite number of steps. An RNN is called *recurrent* because it performs the same task for every element of a sequence, with the output being dependent on the previous computations.

In a traditional neural network, the basic computing component is termed a 'unit', but in

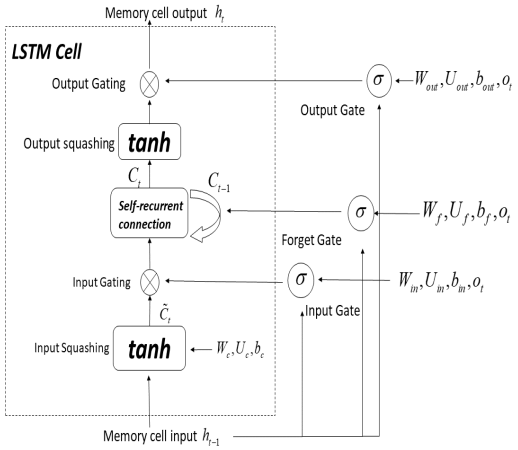


Figure 1.2: The Structure of an LSTM used in the DRQN Nodes

an RNN, the basic unit is termed a 'cell'. In our work, Long Short-Term Memory (LSTM) [55], which is one of the most effective types of cell structures, is applied to an RNN. Its structure is shown in Figure 1.2 and in order to understand the basic principle of an LSTM, a simplified formulation is described in the following.

At time t, the input observation is s_t. First, we compute the input gate values In_t and the candidate value $\tilde{C}_t$ for the states of the memory cells at time t:

$$In_t = sigmoid\left(W_{in}s_t + U_{in}h_{t-1} + b_{in}\right) \tag{1.4}$$

$$\tilde{C}_t = \tanh\left(W_c s_t + U_c h_{t-1} + b_c\right) \tag{1.5}$$

Here, *sigmoid* is known as a logistic function, which we choose to be $sigmoid\,(x) = 1/(1+e^{-x})$, and *tanh* is the hyperbolic tangent function, which is $\tanh\,(x) = (e^x - e^{-x})/(e^x + e^{-x})$ .W_{in}, U_{in} and b_{in} are parameters of the input gate, W_c, U_c and b_c are parameters of the self-recurrent connection and h_{t-1} is information from the memory input. Second, we compute the forget gate value f_t:

$$f_t = sigmoid\left(W_f s_t + U_f h_{t-1} + b_f\right) \tag{1.6}$$

Here, W_f, U_f and b_f are parameters of the forget gate. Based on the values of the input gate, the forget gate and the candidate state, the new candidate state value can be calculated:

$$C_t = In_t \times \tilde{C}_t + f_t \times C_{t-1} \tag{1.7}$$

Finally, we compute the output value Out_t and upgrade the memory information h_t:

$$Out_t = sigmoid\left(W_{out}s_t + U_{out}h_{t-1} + V_{out}C_t + b_{out}\right) \tag{1.8}$$

$$h_t = Out_t \times \tanh\left(C_t\right) \tag{1.9}$$

where, W_{out}, U_{out} and b_{out} are parameters of the forget gate. It is noted that all parameters (weights) in DRQN are updated during learning process.

1.4.3 Deep Recurrent Q Network

In order to solve the POMDP problem, we should make full utilization of previous data. However, inputting previous observations directly into Neural Networks has some problems. First, the method will increase the required resources to store previous data (in [56], its memory size is about 1 million memory tuples). However, not all of the previous information is useful (e.g., if the node only cares about how many steps the last observation will change instead of giving it all the steps' observations). Meanwhile, inputting much more data will also increase the resources for the calculation. Combining a Deep-Q network with a recurrent neural network, not only allows us to make full utilization of sequential information, but can also reduce the redundant information. Unlike the DRQN structures in [57, 58, 59, 60], the proposed DRQN structure is shown in Figure 4.1.

Here, After power sensing, the channel state estimator determined the channel states (idle:0,

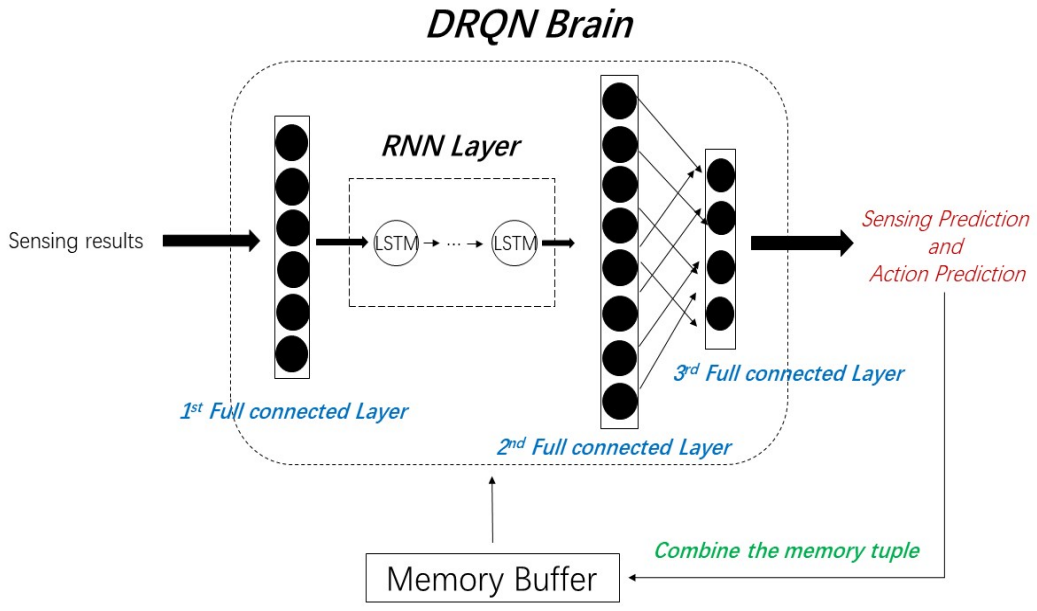


Figure 1.3: The proposed DRQN Structure

occupied:1 and unknown:2). Due to the noise and interference, the power sensing and the estimated process may not be perfect. Meanwhile, compare to DQN, the memory tuple in DRQN is $\langle s, r, a, t, s\_\rangle$ where t is time. In the DRQN brain, the input layer is ignored and the output layer is shown in the last layer.

Therefore, we can abstract the entire mathematical process:

Our goal is find the optimal policy π^* such that

$$Q^*(s,a) = \max_{\pi} E\left[r_t \,|s = s_t, a = a_t, \pi\right] \tag{1.10}$$

Because of different learning approaches, the Q value computation can be represented by different expressions: 1). In DQN approach, the Q value can be represented by $Q(s,a)$, s is the result of sensing/observation and a is action. Meanwhile, due to the structure of DQN, the data we feed into NN is $< s, a, r, s_> $, s is current sensing/observation and $s$is following sensing/observation, r is reward (or penalty), and a is action. 2). In DRQN approach, the Q value can be represented by $Q(s,a,t)$, s is the result of sensing/observation, a is action, and t is the time stamp. Also, due to the structure of DRQN, the data we feed into NN is $< s, a, r, t, s_> $, s is current sensing/observation and s is following sensing/observation, r is reward (or penalty), t is the current time stamp, and a is action. It should be noted that our work mainly has two types of goals: optimal transmission policy and optimal sensing pattern with corresponding transmission policy. Thus, our goals can be represented mathematically using the Bellman equation:

1). DQN:

$$Q_{i+1}(s,a) = E\left(r_i + \gamma \max_{a\_} Q_i\left(s\_, a\_\right) |s, a\right) \tag{1.11}$$

2). DRQN:

$$Q_{i+1}(s,a,i) = E\left(r_i + \gamma \max_{a\_} Q_i\left(s\_, a\_, i\right) |s, a, i\right) \tag{1.12}$$

Here, s represents the result of sensing/observation (i.e, the state). It should be mentioned that it is a partial sensing/observation result due to limited resources. Meanwhile, $s\_$, $a\_$ and r are dependent on the s, a . Further, when the $i \to \infty$, we have $Q_i \to Q^*$, i.e., we will approach the optimal policy.

Because our work focuses on two different aspects of the DSA problem (sensing and transmission), we use different definitions of the reward, r depending on the specific application:

1. Optimal transmission policy:

$$r = f\left(\rho, \varepsilon \,|s, a, t\right) \tag{1.13}$$

   and if the cache is considered, the reward is formulated as:

$$r = f\left(\rho, \varepsilon, \omega \,|(s, \omega), a, t\right) \tag{1.14}$$

2. Optimal sensing pattern with corresponding transmission policy:

$$r = f\left(\rho, \varepsilon, N \,|s, a, t\right) \tag{1.15}$$

   and if the cache is considered, the reward becomes:

$$r = f\left(\rho, \varepsilon, N, \omega \,|(s, \omega), a, t\right) \tag{1.16}$$

Note that $f(\cdot)$ is the reward function. In DSA, our ultimate goal is to improve the throughput of the secondary network and reduce the collision rate with the primary network. Thus, the reward is determined by three variables, ρ (throughput), ε (collision rate), and N (number of sensing/observation channels). However, if we consider the cache, the reward will be a function of four variables: ρ, ε, N, and ω (cache state). Here, it should be mentioned that

the cache state is not only a part of the result but also a parameter for adjusting the reward. It's worth stating that the function is positively correlated with ρ and negatively correlated with and N. When we optimize the transmission strategy with a fixed observation pattern, we ignore N. Meanwhile, the cache state ω is a strategy for adjusting rewards in the form of a piecewise function.

An important issue that needs to be discussed is the relationship between the neural network's size and the system's complexity. In the current work, we have not provided a quantitative analysis but have drawn some qualitative conclusions. In the [61, 62, 63], researchers concluded that the size of a neural network affects the performance (e.g., the accuracy of classification). In our problem, we found that when the number of communication channels is large enough, we need to increase the neural network size to guarantee performance exponentially. However, the more extensive the network, the larger the training time and the amount of data required for training. Neither of these requirements may not be feasible in a practical communication node. Therefore, when we use machine learning in DSA, we make one underlying assumption: our hypothetical environment has a limited number (no more than 10-20) of channels.

1.5 Contributions to the State of the Art

The main contributions of this dissertation are to combine deep reinforcement learning with recurrent neural networks for partial observation-based dynamic spectrum access. In particular, we propose Deep Q learning with an LSTM structure, referred to as DRQN, to learn different environments under partial observation/sensing. Using the DRQN approach, the proposed node can learn to avoid primary users and improve throughput and robustness

with limited spectrum resources. Additionally, in order to reduce delay, we introduce a cache module into the proposed DRQN. Using a reasonable definition of the cache state and corresponding dynamic reward structure, our proposed DRQN approach can maintain proper cache storage and keep relatively high throughput, low collision rate, and suitable delay and packet loss rate. This dissertation will weave together notions from deep learning, reinforcement learning, and dynamic spectrum access to enable these contributions. In summary, our contributions are given as follows:

1.5.1 The Application of Deep Reinforcement Learning to Distributed Spectrum Access in Dynamic Heterogeneous Environments with Partial Observations

In this portion of the dissertation, we assume that SUs sense channel occupancy using a specific observation pattern and determine which action to take (i.e., which band to transmit in or do not transmit at all, also known as the transmission policy) at the next step to improve system throughput and reduce collisions. Note that the state can change at each step. Since throughput is hard to represent directly as input to a neural network, we use successful transmissions and collisions instead. In other words, if the node transmits and the corresponding channel is idle in that time step, the transmission succeeds, and the user receives a positive reward. The success or failure of transmission is assumed to be known using standard ACK/NACK messages. After a successful transmission, the benefits to throughput are reflected in the assignment of a positive reward. If the channel is occupied at the time of the SU transmission, a collision occurs, the transmission fails, and there is a penalty incurred. As mentioned, nodes can also choose to wait (i.e., not transmit), resulting in no

reward or penalty. The specific contributions of this part of the work are:

1. We provide a solution when all channels in the environment are independent, which are more difficult and practical than correlated channels.

2. We provide a solution that can handle partial observations of the channel state, which is again a more practical assumption due to limited resources and a desire for reduced energy consumption.

3. We provide a solution that does not require knowledge of the behavior patterns of each channel or the other nodes in the system. We demonstrate this using two very different types of environment without any prior knowledge: stochastic and deterministic as well as various combinations of the two. Thus, our proposed approach is a general solution to the problem.

4. We show that the proposed DRQN can tolerate different levels of imperfect environmental feedback as well as hidden nodes.

5. We demonstrate that the proposed DRQN can also handle a dynamic environment. Meanwhile, it can not only learn different environments, but also has reasonably fast convergence speed. Moreover, the results provide an important insight: pre-training can provide performance improvement (in terms of convergence speed) for learning nodes.

6. We demonstrate that the approach allows for multiple learning agents to be deployed in the same environment without introducing convergence issues or instability.

7. We develop and demonstrate an approach that includes cache information so that packet delay and drop rate can be controlled.

1.5.2 Improved Dynamic Spectrum Access through DRQN-based Flexible Sensing and Exploitation of the Cache State

Although partial observations are accounted for in the previous work, the approach utilizes a fixed sequential observation pattern (e.g., observing half of the channels followed by observing the other half). Obviously, in some scenarios, a fixed observation pattern may result in a waste of resources. Thus, in this work, we combine Deep Q learning methods with an RNN, in particular long-short term memory (LSTM), to learn the optimal (or near-optimal) sensing policy that chooses one or two channels at each sensing interval to obtain (imperfect) observations of a small fraction of the available channels. We will show we can obtain near-optimal performance with substantially less channel sensing by directing sensing rather than simply transmission. Additionally, we show that by modifying the reward structure based on the number of packets in the cache and adding cache information to the DRQN input, we can exploit multiple open channels and coarsely control packet delay and drop rate. More specifically, in this work, we provide the following contributions to state of the art:

1. We develop and characterize a DSA approach using a DRQN to drive sensing decisions that substantially reduces the number of channels that must be sensed each timeslot with very little loss in performance.

2. We show how relative reward settings can be used to control the behavior of the DSA approach making it more or less aggressive (trading throughput for collision rate).

3. We develop and demonstrate an approach that includes cache information so that packet delay and drop rate can be controlled.

4. We demonstrate that the approach allows for multiple learning agents to be deployed in the same environment without introducing convergence problems or instability. This is sometimes termed the "multi-agent problem". We also demonstrate that the approach reacts well to changes in the environment and converges quickly, especially if the environment has been seen previously.

1.5.3 Accelerating Reinforcement Learning in Dynamic Spectrum Access

In our previous works [64, 65], we utilize a DRQN-based technique (combining a recurrent neural network [55] with a DQN) for making access decisions that are near-optimal despite using partial observations of the spectrum. However, among the previous work in the area of DRL for DSA, there has not been a concerted effort to improve a critical aspect of this online learning approach: convergence speed. In a practical communication system, a user shouldn't wait long to achieve the expected QoS performance. Therefore, improving the learning efficiency and thus reaching the expected performance faster (with limited computing resources) is a goal worthy of research. Thus, in this work, the current work focuses on convergence speed under the assumption of partial observations in a dynamic spectrum access application using deep reinforcement learning. In contrast to most work applying reinforcement learning to DSA, our goal is for the agent to learn the sub-optimal (or even optimal) access action with accelerating converge speed. As we know, in our previous works [64, 65], and other prior works [24, 56, 64]), the convergence speed is an important but little-studied topic. Therefore, in this work, we mainly focus on how to improve the convergence speed. This is an uncoordinated multi-channel access problem with discrete channels under partial observations. In our assumed model, each channel has three possible states (occupied, idle, and unknown). Each PU transmits according to some predetermined schedule and is

assumed to avoid collisions with other PU's. Additionally, the PUs ignore the activity of the SUs, as SUs must avoid the PUs, not vice-versa. Further, each SU has to observe at least some of the channels and choose one channel on which to transmit or choose to wait until a later time. If the SU chooses to transmit and the selected channel remains idle, the transmission is successful (i.e., we assume that interference dominates performance). Otherwise, there is a collision, and the transmission fails. Because of the limited resources available, SUs can observe some channels' states with imperfect feedback each timeslot but not all channels. In this work, we examine three approaches to accelerate learning: (1) learning frequency (parallel learning structures or multiple learning [66]), inherited learning (general transfer learning [67, 68]), and Meta-learning [69, 70]. At the same time, it should be noted that the scenarios in which these three different methods can be used are different. The results demonstrate that the three different approaches can each accelerate the convergence speed and guarantee near-optimal (sometimes even optimal) performance under partial observation constraints. More specifically, in this work, we provide the following contributions to the state of the art:

1. We implement parallel learning structures or multiple learning in a dynamic spectrum access application under partial observations

2. We implement general transfer learning in a dynamic environment under partial observations.

3. We implement Meta-learning with DRQN in dynamic spectrum access application under partial observations.

4. Finally, we compare the three approaches and conclude that all three approaches can accelerate the convergence speed and guarantee near-optimal performance under par-

tial observation constraints. Further, the results show that meta-learning is the best choice for accelerating the learning process. However, general transfer learning is better if the system requires a faster accelerating learning process. The reason is that if the running time is long enough, Meta-Learning must be the best way to improve the convergence speed. However, the best premise is that the period is long enough. The advantage of general transfer learning is that it can quickly improve the convergence speed, but its performance has limitations. For example, if the original convergence takes 10,000 steps, general transfer learning may increase the convergence speed to 5,000 steps earlier than the third run. But ML may need to hit the same level of convergence speed around the fifth run. However, as time goes on, the final ML may be able to reduce the convergence speed up to 3000 steps. Parallel learning is found to be the most straightforward way to accelerate learning. However, it needs more computing and storage resources than the other two approaches.

1.6 List of Publications

- Xu, Y., Yu, J. and Buehrer, R.M., 2020. "The application of deep reinforcement learning to distributed spectrum access in dynamic heterogeneous environments with partial observations," *IEEE Transactions on Wireless Communications*, 19(7), pp.4494-4506.

- Xu Y., Yu J, Buehrer R.M., Cache-Enabled Dynamic Spectrum Access via Deep Recurrent Q-Networks with Partial Observation. In2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN) 2019 Nov 11 (pp. 1-2). IEEE.

- Xu Y, Yu J, Buehrer RM. "Dealing with partial observations in dynamic spectrum access: Deep recurrent Q-networks," In *2018 IEEE Military Communications Conference (MILCOM)* 2018 Oct 29 (pp. 865-870).

- Xu Y, Yu J, Headley WC, Buehrer RM. Deep reinforcement learning for dynamic spectrum access in wireless networks. InMILCOM 2018-2018 IEEE Military Communications Conference (MILCOM) 2018 Oct 29 (pp. 207-212). IEEE.

- Yu, Jianyuan, Yue Xu and R. Michael Buehrer. "Predicting Bit Error Rate from Meta Information using Random Forests and Partial Data Augmentation." SoutheastCon 2021. IEEE, 2021.

- Xu, Y., Yu, J. and Buehrer, R.M., 2021. Improved Dynamic Spectrum Access through DRQN-based Flexible Sensing and Exploitation of the Cache State. IEEE Transactions on Wireless Communications. (Submitted)

- Xu, Y., Yu, J. and Buehrer, R.M., 2022. Accelerating Reinforcement Learning in Dynamic Spectrum Access(To submit)

- Jianyuan Yu, William W Howard, Xu, Y., and R. Michael Buehrer. "Model Order Estimation in the Presence of Multipath Interference using Residual Convolutional Neural Networks." 2022 IEEE TWC (submitted).

Chapter 2

Optimizing Transmission Policy in Dynamic Spectrum Access with Partial Observations

As we mentioned in the abstract, over the past ten years, global mobile data traffic has experienced tremendous growth making spectrum resources even more critical for future wireless communication networks. However, the spectrum is an exorbitant and scarce resource. Thus, Dynamic Spectrum Access (DSA) [20, 21] is a key concept that has the potential to improve spectrum utilization efficiency in wireless networks and help meet the aforementioned need for more spectrum. In the context of DSA research, a standard assumption has been established that secondary users (SUs) may search and use idle channels that are not being used by the primary users (PUs).

This chapter focuses on spectrum prediction in which the SUs sense and analyze the channel states at the current time and predict the optimal transmission actions (e.g., to transmit in a particular band or wait) during the next time slot. In general, this is an uncoordinated multichannel access problem with independent channels under partial observations. Each channel has three possible states (occupied, idle, and unknown) at any given time. Each PU in the environment selects one channel at each time step to transmit a packet (PUs transmit without sensing and are assumed to avoid collisions with each other; they also ignore SUs).

The SUs must observe the spectrum (channels) and choose a channel to transmit or delay its transmission until later. If the SU chooses to transmit and the selected channel is idle during the transmission, we assume the transmission is successful. However, if any other node has selected the channel, there is a collision. The SU is assumed to be able to detect the channel state, although not necessarily perfectly. The PUs are assumed to be of various types based on their spectrum behavior which will be described in more detail shortly.

Generally speaking, in order to exploit idle channels and improve spectrum utilization efficiency without any prior knowledge, reinforcement learning (RL), especially Markov Decision Process (MDPs), is one potential solution [42]. The advantage of RL is learning through experience in situations where knowledge about the environment is uncertain but can be learned. However, traditional RL methods typically estimate transition matrixes and reward matrices requiring prohibitively large computational resources when applied to dynamic spectrum access. Thus, in order to overcome this challenge, Deep Reinforcement Learning [40], and in particular, the Deep Q-Network (DQN) has been proposed to solve the optimal channel access policy via online learning. DQN and its derived algorithms (e.g., Double-DQN and Dueling-DQN) are able to deal with large state spaces and find a good suboptimal or even the optimal policy directly from experience without any prior information about the system dynamics [40][71].

Obviously, the more information the SU obtains, the higher the probability that it will select the optimal action. However, in practice, the node cannot observe all channel states at each time step. Thus, the learning approach (e.g., a DQN) may not be able to completely solve the resulting Partially-Observable Markov Decision Process (POMDP). Although the SUs can't observe all state information simultaneously, it can observe the full environment sequentially. Extracting and analyzing the related information from these sequential observations can be achieved by combining a recurrent neural network [55] with a DQN. The resulting structure is known as a Deep Recurrent Q-Network (DRQN), which we show can

predict correct actions at the following step by using partial observations. We will show that this approach is effective for DSA in the examined environments.

2.1 Related Work

In recent years, the development of machine learning has attracted the attention of researchers in the area of wireless communications, as it is applied to many of the challenges facing wireless systems. Many works have begun to focus on the more practical and complex DSA problems where the system dynamics are unknown. The most popular algorithm for accomplishing this is Q-learning since it is a model-free method and can learn the optimal policy directly via online learning, [72, 73, 74, 75, 76]. These works mainly use Q-learning and its derived algorithms (e.g., SARSA) to establish and tabulate the transition matrix based on observations that learning nodes use to make decisions. Compared to traditional algorithms, these methods are better able to deal with the lack of prior knowledge. However, the complexity of tabulating the Q-functions is still fairly high, and the performance still has room for improvement. 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 classic Q-learning [40, 41].

Recently, some researchers have attempted to solve the DSA problem using a DQN. For example, [56] examines correlated channels in DSA. However, when channels are correlated, the number of possible states in the entire system decreases. Further, because DRL doesn't need to calculate the system transition matrix when channels are correlated, fewer possible states will reduce the difficulty of learning. Thus, to some extent, independent channels would be more general. Real-world tasks often feature incomplete and noisy state informa-

tion resulting from partial observability and imperfect feedback. Unfortunately, DQNs are limited in the sense that they learn a mapping from a limited number of past states. Thus, in general, a DQN is unable to fully solve Partial Observation Markov Decision Problems (POMDPs) in a communication system. However, by combining Recurrent Neural Networks (RNN) with a DQN, [57] and [58] solve such POMDPs in computer games. Based on the core ideas of an RNN, some researchers, such as in [59] and [60], provide a solution that combines an RNN with a DQN to solve such DSA problems and made some breakthroughs. In [59], the DQN is trained for all users at a single unit (e.g., the cloud) which means the nodes update their DQN weights by communicating with the central unit. However, in our problem, the learning nodes are unable to communicate with each other even through a central unit. Moreover, the assumption of offline training and the number of channels in [59] are very different than what is considered here. [60] combines a reservoir computing algorithm (one kind of RNN) with a DQN to solve POMDPs when learning nodes can't communicate with each other, but it is not a clear spectrum prediction method, and the performance could be improved in terms of both convergence speed and collision rate. Thus, in this paper, we will attempt to combine these DRL methods with another kind of RNN to solve the spectrum prediction problem with partial observations.

As compared with previous related work, this work advances state of the art in several aspects:

1. We assume all channels in the environment are independent, which is a more difficult and practical problem than correlated channels.

2. We assume the learning agent only has access to partial observations of the channel state due to limited resources or a desire for reduced energy consumption.

3. We assume that the DRQN node does not know the behavior patterns of each channel

or the other nodes in the system. We examine two very different types of environments without prior knowledge: stochastic and deterministic, as well as various combinations of the two. Thus, the developed approach is a more general solution.

4. We show that the proposed DRQN can tolerate levels of environmental feedback error up to 15% and even hidden nodes.

5. We demonstrate that the proposed DRQN can also handle a dynamic environment. In fact, it can not only learn a variety of environments, but it does so with a relatively fast convergence speed. Moreover, the results provide an important insight: pre-training can improve performance (in terms of convergence speed) for learning nodes.

6. We demonstrate that the approach allows multiple learning agents to be deployed in the same environment without introducing convergence issues or instability.

7. We develop and demonstrate an approach that can include cache information so that packet delay and drop rate can be (coarsely) controlled.

2.2 System Model and Problem Formulation

2.2.1 System Model

In this paper, we consider a dynamic spectrum access scenario with a mesh network of $2N$ primary radio nodes, including N transmitters and N receivers. The N primary transmitters choose to transmit on one of K independent channels with a behavior that is based on their node type at each time step. In the system, we assume that there are several types of nodes:

- Legacy node (PU): Occupies one fixed channel at all times

- Hopping node (PU): Dynamically occupies a single channel which changes at each time step using a regular pattern
- Intermittent node (PU): Periodically occupies one fixed channel (this could be a TDMA node)
- Stochastic node (PU): Occupies one fixed channel based on a probability distribution
- Greedy node (SU): Occupies the first available channel it observes. It can be regarded as a type of DSA node
- Myopic node (SU): Learning node with Myopic policy [25]
- DQN node (SU): Learning nodes using standard DQN [77]
- DRQN node (SU): The proposed learning node

It should be noted that, in the simulated environments, the Legacy node, Hopping node, Intermittent node, and Stochastic node are PUs. Meanwhile, the Greedy node, Myopic node, DQN node, and DRQN node are SUs that can examine ACK/NAK information to determine whether their respective transmissions were successful. We also 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. Further, the channel states can change at each step.

2.2.2 Problem Formulation

Based on the system model assumed above, SUs sense channel occupancy using a specific observation pattern and determine which action to take (i.e., which band to transmit in or

not transmit at all) at the next step to improve system throughput and reduce collisions. Note that the state can change at each step. Since throughput is hard to represent directly in a neural network, we use successful transmissions and collisions instead. In other words, if the node transmits and the corresponding channel is idle in that time step, the transmission succeeds, and the user receives a positive reward. Success or failure of transmission is assumed to be known using standard ACK/NACK messages. During a successful transmission, the benefits in terms of throughput are reflected in the assignment of a positive reward. If the channel is occupied at the time of the SU transmission, the transmission fails, and a penalty is incurred. As mentioned, nodes can also choose to wait (i.e., not transmit), which results in no reward or penalty.

The goal of this node is to optimize (i.e., maximize) its network throughput while minimizing collisions. However, a DRQN is designed to maximize utility (essentially long term expected reward):

$$U = \max_{a} Q(s, a) \tag{2.1}$$

where $Q(s, a)$ is the Q-function which represents the value of taking action a when in state s. and is at the core of Q-learning. In standard Q-learning, it is updated by:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{2.2}$$

Where, s is the current channel state and the s' is the following channel state, a is the current action and a' is the following action, r is the immediate reward and γ is the discount parameter (between 0 and 1).

In our work, As stated above, we formulate the problem as follows: our goal is to maximize throughput while minimizing collisions. Throughput is directly related to successful transmissions. Thus, we seek to maximize the number of successful transmissions while

minimizing collisions. To do this within the context of Q-learning, we define the rewards as positively related to successful transmissions and negatively related to collisions. The goal is then to maximize expected cumulative reward. We do this using Q-learning which learns the Q-value (the long term utility or discounted cumulative reward of taking an action when in a particular state) of each state-action pair. Specifically, we learn the Q-values using a deep recurrent neural network and then choose the action with the maximum the Q-value in each state.

2.3 Simulation results

This section investigates the performance of the proposed DRQN node via simulation using TensorFlow [78]. In the overall communication environment, we assume that there exist 10 discrete frequency channels. For these experiments, we have two kinds of learning nodes. The DQN structure has four fully connected hidden layers (each layer containing 30, 40, 50 and 25 units respectively). Our proposed DRQN neural network structure has five layers: one input layer, one output layer, one RNN (40 hidden cells) layer and two fully connected layers (containing 50 and 25 units respectively). The activation function used for the fully connected layers is the ReLU (Rectified Linear Unit) function. When updating the weights of the network, a minibatch of experience-tuples are randomly selected from a memory box of 1500 prior experiences for the computation of the loss function [40]. The Adam algorithm [79] is used to conduct a stochastic gradient descent for updating. In order to avoid getting stuck with a suboptimal decision policy before the node has sufficient learning experience, we apply an adaptive ε greedy algorithm [40] which is initially set to 0.1 and is decreased by 0.00005 every time step until reaching 0.01. Moreover, in order to obtain an accurate

estimate of the performance, all simulation results are the average of more than 20 runs.

2.3.1 Reward Setting

The reward setting is an interesting issue that we have not fully explored in this paper. However, our choice for the reward is, we believe, a logical starting point. Before explaining the physical meaning, we should emphasize that the goal of our DRQN is to improve system throughput while minimizing collisions. However, throughput is hard to input to a NN directly. Thus, we use successful transmissions instead. In other words, if the DRQN selects an idle channel and completes a transmission successfully, the agent receives a reward. Therefore, the physical meaning of the reward is that it is directly tied to successful transmissions. Further, the reward is negatively related to collisions (e.g., there is a penalty associated with collisions). Thus, collisions are worse than not transmitting at all. This reduces collisions, which is an important aspect of system performance in secondary spectrum sharing.

To be clear, the value of reward and penalty of the DRQN can be varied. Based on our experience, if the value of the reward is much more significant than (let's say more than twice) the value of the penalty, the learning node becomes aggressive (i.e, it risks collisions in an attempt to increase throughput). On the other hand, if the value of the penalty is higher, the learning node becomes conservative. The reward setting, in our paper, is moderate (i.e., in between aggressive and conservative): the value of the reward for a successful transmission is 100 (per channel) and the the of the penalty for a collision is -50 (per channel). Thus, the reward is essentially a guide to desired behavior. Further, it should be noted that the relationship between the reward and the penalty values is more important than the absolute values of the reward and the penalty.

2.3.2 Stochastic Environment

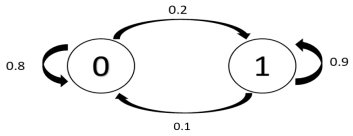


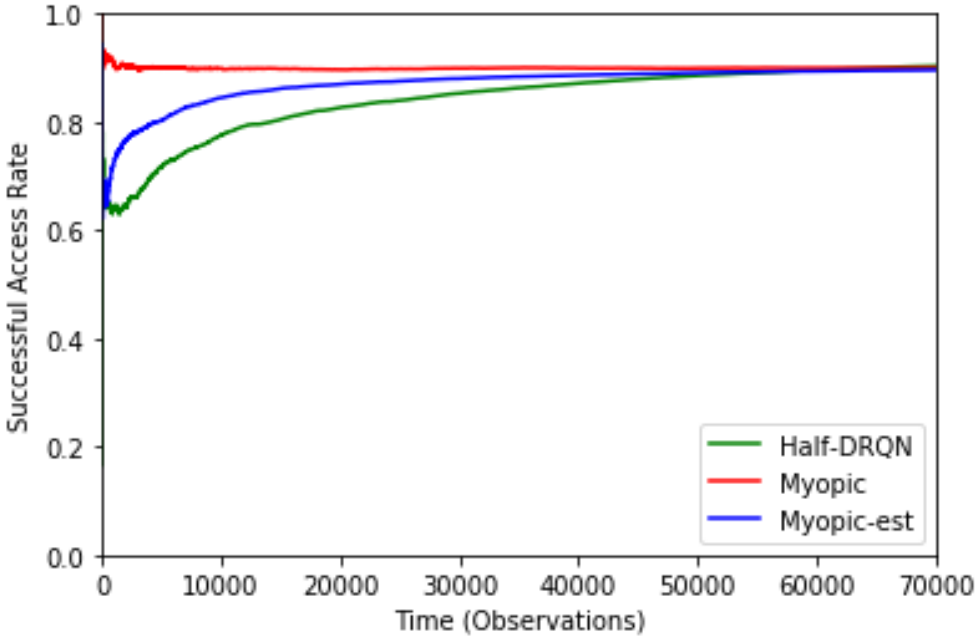
Figure 2.1: Markov Chain Model of Stochastic PU

In the first set of simulations, the DRQN nodes are able to observe one half of the channels during each time step. In the stochastic environment #1, all channels are occupied by stochastic nodes which are described by a two-state Markov chain as shown in Fig 2.1. The transition probability of the two-state Markov chain on the nth channel is:

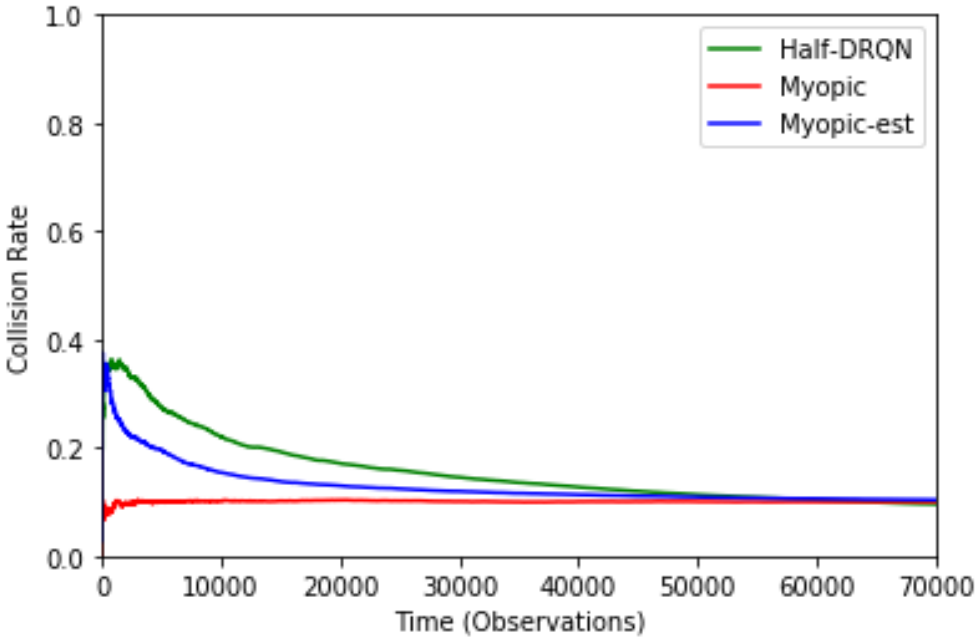
$$P_n = \begin{pmatrix} p_{00}^n & p_{01}^n \\ p_{10}^n & p_{11}^n \end{pmatrix} \tag{2.3}$$

where, $p_{ij} = Pr\left(state_j | state_i\right)$. Because [25] has already proved that, in this type of environment, the Myopic approach with prior knowledge is optimal, we wish to compare a DRQN with the myopic approach. Meanwhile, because the node with the Myopic Policy needs the channel transition probabilities, we examine two different kinds Myopic nodes: (a) the Myopic node with prior knowledge of P_n; (b) Myopic node which must estimate P_n.

To examine the performance, we investigate two metrics, the successful access rate and the collision rate. The successful access rate is the number of successful transmissions divided by the number of time slots considered. The collision rate is the number of collisions divided by the number of time slots. The two metrics are plotted versus time in Fig 2.2 where 'Myopic' refers to a myopic node with prior knowledge; 'Myopic-est' refers to a myopic node with random initial Markov transition matrices which are updated each step (the myopic node will take actions based on the current estimated transition matrices); 'Half-DRQN' is a DRQN which can observe half the channels at each time step. From Fig 2.2, the proposed DRQN



(a) Successful Access Rate in Stochastic Environment #1



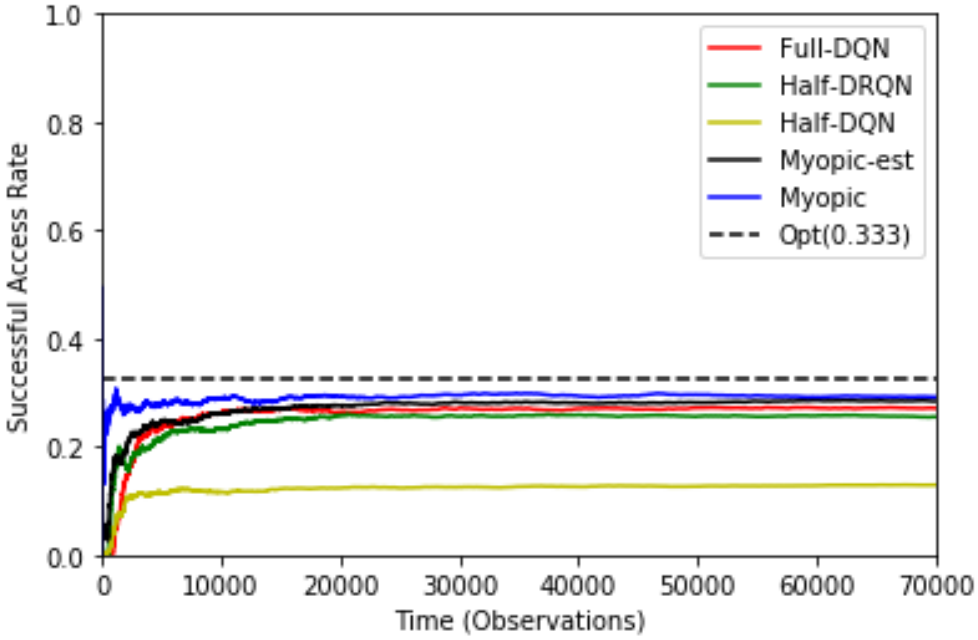
(b) Collision Rate in Stochastic Environment #1

Figure 2.2: Performance (Successful Access Rate and Collision Rate) in Stochastic Environment #1

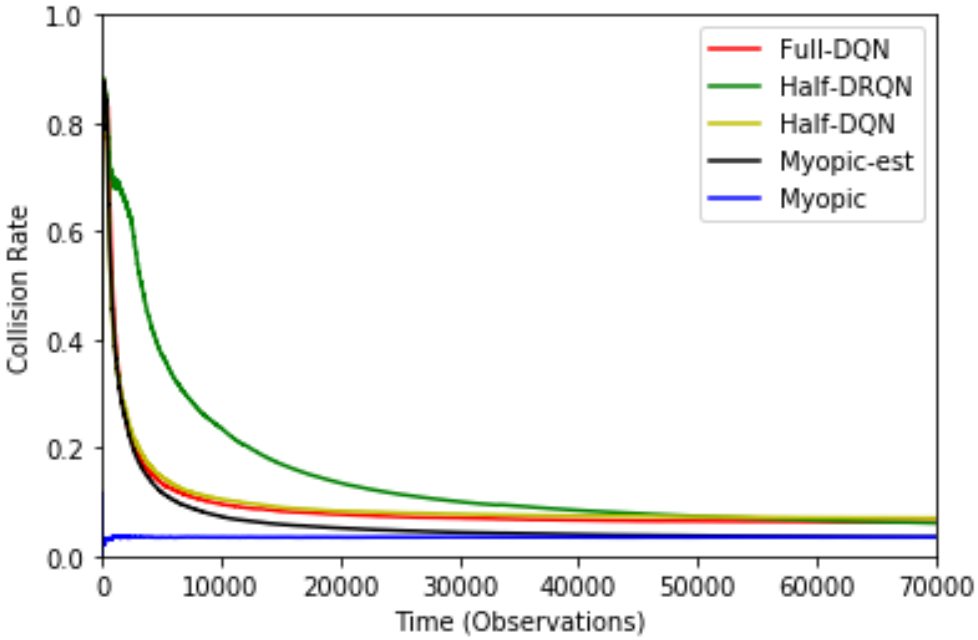
node can reach optimal performance although more slowly than 'Myopic', which is always optimal, and 'Myopic-est' which learns the state transition matrices. Since all PUs follow the Markov chain model of Fig 2.1, it is expected that a myopic approach should be optimal. Further since all channels are occupied by stochastic PU nodes, there are potentially many available channels at any given time. To get better understanding of the DRQN's capability, we should examine a more extreme communication environment.

A more extreme scenario has only one available channel. Thus, Stochastic Environment #2 contains nine legacy nodes (#1 to #9) and one stochastic node (#10) which means the learning nodes can only utilize the idle period from channel #10 (which has 0.333 probability of being idle). Here, we also examine the performance of a DQN node with full observations and a DQN node with 50% observations.

According to Fig 2.3, among all nodes, the successful access rate of the myopic approach is the best which can reach approximately 31% while the DRQN has a successful rate of about 29% and the full observation DQN is about 30%. However, the collision rate of the DRQN, full observation DQN and half observation DQN are about 5% while myopic with prior-knowledge and Myopic-est are about 3%. Based on the low collision rate we conclude that all the learning nodes can learn to wait when they believe that there is no available channel to access. Second, once the learning has converged, the collisions occur at the edges of channel state changes. As one might expect, when a DQN has partial observations, its successful access rate is significantly lower than full observations, although perhaps surprisingly their collision rates are almost the same. The reason is, as compared to the DRQN node, the DQN node has no structure to utilize previous observations and must be more conservative. Thus, we can conclude that the DRQN node can extract and utilize the information from previous observations and predict the future states. Overall, in a stochastic environment, the proposed DRQN node is able to learn the optimal approach, even when observing the channels only half of the time.



(a) Successful Access Rate in Stochastic Environment #2



(b) Collision Rate in Stochastic Environment #2

Figure 2.3: Performance (Successful Access Rate and Collision Rate) in Stochastic Environment #2

2.3.3 Complex Environments with Different Observation Patterns

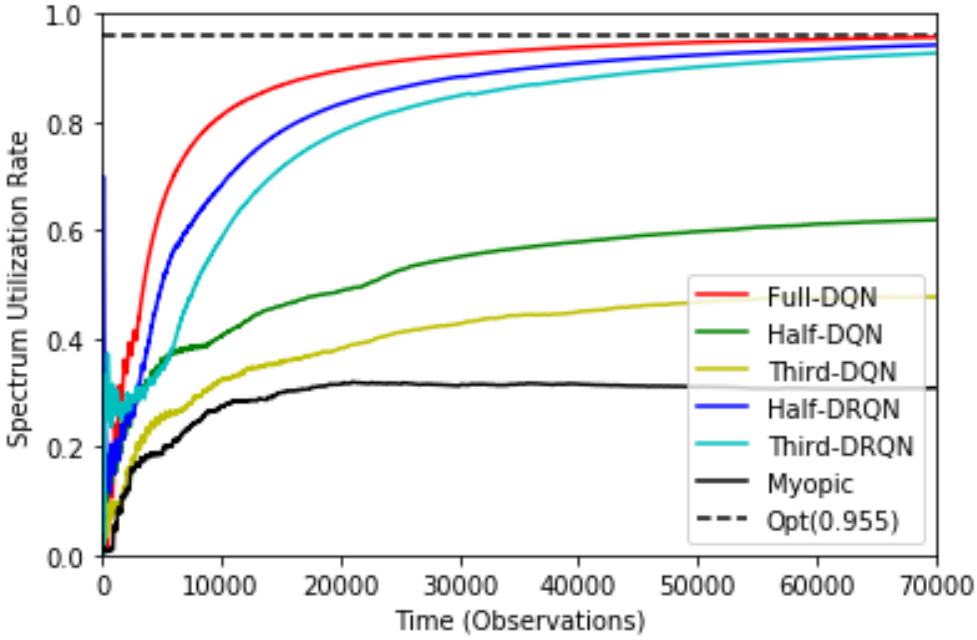
In this set of simulations, we compare two different observation patterns: (1) Observing one half of the channels at each step and (2) Observing one third of the channels at each step. Additionally, the proposed node is examined under two different complex scenarios. We want to determine whether or not, in complex environments, the proposed DRQN node can learn the pattern of heterogeneous nodes includes legacy nodes, hopping nodes, intermittent nodes, hidden nodes and greedy nodes. In order to illustrate the improvement over other approaches, a DQN with different observation rates and a node employing a Myopic Policy are also examined. It should be noted that, because the channels don't have knowledge of the channel transition matrices, the Myopic node will estimate the transition probabilities during the first 1000 steps using full observations.

In the Complex Environment #1, there is one hidden node (channel #1) and two legacy nodes (channels #2 and #3). Further, there are three hopping nodes using dynamic channel occupancy patterns of #5#6#7 → #6#7#8 → #7#8#5 → #8#5#6 → #5#6#7[2]. There are also intermittent nodes on three channels: #4 (node occupies this channel for 250 consecutive steps and is idle 12 consecutive steps), #9 (node occupies the channel for 250 consecutive steps and is idle for 10 consecutive steps) and #10 (node occupies the channel for 250 steps and is idle for 5 steps). Thus, in this system, there is always an available channel that can be used by the node under test (DQN, DRQN or myopic). It also should be noted that the channel occupied by the hidden node appears idle when the learning node is sensing, but is actually occupied.

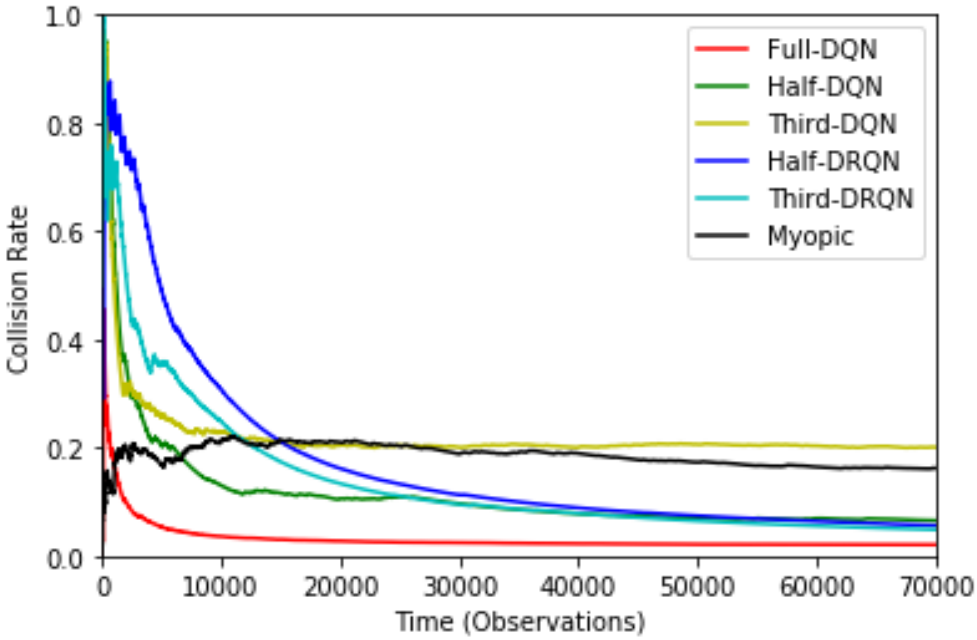
[2]#X#Y#Z represents the channels occupied by the three hopping nodes at a particular time. The sequence represents the hopping pattern of the three nodes

For these simulations, we define the spectrum utilization rate as the number of the channels occupied without collisions divided by the total number of channels. Assuming perfect operation of a DRQN node (i.e. it always finds an open channel), the spectrum utilization rate is about 95.5%. If the node learns perfectly, the full spectrum utilization will be obtained except when the intermittent nodes aren't transmitting. Thus, the highest possible value is approximately $1 - \max(5/255, 10/260, 12/262) \approx 0.955$. From Fig 2.4, it can be seen that, firstly, compared to other learning policies, the myopic policy isn't effective in this environment. Second, as compared to the full observation DQN node, the DRQN node learns the occupancy patterns and attains near optimum full spectrum utilization rate. Thus, the DRQN node can learn the pattens of the legacy nodes, hopping nodes, intermittent nodes and hidden nodes. However, the DRQN node's collision rate, which is approximately 5% in the environment, is slightly higher than the full observation DQN. The reason is, due to the partial observation of the DRQN (i.e. the node can not have full information of the system at each time slot) and the reward setting (i.e. reward of successful transmission is much higher than the penalty of a collision), in order to increase the channel utilization rate, the DRQN policy will take more risk which causes the collision rate to be increased slightly. However, from another viewpoint, the DRQN node only sacrifices a 3% collision rate but uses half the sensing resources, which is undoubtedly a worthwhile tradeoff.

In the Complex Environment #2, we assume one hidden node (#1), two legacy nodes (channels #2 and #3), three hopping nodes, one greedy node, one DRQN node and two intermittent nodes. The dynamic pattern of the hopping nodes is: #4#5#6 $\rightarrow$ #5#6#7 $\rightarrow$ #6#7#8 $\rightarrow$ #7#8#4 $\rightarrow$ #8#4#5 $\rightarrow$ #4#5#6. Intermittent nodes occupy channels: #9 (node occupies the channel for 250 consecutive steps and is idle for 10 consecutive steps) and #10 (node occupies the channel for 250 steps and is idle for 5 steps). The greedy node and the DRQN observe the channel state and take actions at the same time. Here, because the DRQN node and the greedy node take actions simultaneously, the greedy node could be



(a) Spectrum Utilization Rate in Complex Environment #1



(b) Collision Rate in Complex Environment #1

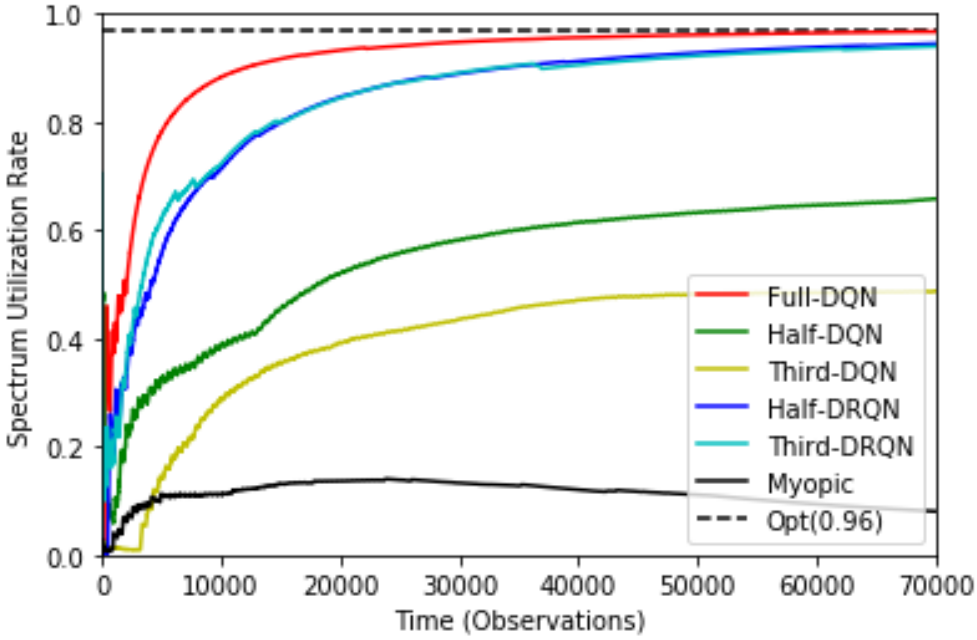
Figure 2.4: Performance (Spectrum Utilization Rate and Collision Rate) in Complex Environment #1

seen as a special type of hidden node which has dynamic attributes.

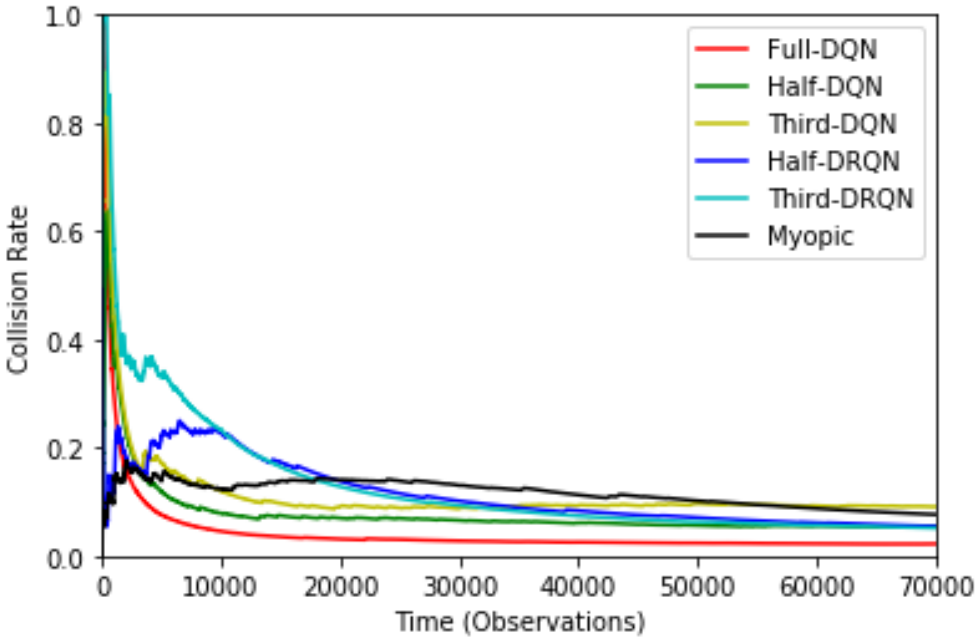
Assuming perfect operation of the DRQN node (i.e., it always finds an open channel), the spectrum utilization rate is about 96% ($1-\max(5/255, 10/260) \approx 0.96$). It can be seen that, from Fig 2.5, the Myopic node is again unable to provide a successful channel access strategy. Compared to the full observation DQN node, the DRQN node can also provide good performance, even close to the optimal full spectrum utilization rate. Thus, the proposed DRQN node can learn the patterns of the legacy nodes, hopping nodes, intermittent nodes, hidden nodes, and greedy nodes. However, the DRQN node still has an approximately 5% collision rate which is relatively high as compared to the full observation DQN. The reason is the partial observation of the DRQN (i.e., the node does not have full information of the system at each time slot) and the reward setting (i.e., reward for successful transmission is much higher than the penalty of collision), which motivates more risky behavior and an increased collision rate.

Based on both simulations, we find that because channels in both environments don't meet the requirements mentioned in [25], the Myopic node can not achieve optimal performance, which is simply due to the fact that the nodes do not follow the model of Fig.2.1. Thus, the performance of the Myopic node is poor. Compared to the performance of the DQN node with partial observations, the DRQN node is able to reach optimal performance with fewer observations, which means the DRQN node is robust in the presence of partial observation patterns. The reason is that although the number of observed channels is reduced, the DRQN can give the corresponding predictions based on the sequential information from previous observations. Therefore, its performance is good and relatively stable. Conversely, due to the structure of the DQN, it can not fully utilize previous observations. Thus, the performance without full observations is significantly worse than the DRQN node.

Second, when the environment becomes more complex, the performance of the DQN node with fewer observations and the Myopic node is worse. Conversely, the DRQN node can still



(a) Spectrum Utilization Rate in Complex Environment #2



(b) Collision Rate in Complex Environment #2

Figure 2.5: Performance (Spectrum Utilization Rate and Collision Rate) in Complex Environment #2

ultimately reach near-optimal performance in both environments. It can be concluded that the DRQN node can adapt to different communication environments without adjusting its parameter settings.

Based on the results of section 2.3.8 and section 4.3.2, we find that the Myopic is optimal in environments that meet the appropriate model assumptions. However, when faced with environments that violate these assumptions, performance suffers. Conversely, regardless of the environment, the DRQN node is able to perform well and even reach near-optimal performance. Further, in complex cases, the performance of the DRQN node with different observation patterns remains stable while the performance of the other nodes does not.

2.3.4 The Effect of Imperfect Observations

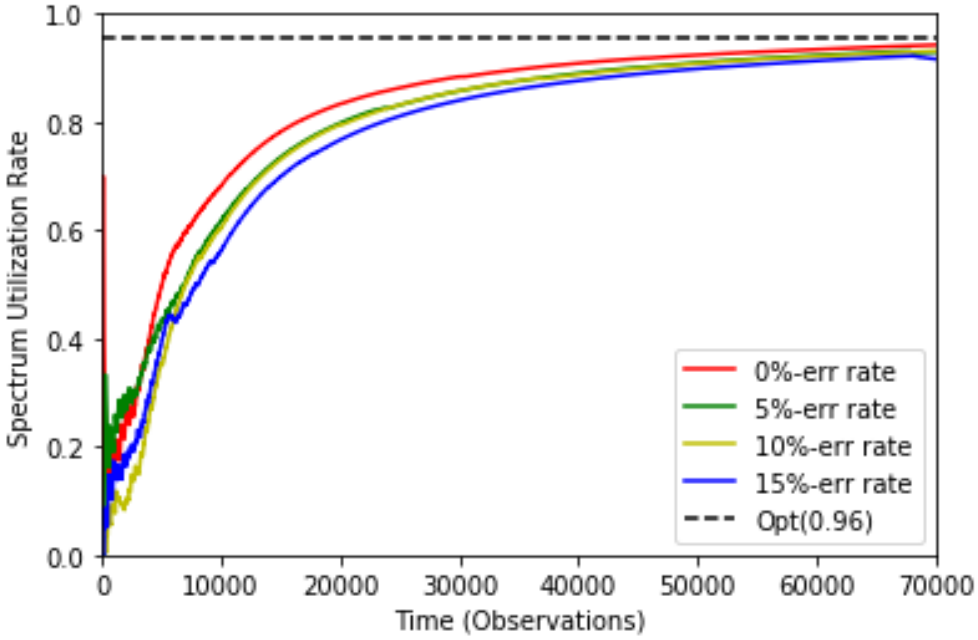
Based on the simulation results of the previous subsection, we conclude that DRQN nodes can learn the different nodes' spectral occupancy patterns/behaviors and even reach near-optimal performance in different complex environments when observing half or even less of the channels at each time step. We now examine the robustness (tolerance to observation error) of the DRQN node in complex environments with different observation patterns. In reality, because of noise and/or other interference, nodes will not receive perfect observations from the environments. Thus, it is important to understand if imperfect observations of the environment have a significant impact on the learning nodes.

Here, we still use the same two complex environments which we described previously, but now there are three different observation error rates: 5%, 10%, and 15%. Different error rates mean that the learning nodes have corresponding error probability when sensing each channel at each time step. The 'error' means that the result of sensing is the opposite of reality. It should be noted that the errors in sensing are assumed to be i.i.d.

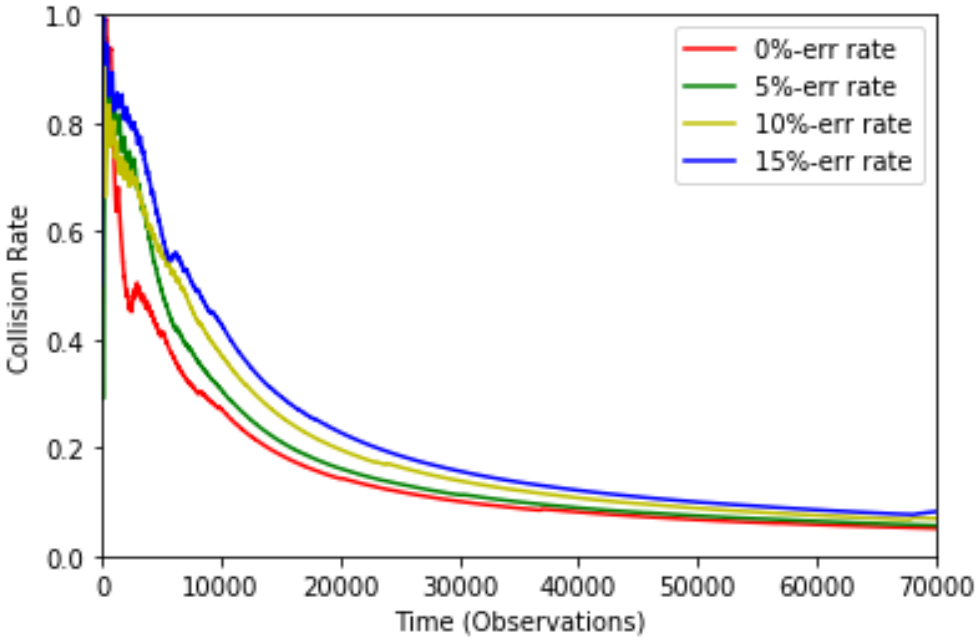
From Fig. 2.6 and Fig. 2.7, by comparing different error rates, it can be seen that, as one would expect, sensing errors have a negative impact on the learning rate of the DRQN node, but the effect is limited. Furthermore, the DRQN node's performance is robust and can ultimately achieve near-optimal performance. The reason why the DRQN node has such robust performance is as follows. First, due to the relatively low error rate, the probability of multiple channel sensing errors at one step and consecutive sensing errors on the same channel is low. Therefore, even if the data has a few errors, it will not affect the extraction and utilization of the previous observation. Second, simultaneously, since experience replay is also exploited by randomly selecting the memory tuples from the memory box, the probability of learning erroneous information is also reduced, thereby improving the robustness of the DRQN node. However, it also should be noted that the robustness of the DRQN node also has a limit. According to simulations with higher error rates, when the error rate is greater than 20%, the performance becomes worse and more unstable (the lowest spectrum utilization rate is about 70%, and the highest collision rate is about 20%). Moreover, when the error rate reaches 35%, the performance is no longer acceptable (the lowest full utilization rate is about 50%, and the highest result shown in collision rate is about 50%).
Based on the results of section 4.3.2 and section 2.3.4, we have found that the proposed DRQN node can learn the patterns of different complex environments with different observation patterns and give correct predictions. Meanwhile, the proposed DRQN node is also robust to imperfect feedback.

2.3.5 Multi-Band and Multi-Agent Operation

In this subsection, we still consider two observation rates (50% and 33.3%). Meanwhile, in order to further examine the performance, our proposed communication environment will

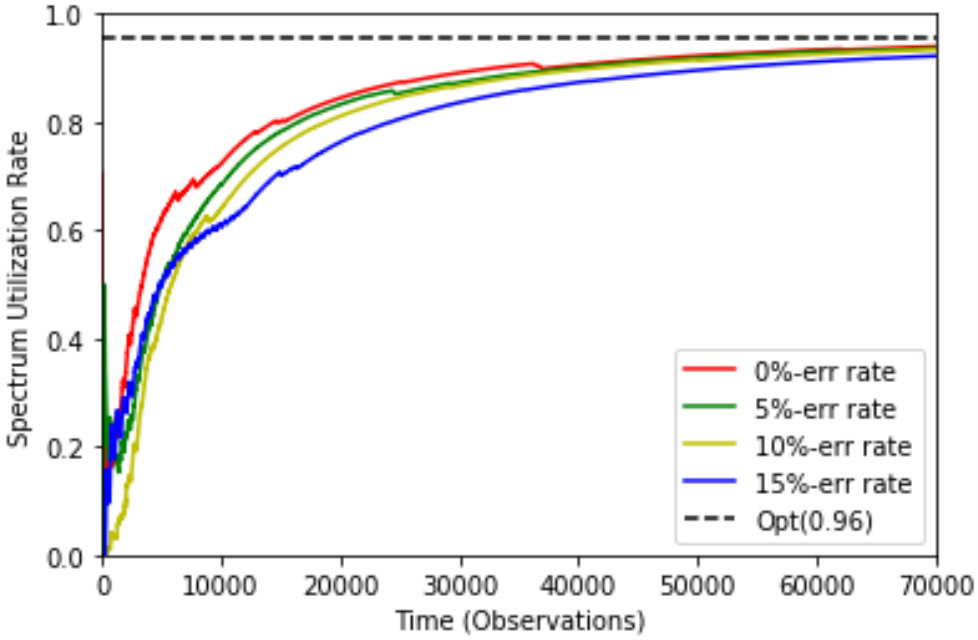


(a) Spectrum Utilization Rate in Complex Environment #1

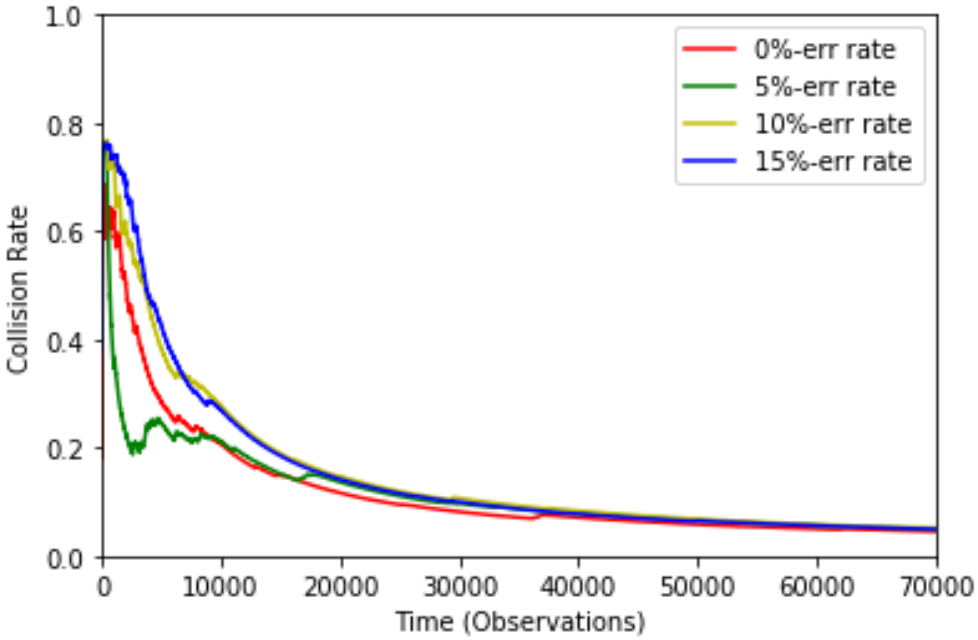


(b) Collision Rate in Complex Environment #1

Figure 2.6: Performance (Spectrum Utilization Rate and Collision Rate) in the presence of Imperfect Observation in Complex Environment #1

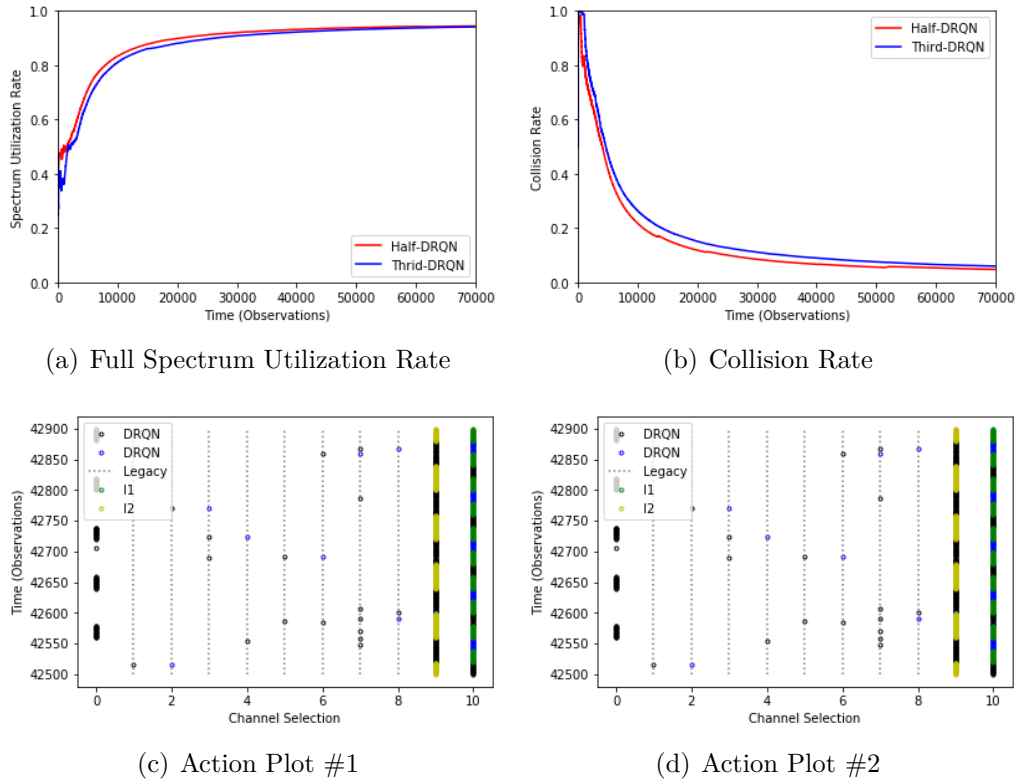


(a) Spectrum Utilization Rate in Complex Environment #2



(b) Collision Rate in Complex Environment #2

Figure 2.7: Performance (Spectrum Utilization Rate and Collision Rate) in the presence of Imperfect Observation in Complex Environment #2



(a) Full Spectrum Utilization Rate (b) Collision Rate

(c) Action Plot #1 (d) Action Plot #2

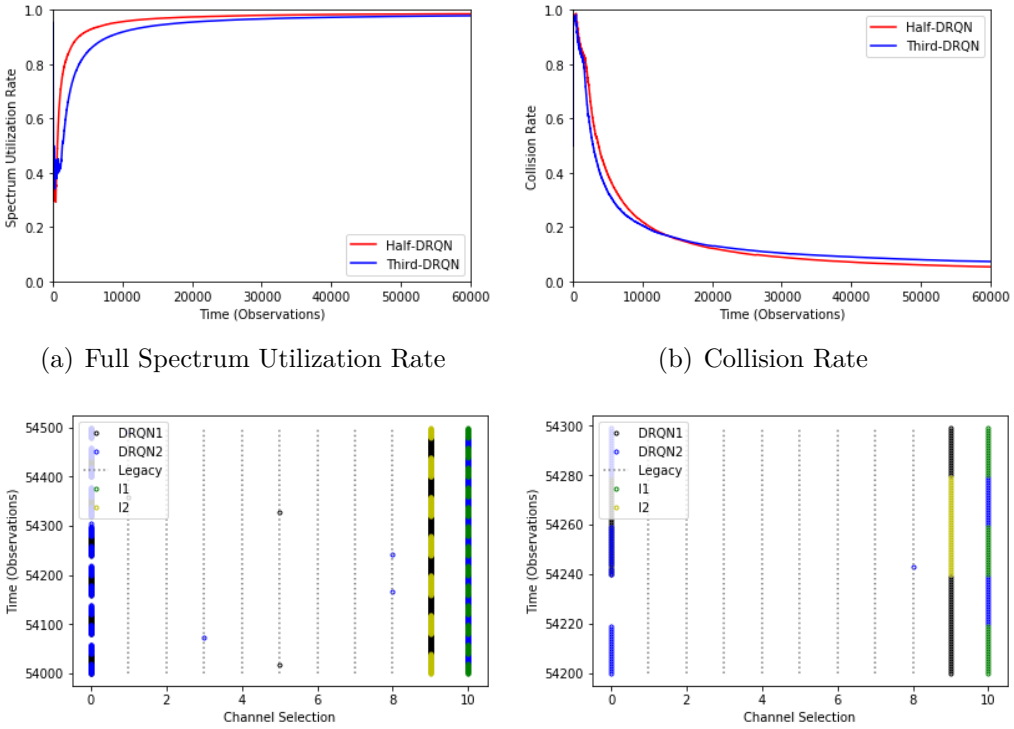
Figure 2.8: Spectrum Utilization Rate and Collision Rate Performance with Multi-Band Transmission

also be more extreme. In Complex Environment #3, there are eight legacy nodes (#1 to #8) and two intermittent nodes (#9 and #10). The intermittent nodes have open times which overlap. Thus, node #9 occupies the channel for 40 steps and is idle for 40 steps, and node #10 occupies the channel for 20 steps and is idle for 20 steps. Because only the channels which are occupied by intermittent nodes have idle periods, this scenario also shows that the DRQN node is able to learn different intermittent nodes at the same time. Further, we expand the capabilities of the DRQN node by allowing it to transmit on two bands simultaneously (i.e., expand the action space). Additionally, to examine the interaction between learning nodes, we simulate two simultaneous DRQN nodes.

It should be clarified that because the DRQN node can use more than one available chan-

nel when the system has multiple open channels, the reward of successfully transmitting on multiple bands is higher than just using one available channel (here, to encourage DRQN node to use multiple available channels, when DRQN node successfully transmits on multiple bands, we provide three times the reward). Also, the successful access rate doesn't provide a good measure of whether the DRQN node uses all the available channels. Thus, in this simulation, we plot the spectrum utilization rate, the collision, and the DRQN's actions. Obviously, since the DRQN has an infinite buffer, the optimal full spectrum utilization rate of the system should be 100%. From Fig 2.8, it can be seen that, first of all, the performance of the DRQN node is good. The utilization of the channel (about 96%) can even approach the optimal performance, but from the collision rate, we see that the DRQN still has some risky behavior. The reason is that in order to make the channel utilization as high as possible, the reward for transmitting in multiple bands is much higher than the reward for a collision. Thus, the DRQN node may try more times on multiple bands to obtain 'high profit', thus increasing the collision rate. However, we believe that we can adjust/trade collisions for spectrum occupancy using the reward structure to improve performance. Meanwhile, since the behavior of the DRQN node is more complicated than the previous simulation (i.e., it can select two available channels), we examine its performance with fewer observations (one-third). By comparison, it can be found that the one-third observation mode's performance has a small loss compared to the one-half observation. Although the number of observed channels is reduced, since the DRQN always has the previous information, its performance can ultimately reach the same level.

In the second simulation, it should be clarified that the two DRQN nodes have the same sensing result (observation) and take their actions simultaneously without communicating with each other. From Fig 2.9, since each node has a 1% exploration rate, the minimum collision rate is about 2%. The total collision rate in our system is about 4%. The results show that our DRQN nodes can learn to avoid each other and exploit open channels in this



(a) Full Spectrum Utilization Rate

(b) Collision Rate

(c) Action Plot #1

(d) Action Plot #2

Figure 2.9: Spectrum Utilization Rate and Collision Rate Performance in Multi-Agent Scenario

multi-agent environment. At the same time, by analyzing the action plots, it can be seen that although the channel selection results of the DRQN nodes in different runs are different, they will ultimately focus on one channel and avoid using the other and avoid the ping-pong effect[3]. By analyzing the structure of the DRQN, we find that the results are mainly due to the following: first, although these DRQN nodes have the same structure, the initial weights of each layer are different (due to random initialization). Second, when using the stochastic gradient descent algorithm, the 'descent routes' selected are also different, which leads to the different final behaviors. Third, the sequential data they use for learning is also different due to random exploration and the randomness of memory replay.

2.3.6 Dynamic Environment

In this subsection, the DRQN node will be tested in a somewhat more realistic environment which is dynamic while maintaining a one half observation pattern. In general, the communication environment is not always static. Therefore, in order to determine whether a DRQN node can adapt to a dynamic environment, the test environment should be dynamic. Thus, the proposed environment has five sub-environments. At the beginning, the sub-environment #1 has one hidden node (#1) and one legacy nodes (#2), three hopping nodes with a dynamic channel occupancy pattern of #5#6#7 $\rightarrow$ #6#7#8 $\rightarrow$ #7#8#5 $\rightarrow$ #8#5#6 $\rightarrow$ #5#6#7 and one intermittent node: #9 (node occupies the channel for 250 consecutive steps and is idle for 10 consecutive steps). After 40000 time steps, the environment will add two nodes (sub-environment #2): one legacy nodes (#3) and one intermittent node (#10, occupies the channel for 250 consecutive steps and is idle for 5 consecutive steps). After 40000 time steps, the environment will add one intermittent node (#4, occupies the channel

[3]The 'ping-pong effect' is the situation where nodes frequently exchange their selections

for 250 consecutive steps and is idle for 12 consecutive steps). At this moment, the environment would be as same as Complex Environment #1 (sub-environment #3). After 40000 steps, the environment changes to Complex Environment #2 (referred to as sub-environment #4). After another 40000 time steps, the environment returns to the sub-environment #2. In the end, the environment will return to its initial state.

According to our previous simulation results, a DRQN node reaches its best performance after approximately 50000 steps. Because more training time (i.e., more data) makes the DQRN's performance better and more robust, in order to make the simulation more general, we do not give the DRQN node enough training steps to converge. It should be noted that DRQN node doesn't know the environmental switch time and the final exploration rate is about 5%.

Because there is more than one available channel during the entire dynamic environment,

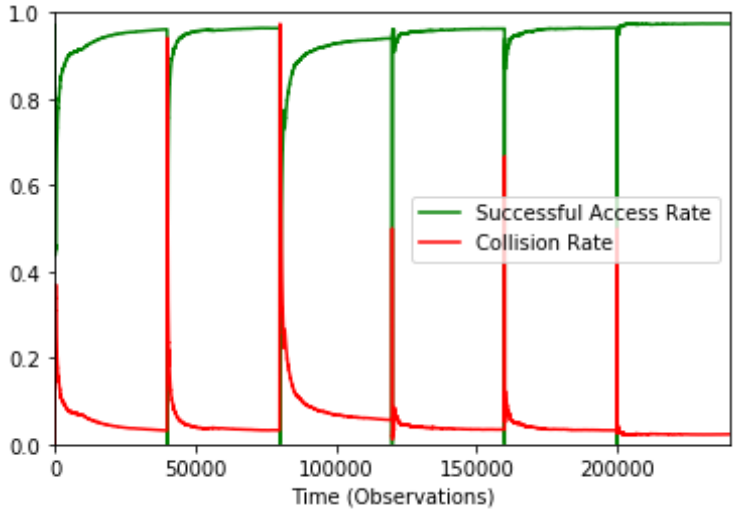


Figure 2.10: Successful Access Rate and Collision Rate of DRQN in Dynamic Environment

the successful access rate and collision rate will be plotted. Futher, when the environment changes, these two metrics are reset. From Fig 3.9, it can be seen that: first, the DRQN node can quickly adapt to the dynamic environment. Meanwhile, comparing the convergence speed in different environments, after the sub-environment #1, the convergence speed

in following environments is accelerated. In particular, comparing the sub-environments #3 and #4 changes with the results of Complex Environment #1 and #2 (subsection 4.3.2), the convergence speed is increased by a factor of two or even higher. This not only illustrates that the DRQN node can deal with a dynamic environment, but also illustrates that pre-training can improve the learning node's performance even if pre-training occurs in a simple environment.

2.3.7 Cache State and Reward Setting

In this work, we assume a fixed-size cache in our learning nodes. In order to utilize the cache state to improve the throughput, the reward feedback from the environment should correspond to the state of the cache. Instead of directly inputting the specific value of cache space, the 'state' of cache is a more general approach. Thus, we should define the cache state in advance. It should be clear that different definitions of the cache state may result in different performances, and it is an issue we will investigate further. Here, we provide our cache state definition in Table 4.1.

Different cache states have different rewards. Here, because different specific reward values would influence the performance, we just provide a general reward setting rule along with what we used in the paper (see Table 4.2).

When the cache state is 0, there are no packets in the cache. Thus, at this moment, 'wait' is the best choice, and any other action is wrong. When the cache state is 1, the cache still has enough space to store packets. Thus, the 'wait' action is neither good nor bad, and its reward is small or no reward. Meanwhile, because our learning nodes can transmit on multiple bands, the reward of successful transmission and penalty of a collision should also vary (Reward and Penalty in the table). When the cache state is 2, compared to 1,

Table 2.1: Cache State Definition

STATE	**0**	**1**	**2**
	Empty	**(0,25%size]**	**(25%size,50%size]**
STATE	**3**	**4**	**5**
	(50%size,75%size]	**(75%size,100%size)**	**Full**

Table 2.2: Reward Setting

	Action		
State	**Wait**	**Success**	**Failure**
0	**Medium Reward(+50)**	**small Penalty2(-20)**	**small Penalty2(-20)**
1	**Small(+1) or No reward**	**Reward**	**Penalty**
2	**No reward or small penalty1(-1)**	**Reward**	**Penalty**
3	**medium penalty1(-20)**	**Reward**	**medium Penalty3**
4	**medium penalty2(-40)**	**Reward**	**medium Penalty4**
5	**high penalty(-100)**	**Reward**	**small Penalty3**

although the cache space is sufficient, we do not want to 'wait' often. Therefore, 'wait' has no reward, even a minor penalty. When the cache state is 3, the cache space does not have much room for future packets. Thus, the packets stored in the cache need to be transmitted. Thus, 'wait' will have a medium penalty1. Further, because we want our learning node to transmit as often as possible, the penalty of a collision is reduced (medium Penalty3). The cache space is nearly full when the cache state is 4. Although it may have space for packets, we do not want to 'wait'. Thus, 'wait' has a higher medium penalty (medium Penalty2). Meanwhile, we encourage the learning nodes to take an aggressive approach, which means the penalty of a collision will be a lower medium penalty (medium Penalty4). When the cache state is 5, the cache is full. If no packet can be transmitted, it will be discarded, and the packet drop (loss) rate will increase. Thus, 'wait' is unacceptable and should incur a high penalty. Because of the full state, a more aggressive strategy should be taken. Thus, the penalty for collision is small (small Penalty3). The reward will be discussed more in the following section.

Table 2.3: Distribution of stochastic nodes

Prob	**0.4**	**0.3**	**0.2**	**0.1**
idle	**3**	**4**	**5**	**6**
transmission	**6**	**8**	**10**	**12**

2.3.8 Performance in Stochastic Environment (Cache-based)

In the simulation, we will test four different learning nodes and one kind of reactive node in a stochastic environment. In the environment, there are five available channels, two of them (#1 and #2) are occupied by fixed nodes. The rest of the channels (#3, #4, and #5) are occupied by stochastic nodes. The stochastic node has a random combination of idle period and transmission period, and these two periods are i.i.d. The distribution of the two periods is shown in Table2.3:

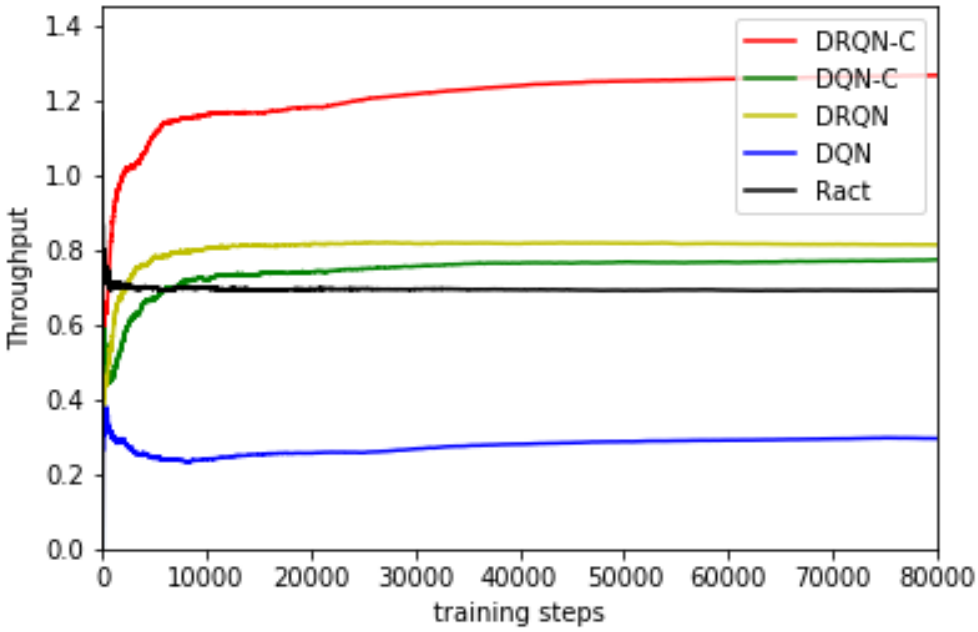
Here, fixed and stochastic nodes are PUs. Thus the learning nodes must learn to use the idle periods of the stochastic nodes. The stochastic node combines idle and working periods based on real protocols (WIFI as an example). In WIFI, the standard size range of the packet is 256-2346 bytes. Here, the learning nodes are SUs, and they can only use the idle period of PUs that prioritize determining the size of the transmitted packet. Thus, the assumption that the learning node is restricted to transmitting a small fixed-size packet whenever of transmit is reasonable (Note: in this paper, the successful transmission duration of an SU is a one-time step). Due to the range of packet sizes, the ratio of learning nodes' and stochastic nodes' times should be from 1 to 0.1. In addition, too long or too short a length of the PU's idle duration (used by SU) obviates the need for learning nodes. This is because the predictive behavior has a high possibility of the collision on the boundary, so learning nodes cannot learn for brief periods. Meanwhile, very long idle times will make the performance unrepresentative. Therefore, based on these two considerations, we set the distribution of the stochastic node in Table 3.1.

In order to illustrate the performance of the learning nodes, we also include one kind of reactive node. The reactive node takes action based only on the currently sensed channels state. For example, when the current channel is occupied, the reactive node will 'wait' until the following sensing interval. Conversely, if the current channel is sensed to be idle, the reactive node will choose to transmit on the channel. Thus, it can be seen that the reactive node's collision and wait only occur at the boundary of a channel state change.

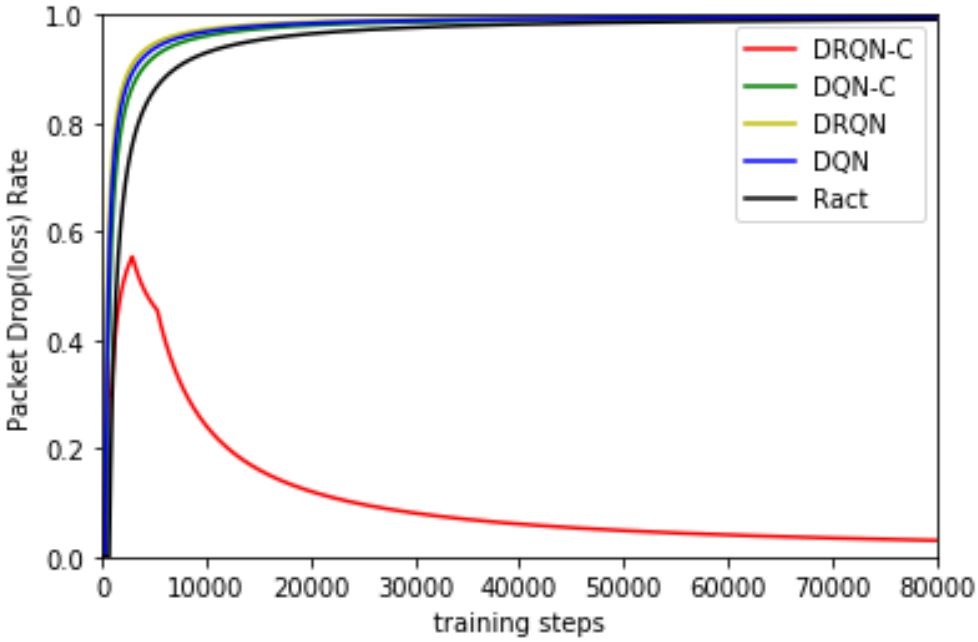
It should be noted that all SU nodes can use channels at once. Thus, based on the environment, the reward corresponds to the number of channels the node successfully uses at one time. In the paper, the reward per channel is 100, and the penalty per channel is -80 (e.g. if a node transmits on three channels but one of them experiences a collision, the reward is $200 - 80 = 120$). Further, due to the cache state, there are three different penalties of collision (medium3, meduim4, and small3). The only difference among these three levels is the weights: medium3 is 75% of the normal penalty, medium4 is 50% of the normal penalty, and small3 is 20%. According to the environment, the optimal throughput of SU is approximately 1.42 $\left(1/\left((1/3)^3 + C_3^2(1/3)^2(2/3) + C_3^1(1/3)^1(2/3)^2\right)\right)$.

First of all, it should be clear that all learning nodes have the same partial observation pattern, which is observing the channel state at an even time step but taking actions at every time step. Meanwhile, the reactive node utilizes full observations. In Fig.2.11, we compare two main performance metrics: throughput and packet drop (loss) rate[1]. It can be observed that, under a 50% observation pattern, 1.) Compared to the same learning structure, inputting the cache state (DRQN-C and DQN-C) improves the performance; 2.) Different learning structures have different performances. Comparing the DRQN structure with the DQN structure, under the same available information, the DRQN structure performs better than the DQN structure; 3.) Even if the DRQN's structure node doesn't use the cache state,

[1]It should be noted that collisions don't necessarily result in a lost packet. A packet is lost only if a packet arrives and the cache is full



(a) Throughput



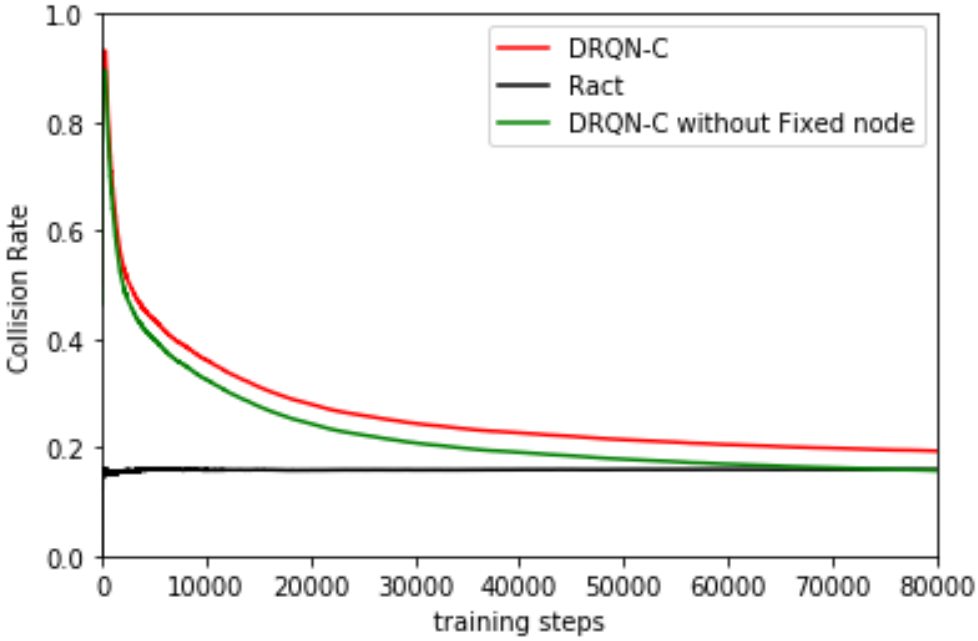
(b) Packet Drop(Loss) Rate

Figure 2.11: Throughput and Packet Drop Rate Performance of all Secondary Nodes (Cache-based)

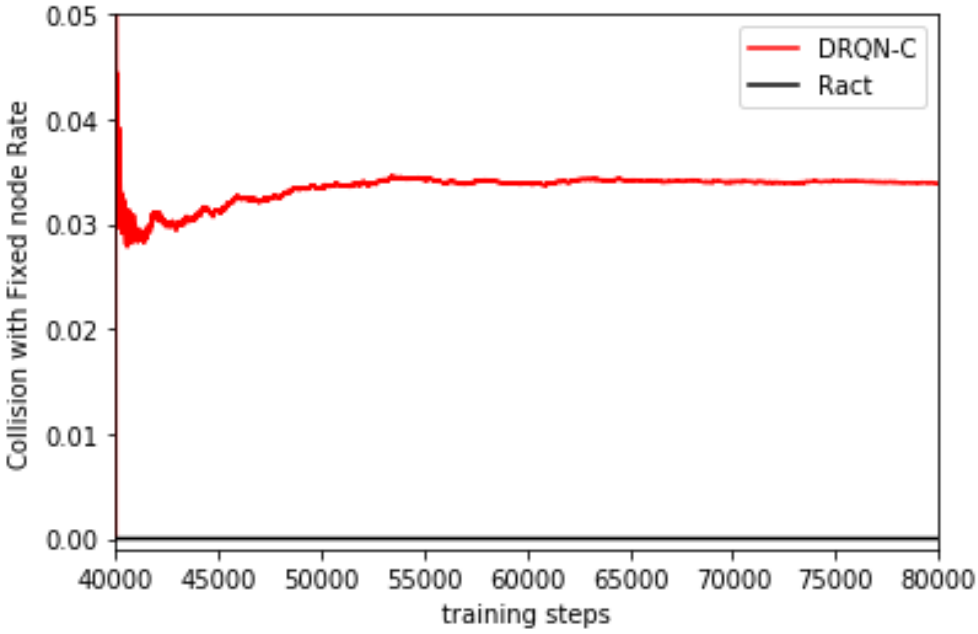
the performance is still better than the two types of DQN nodes and the reactive node. The reason is 1.) Although the four learning nodes share the same reward structure, the DRQN-C and DQN-C can learn the relationship between the various rewards and the cache state. Thus, they perform better than the same structure without providing the cache state (To some degree, the various reward seems to be 'random' to the node without cache state information); 2.) As our previous work proved, the DRQN's structure can utilize the previous information and make a good prediction of the future channel state. The DQN'structure can not. 3.) A reactive node always loses some performance at the 'boundary' of a channel state change. However, the learning node can learn the pattern of the PUs. Thus, it can avoid some collisions and decrease 'useless waiting' to improve performance. To summarize, the DRQN-C is the best-performing structure for both throughput and packet drop rate. Thus, we will only compare DRQN-C with the reactive node in the following plots.

In Fig.2.12, we provide the collision rate plots. Here, because the reactive node's collisions only occur at the channel state changes from 'idle' to 'occupied', its collision rate is a lower bound without prediction. Comparing the overall collision rate, the DRQN-C node is a little higher than the reactive node. However, because of the 5% random actions, our node has some collision with the fixed node (about 3.5%). Thus, in other words, if we only count the collisions with the stochastic nodes, our DRQN-C performance is the same as or even better than the reactive node. It can be explained that the DRQN-C can learn the pattern of stochastic nodes although it is unable to avoid collisions completely due to the random behavior of the PU (Examining the actions plot, DRQN-C may wait or even avoid taking actions when it meets the 'boundary'.). The reactive node always the collisions at boundary.

Finally, because we have a fixed size cache, we should examine the cache state as well as the delay per packet. From Fig.2.13, it can be seen that, except for the DRQN-C, all of the nodes need a infinite sized cache. Thus, these nodes' packet drop rate is almost 1. Meanwhile, due to the finite cache size, the delay plots show that the other four kinds of

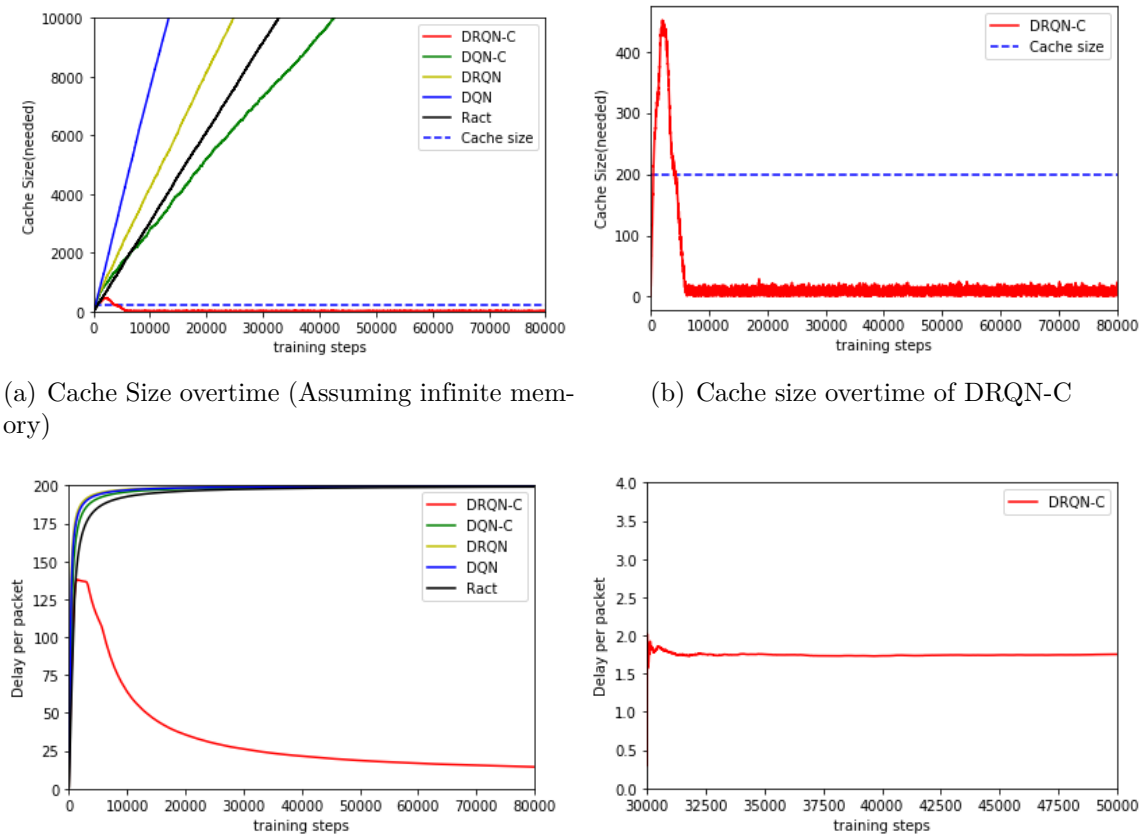


(a) Overall Average Collision Rate



(b) Collision Rate of Secondary Nodes with Fixed Primary Node

Figure 2.12: Collision Performance of the Proposed DRQN-C Node as compared to an Ideal Sense and Avoid (Reactionary) Secondary Node



(a) Cache Size overtime (Assuming infinite memory)

(b) Cache size overtime of DRQN-C

(c) Average Delay per Packet for all approaches

(d) Average Packet Delay of DRQN-C (after training)

Figure 2.13: Cache Occupancy and Average Packet Delay for All Approaches (Cache-based)

nodes approach are worst cases. The DRQN-C node, however, can maintain a finite delay by learning the impact of cache state.

2.4 Conclusion

In this chapter, we have considered a general and complex dynamic spectrum access problem where the system has several PUs dynamically accessing multiple channels and SUs which have no prior knowledge of the PUs' behaviors and base spectrum access decisions on partial observations of the spectrum. We have applied a DRQN approach to find the *optimal access policy* via online learning. In an environment with stochastic nodes, we have also examined the myopic policy which is the optimal access policy when nodes follow a Markov chain model and the statistical information is known. Moreover, we have considered cache-enabled DRQN-based secondary DSA nodes where the system has several primary nodes dynamically accessing multiple channels. Through simulations, we have shown that, although the DRQN is relatively slow, it is able to achieve nearly the same optimal performance even without any prior knowledge. Even in more extreme stochastic environments, the DRQN-based SU can still achieve excellent performance. Further, we examined the DRQN in the presence of nodes which follow fixed-pattern channel switching. Through simulations, we have shown the DRQN can achieve nearly optimal performance without any prior knowledge under different observation patterns, while a myopic policy and a DQN can not. We also have shown that the DRQN is more robust in the presence of imperfect environment observations than the DQN. Third, by simulating learning nodes in specific environments, we have shown that the proposed DRQN can use multi-band transmission, accommodate multiple agents, and its performance is nearly optimal. Furthermore, we have shown that our DRQN can handle

dynamic environments without any artificial adjustment or prior knowledge. Meanwhile, the results also suggest that pre-training in a DRQN can improve its performance. Finally, in the cache-based stochastic environment, we have shown that the DRQN node not only the spectrum occupancy based on partial observation, but also the benefit of not allowing the cache size to grow.

However, in the current work, we only learn the access policy under a fixed observation pattern. This may not be the best use of limited sensing resources and can even result in a waste of resources. Thus, we would like to extend our approach to predict the most effective sequence channels to sense (for a specific number of channels). Thus, in the next chapter, we will extend the DRQN to encompass sensing.

Chapter 3

Optimizing Sensing Strategy for Improving Dynamic Spectrum Access

As shown in the previous chapter, using DRQNs, we can find excellent sub-optimal or even near-optimal access (transmission) policies under the fixed sensing/observation patterns. However, better performance can be achieved in many scenarios. For example, with the development of wireless communication, more and more small wearable devices need to be connected to the network. As a small secondary user device, it is difficult to expend a large amount of energy on channel sensing due to limited power. Therefore, in this chapter, we focus on a more intelligent observation/sensing approach.

SUs sense one or more frequency bands (channels) to assess whether a band is idle, and if one is, switch to and use it for an appropriate period. Thus, in order to enable DSA, SUs should be capable of sensing (the ability to observe and locate spectrum opportunities), identifying when to transmit (the ability to analyze and characterize these opportunities), and tolerating errors (the ability to tolerate imperfect observations). In order to overcome the defects cause by traditional approaches, the combination of sensing with artificial intelligence has been shown to be a promising alternative to simplistic approaches. Obviously, the more information the SUs obtain, the higher the probability of selecting the optimal actions. However, due to resource limitations, the nodes may not be able to continuously monitor all frequency bands in practice. Further, many learning approaches cannot solve

the resulting Partially-Observable Markov Decision Process (POMDP). Thus, in our previous works [64, 65], by combining a recurrent neural network [55] with a DQN, we have shown that a DRQN-based technique for making access decisions can make near-optimal decisions in when making partial observations of the spectrum. However, in order to scan/sense the entire spectrum during each decision-making cycle, the agents in [64] [65] adopt a fixed observation pattern. While effective, the fixed observation pattern is in many cases a waste of resources. For example, if the first half of the channels are occupied a high percentage of the time, the agents likely should not spend their limited resource to sense them. As a result, a more flexible approach is desired which can more efficiently use its resources, thus saving energy.

In this chapter, we focus on dynamic spectrum access in which the SUs acquire knowledge

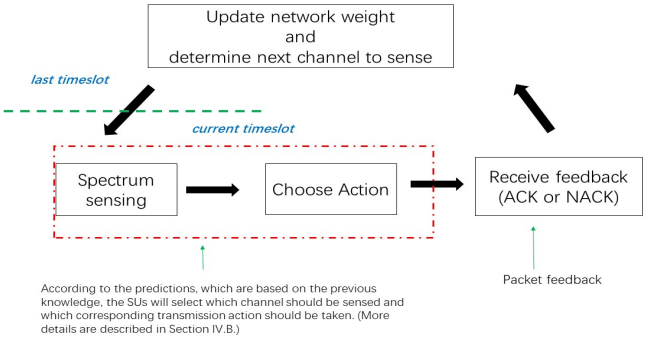


Figure 3.1: The structure of one timeslot

from the environment to determine *which channels should be sensed in addition to which actions should be taken* at a given time for spectrum access. In contrast to most work applying machine learning for DSA, our goal is for the agent to learn the optimal (or near-optimal) sensing pattern (and associated transmission policy) rather than the optimal access action. Further, we apply a new time-slot structure which is 'sense-decision-action' (see Fig.3.1) and is commonly used in studies aimed at examining the Age of Information (AoI) [80][81]. In general, this is an uncoordinated multi-channel access problem with discrete channels using

partial observations. In our assumed model, after sensing, each channel has three possible states (occupied, idle, and unknown). Each PU transmits according to some predetermined schedule[1] and is assumed to avoid collisions with other PU's. Further, the PUs ignore the activity of the SUs. Thus, each secondary node must sense the spectrum (one or more channels) and choose a channel to transmit or delay its transmission until later. If the secondary node chooses to transmit and the selected channel remains idle, the transmission is successful (i.e., we assume that interference dominates performance). Otherwise, there is a collision, and the transmission fails. The secondary node is assumed to detect the sensed channel's state using an unspecified spectrum sensing approach (e.g., energy detection), although not necessarily perfectly. Additionally, in order to examine (and improve) delay performance, we include a packet cache. In practice, it is unacceptable to discard (i.e., drop) many packets if no channel is available. Therefore, it is necessary to study our partial sensing approach's impact on packet delay. It should be noted that the channels are independent and thus do not provide information about neighboring channels. Further, because the channels can change between the observation and when the agent takes action, observations do not provide absolute guidance about the correct action. This is where learning is beneficial.

3.1 Related Work

In the previous chapter (also in [64, 65]), we examined an online learning approach that combines a recurrent neural network (RNN) with a DQN. The RNN is specifically included to solve the POMDP problem (i.e., assuming partial observations each timeslot) while not requiring learning nodes to communicate with a central node or with each other. In that

[1] Note that the schedule is unknown and thus appears random to the SUs.

chapter, we showed that a DRQN can successfully handle partial observations. However, one weakness of that approach is the use of a fixed observation/sensing pattern of a subset of the channels. The main defect is the inefficient use of the limited sensing resources through a fixed channel sensing pattern. In other related work [82, 83] use echo state networks (ESNs) [84] to create a DEQN-based scheme to learn a channel sensing pattern. However, while a DEQN converges quickly, it generally loses some performance relative to a standard DQN, as we will show. Due to the use of sparse connections in EQN and DEQN (the input layer's unit(s) are sparsely connected to neurons in the reservoir units, and the weights in the reservoir units are randomly initialized), the MEQN uses resources less effectively.

In the current chapter, we utilize a DRQN-based approach to learn the optimal (or near-optimal) sensing policy that chooses one or two channels at each sensing interval to obtain (imperfect) observations of a small fraction of the available channels. We will show that we can obtain near-optimal performance with substantially less channel sensing by directing sensing rather than simply transmitting. Additionally, we show that we can exploit multiple open channels and control packet delay and drop rate by modifying the reward structure based on the number of packets in the cache and adding cache information to the DRQN input. More specifically, in this chapter, we provide the following contributions to the state of the art:

1. We develop and characterize a DSA approach using a DRQN to drive sensing decisions that substantially reduce the number of channels that must be sensed each timeslot with very little loss in performance.

2. We show how relative reward settings can be used to control the behavior of the DSA approach making it more or less aggressive (trading throughput for collision rate).

3. We develop and demonstrate an approach that includes cache information so that packet delay and drop rate can be controlled.

4. We demonstrate that the approach allows for multiple learning agents to be deployed in the same environment without introducing convergence issues or instability. This is sometimes termed the "multi-agent problem". We also demonstrate that the approach reacts well to changes in the environment and converges quickly, especially if the environment has been seen previously.

3.2 System Model

3.2.1 Problem Formulation

Before providing the problem formulation, we first clarify the definition of the learning agent (i.e., the secondary node or SU). An intelligent agent in reinforcement learning [41] interacts independently with the environment over a sequence of discrete times to accomplish a specific task. At the time t, the agent senses a particular channel and obtains an observation $o_t \in O$ of the environment, where O is the set of possible observations (spectrum occupancy in our case). Upon obtaining the observation, the agent takes a corresponding transmission action $a_t \in A_{o_t}$, where A_{o_t} is the set of possible transmission actions (transmission in a particular channel or waiting, i.e., not transmitting, in our case) based on observation o_t. As a result of the transmission action, the agent receives a reward r_{t+1}. More specifically, the reward is based on the successful/unsuccessful reception of the transmitted packets or waiting. Consequently, the agent will make a new sensing decision and the environment provides a new observation o_{t+1} at time $t+1$. It can be understood that the agent (SU) must make two decisions each time slot: (a) which channel to sense and (b) whether to transmit on a

given channel or to wait. The two decisions will be based on the Q-values estimated by the DRQN.

Thus, we consider a dynamic spectrum access scenario with a mesh network of $2N$ primary radio nodes, including N transmitters and N receivers. The N primary transmitters choose to transmit on one of K channels based on their operating mode and traffic load at each time step. We also assume no duplex collisions (i.e., collisions between up/down transmissions of the same communications link) occur. Collisions only occur when two different transmitter nodes attempt to occupy the same channel simultaneously. Here, we should mention that the simulation assumes that the SU receiver is aware that the channel used by the transmitter has changed, although he does not investigate the specific technique. A simple (but somewhat impractical) approach is for the receiver to listen on all channels simultaneously simply. A more reasonable approach is to have a narrowband control channel that is used to signal the data channel to be used for subsequent transmissions. We further assume that the secondary agents have limited resources. Therefore, instead of observing all channels during each time slot, the agents must learn an optimal sensing strategy as well as an optimal transmission strategy based on the sensing result (more details are described in Section 3.2.3). The overall process is represented in Fig.3.1. Thus, in this paper, our goal is for the agent(s) to learn near-optimal sensing and transmission strategies based on the assumed timeslot structure and feedback from the environment. Meanwhile, it is assumed that the state (i.e., channel occupancy) can change at each time step. If the node transmits and the corresponding channel is idle in that time step, the transmission succeeds, and the SU receives a positive reward. The result of transmission is assumed to be known using standard ACK/NACK messages. Due to varying channel quality (i.e., temporal fading), observations include sensing errors. The reward for successful transmission can be fixed (based on single-channel transmission) or variable (based on multi-channel transmission). The details of the exact reward structure are described in Section 3.2.4). As mentioned, nodes

Table 3.1: Distribution of Type 1 PU Idle and Transmission Periods

Prob	**0.4**	**0.3**	**0.2**	**0.1**
length of idle	**3**	**4**	**5**	**6**
length of transmission	**6**	**8**	**10**	**12**

can also choose to wait (i.e., not transmit) receiving a small reward (or even a penalty) in the hopes of receiving a higher future reward (by avoiding collisions). We also apply a waiting penalty to avoid waiting consecutive time slots. Note that different reward structures can be used to make nodes exhibit different behaviors (e.g., aggressive, neutral, or conservative). Ultimately, our goal is to maximize the secondary user's throughput while minimizing the impact on the primary user and using as few sensing resources as possible. The latter goal (minimization of sensing resources) is the primary contribution of the paper. It is important to note that when using a DRQN, this is accomplished by maximizing the expected long-term reward, which is the fundamental goal of any Reinforcement Learning technique. This is specifically accomplished by the proper design of the reward function along with the action space. The result is that the agent learns intelligent sensing strategies that maximize resource utilization through the maximization of expected long-term rewards.

3.2.2 Primary User Behavior

In the assumed environment, there are three types of primary user behavior: fixed, Type I stochastic, and Type II stochastic. A fixed node has the simplest behavior and always transmits and occupies a constant channel. The Type I stochastic node has random idle and transmission periods, and these two periods are i.i.d (see Table 3.1). From the table, we can see that the idle/transmission duration can take on four different values, each with different probabilities. The assumption of this stochastic PU is based on real-world protocols

Table 3.2: Types of PUs

Types of PU	Function
Fixed Node	Occupies one fixed channel at all times
Hidden Node	Occupies one fixed channel at all times, but can not be sensed directly
Stochastic Node(Type I)	Occupies one fixed channel, with a probability distribution
Stochastic Node(Type II)	Occupies one fixed channel, with a Markov Transfer Matrix distribution

(e.g., WiFi, which has a typical packet size of 256-2346 bytes. The SUs are agents who attempt to learn how to use the idle periods of PUs. Thus, the learning nodes will generally transmit small fixed-size packets whenever the transmission is possible. Since the learning nodes cannot completely avoid collisions when channels change state, an excessively long or short idle duration would prevent the need for learning nodes since a very short idle period mitigates the efficacy of learning while a very long idle time reduces the benefit of learning. The Type II PU node is described as a two-state Markov chain. This PU type is more general, and the corresponding transition probability of the two-state Markov chain on the *nth* channel is:

$$P_n = \begin{pmatrix} p_{00}^n & p_{01}^n \\ p_{10}^n & p_{11}^n \end{pmatrix} \tag{3.1}$$

where, $p_{ij} = Pr\left(state_j | state_i\right)$. In this work, we assume that the $p_{00} = 0.8$ and $p_{11} = 0.9$. Thus, the probability of the channel being idle is about 0.333 and the probability of the channel being occupied is about 0.667 for both Type I and Type II PUs. Finally, we note that PUs may be "hidden" in that they cannot be sensed by the transmitter although they can cause interference to the receiver. While a simple "sense-and-avoid" process may not be able to handle this situation, machine-learning approaches are found to be effective in detecting such a PU. Overall, here is the Table 3.2 of PUs' type in the paper.

3.2.3 Action Space

First of all, we explain the timeslot structure we used here. As shown in Fig. 3.1, each timeslot is divided into a spectrum sensing period, a data transmission period, and an ACK-/NACK (feedback) period. In research field of AoI (e.g., in [85] [86] [81]), this timeslot structure has been widely applied. The advantage of this structure is that it can typically reduce the collision rate. However, its advantages are based on the assumption that the sensing performance is accurate and that the environment doesn't change between sensing and transmission. But, if there are sensing errors and the environment can change, the SUs must predict which channels should be sensed and what actions should be taken based on learned relationships between current observations, actions, and resulting rewards.

Based on the timeslot structure assumed the agent should determine (a) which channel should be sensed and (b) whether or not to transmit on the corresponding channel based on the observation. We examine two main approaches to solve the problem. Approach 1 uses a DRQN to determine which channel should be sensed and what transmission action to take. Thus, the number of Q-values in this approach is $2K$ since there are K channels and two transmission actions (transmit or wait). Approach 2 uses the DRQN to predict only which channel to sense and uses a simple strategy for transmission (if the channel is occupied, do not transmit, if clear, transmit). Clearly, the first approach is more intelligence. Nevertheless, training two NNs simultaneously is complex due to the use of online learning and may not converge quickly. Thus, we apply the second approach in the chapter. Using Approach 2, our final actions combine two parts: the sensed channel and the corresponding actions (including transmit, wait, and random). For exploration, both the channel to be sensed and the resulting action is chosen randomly. Note that the number of simultaneously sensed channels is a parameter of the SU node. Our initial assumption is that the SU senses a single channel only. However, we also examine the performance of a DRQN-based SU that

can sense two channels simultaneously. In this case, more actions are possible since any combination of the channels could be sensed together.

3.2.4 Reward Structure

In the chapter, we consider two types of reward structures: one that considers the cache size and one that does not take the cache into account. Here, we first introduce the reward structure without considering the cache. Based on the ACK/NACK feedback, the initial reward used is 100 per successful transmission. For a variable rate node that is capable of transmitting on multiple channels simultaneously, the reward is increased for each successful transmission across channels (e.g., successfully transmitting on two channels results in a reward of 200). If the transmission fails, the reward is -50 per channel. Meanwhile, nodes can also choose to wait (i.e., not transmit) to receive a higher future reward. Because we do not want nodes to wait too often, the reward for waiting is set to 5 but decreases by 5 for each consecutive slot where waiting is chosen with a minimum reward of -50.

The reward structure when considering the cache is given in Table 3.4. In order to utilize the cache state to improve throughput, the reward feedback should correspond to the state of the cache (see Table 3.3). We define the cache state 0 as the case where there are no packets in the cache. Thus, in this state, 'wait' is obviously the best choice, and any other action is clearly incorrect. Thus, any other action receives a huge penalty of -100. However, the agent still needs to perform sensing to accumulate knowledge of the environment. Obviously, this action pattern is not a perfect choice. If there is no packet in the cache, the agent could choose 'sleep' to save energy. Nevertheless, it has some influence on the knowledge obtained, and thus, we choose to sense even if there is not a packet awaiting transmission. We will add a 'sleep' mode and the corresponding reward structure in future work. When the cache

state is 1, the cache still has plenty of space to store packets. Thus, the 'wait' action is neither good nor bad, and its reward is small or no reward (initially). However, we do not want continuous waiting even in this cache state. Thus, to prevent continuous waiting, we increase the penalty by -5 for each consecutive wait action. Additionally, we assume in this set of simulations that the learning nodes can transmit simultaneously on multiple bands. Correspondingly, the reward for successful transmission and the penalty for a collision also varies directly based on the number of simultaneous transmissions. When the cache state is 2, although there is still available cache space, we do not want to 'wait' often. Therefore, 'wait' has no reward (or even a small penalty) and decreases to a medium penalty if there is waiting in consecutive time slots. When the cache state is 3, the cache space does not have much room for future packets. Thus, the packets stored in the cache need to be transmitted. Thus, 'wait' has a medium penalty (-20). Further, because we want our learning node to transmit as often as possible, the collision penalty is reduced (-80). When the cache state is 4, the cache space is nearly depleted. Although it may have space for packets, we do not want to 'wait'. Thus, 'wait' has a higher penalty (-50). Meanwhile, we encourage the learning nodes to take an aggressive approach, which results in a lower collision penalty (-40). When the cache state is 5, the cache is full. If no packet can be transmitted, it will be discarded, and the packet drop (loss) rate will increase. Thus, 'wait' is unacceptable and should incur a high penalty. Because of the full state, a more aggressive strategy needs to be taken. Thus, the penalty of collision is small (-40). We need to clarify that the reward setting and cache state we use in the paper is a combination that we think is relatively stable across diverse environments and scenarios. However, we need to point out that different reward settings and cache states may lead to different behavior patterns.

Table 3.3: Cache State Definition

STATE	0	1	2	3	4	5
Cache Occupancy	Empty	(0,25%size]	(25%size,50%size]	(50%size,75%size]	(75%size,100%size)	Full

Table 3.4: Reward Structure when Cache State is Included

	Action & Feedback		
State	Wait (No feedback)	Transmit (ACK)	Transmit (NACK)
0	+50	-100	-100
1	+5 or 0	+100	-100
2	0 or -5	+100	-100
3	-20	+100	-80
4	-50	+100	-60
5	-100	+100	-40

3.3 Simulation Results

This section investigates the performance of the proposed DRQN node via simulation using TensorFlow [78]. Here, we assume that all communication environments have 10 discrete frequency channels. Unlike the DRQN structures in [57, 58, 59, 60], the proposed DRQN structure (shown in Fig.4.1) has 4 hidden layers: three fully connected layers (each layer containing 30, 50 and 25 units respectively) and one RNN (40 hidden cells) layer. Note that fully connected layers are on either side of the single RNN layer. The ReLU (Rectified Linear Unit) function is used as the activation function and is applied to the fully connected layers. A mini-batch of experience-tuples are randomly selected from 1000 prior experiences for the computation of the loss function [40] when updating the weights of the network. Moreover, the Adam algorithm [79] is used for updating the weights. Finally, an adaptive ε greedy algorithm [40] (initially set to 0.1 and is decreased by 0.00005 every time step until reaching 0.05) is used to trade exploration and exploitation.

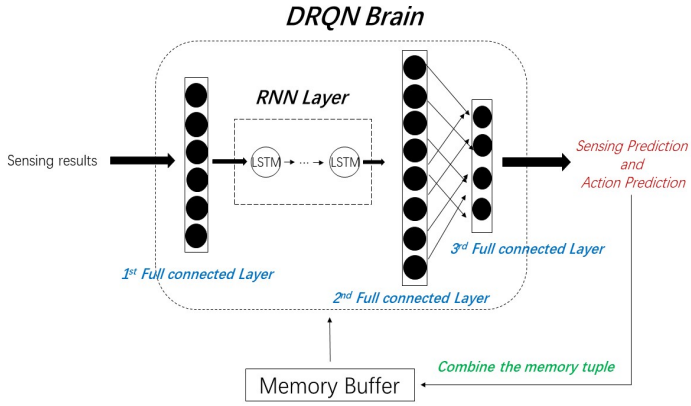


Figure 3.2: The proposed DRQN structure

3.3.1 Types of Nodes

In the communication environment, there are several types of nodes assumed:

- Fixed node (PU): Occupies one fixed channel at all times.
- Hidden node (PU): Occupies one fixed channel at all times, but can not be sensed directly (e.g., the channel occupied by the hidden node appears to be in the "idle" state when sensed by the SU).
- Stochastic node (PU): Occupies one fixed channel, but transmits based on a specified probability distribution. In this work, we assume there are two types of stochastic nodes labeled Type I and Type II based on the specifics of the stochastic behavior as described in Section 3.2.2.
- Reactive node (SU): This is a DSA node that senses all channels and makes a simple transmission decision. Because a simple reactive node would always collide with a hidden node, the reactive node initially utilizes a channel occupancy estimation process. During the first 2000 steps, the reactive node estimates the probability of success for each channel based on full observations of all channels. If there is no available channel,

the node waits. Otherwise, it randomly selects one channel on which to transmit. According to the feedback from environment, it updates the probability of success for that channel. After the estimation process is over (i.e., after 2000 transmissions), the reactive node chooses the best channel to sense based on the success probabilities and transmits if it is idle. It then continues to update the channel success probabilities.

- Myopic node (SU): This is a learning node employing a Myopic policy [25] able to observe all channels simultaneously ('full observation'), i.e., it is assumed to sense all channels each transmission interval (examined for comparison purposes).

- DQN node (SU): This is a earning node using standard DQN [77] which is also able to fully observe all channels each transmission interval (examined for comparison purposes).

- MEQN node (SU): This is a learning node using an MEQN structure with 1-layer and 32 units [82] (examined for comparison purposes).

- Fixed pattern DRQN node (SU): This is a learning node using a DRQN structure for making transmission decisions with a fixed half-sensing pattern, i.e., it observes an alternating half of the channels each interval [64] (examined for comparison purposes).

- Flexible pattern DRQN node (SU): The proposed learning node with an enhanced sensing approach (details are described in Section 4.3.1). Note that there are a few variations of this structure examined. The first structure chooses a single channel to observe/sense each transmission time interval ('1-channel'). The second structure chooses two channels to observe each transmission time interval ('2-channel'). In both structures the DRQN makes both the sensing decision (which channel(s) to sense) and the transmission decision (whether or not to transmit based on the sensing result). The third structure senses a single channel as determined by the DRQN, but uses a simple

transmission strategy ('simple') as opposed to allowing the DRQN to also determine the transmission decision.

It should be emphasized that fixed nodes and stochastic nodes are PUs. Meanwhile, the myopic node, DQN node, MEQN node and DRQN node are SUs that can use ACK/NAK information to determine whether or not their transmissions are successful. The flexible DRQN is the proposed learning approach, whereas the others are examined for comparison purposes.

Here, we should mention that the simulation assumes that the SU receiver is aware that the channel used by the transmitter has changed, although it does not investigate the specific technique. A simple (but somewhat impractical) approach is for the receiver to listen on all channels simultaneously simply. A more reasonable approach is to have a narrowband control channel that is used to signal the data channel to be used for subsequent transmissions.

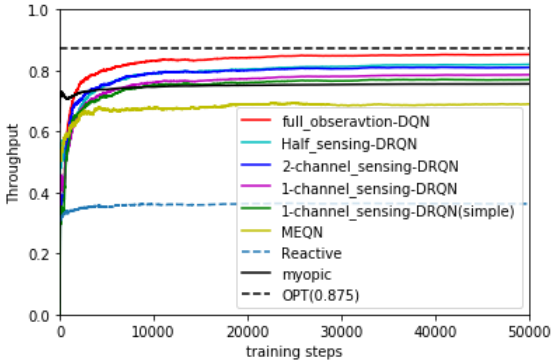
3.3.2 Basic Performance Tests

In the first set of simulations, we conducted three main simulation-based tests: (1) a general stochastic environment test, (2) a robustness test, and (3) tests to examine the relationship between reward settings and node behavior. Through these simulations, we found that in the general stochastic environments, the proposed DRQN node can learn the behavior of the PUs and make sensing and transmission decisions which result in low collision rates and near-optimal throughput while sensing significantly fewer channels than previously proposed approaches. Additionally, we found that through proper adjustment of the reward settings, the behavior of the DRQN node can be modified to make it more and less aggressive, resulting in higher or lower throughput but with an associated higher or lower collision rate.

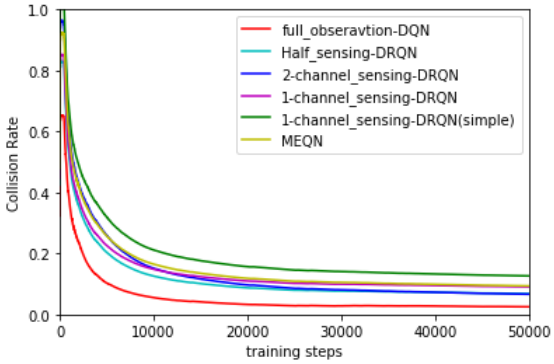
In order to illustrate the performance of the proposed approach relative to previously proposed techniques, we compare our proposed flexible sensing DRQN node (both 1-channel and 2-channel versions) with several previously proposed techniques listed above. We also examine a "simple" variant of our 1-channel sensing flexible DRQN node. This node uses a simple action policy for transmission. Specifically, the node has the same proposed NN structure but has fewer actions available to the NN since the NN makes only the sensing decision. The transmission decision uses a straightforward strategy: with high probability (0.95) use the sensed channel information to determine transmission on the sensed channel ('occupied': wait; 'idle': transmit), and with low probability (0.05) take a random transmission action. Further, the simple action policy applies the same waiting penalty structure to avoid continuous waiting. Here, the full-observation pattern DQN node is (near) optimal and serves as an upper bound on the performance. Additionally, because some stochastic channels do not follow a Markov transfer matrix, the myopic node in the environments will count the first 2000 steps to estimate the transfer matrix using full observation of the ten channels at each time step.

In the first environment, which is examined (Fig 3.3), there are two hidden nodes (operating in bands #1 and #5) and three fixed nodes (bands #2 to #4). Further, the environment also has two Type I stochastic nodes (bands #6 and #7) and three Type II stochastic nodes (bands #8 through #10). Thus, in this system, the probability that all channels are occupied is about 13% ($(2/3)^5 \approx 13\%$). To make our environment more realistic, we assume the feedback of the environment (i.e., the observation) is imperfect (i.e., the actual channel occupancy is not equal to the sensing result). The assumed error rate is 5%. However, observations of the channel(s) occupied by the hidden node always indicate an idle channel, regardless of the true situation.

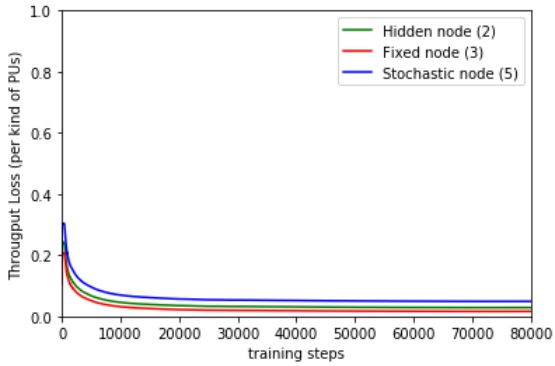
Based on the channel idle rate of the overall system, the optimal throughput (successful transmissions per time slot) is about 0.875 ($1 - (2/3)^5 \approx 0.875$). From Fig 3.3, we find



(a) Throughput in General Environment



(b) Collision Rate in General Environment



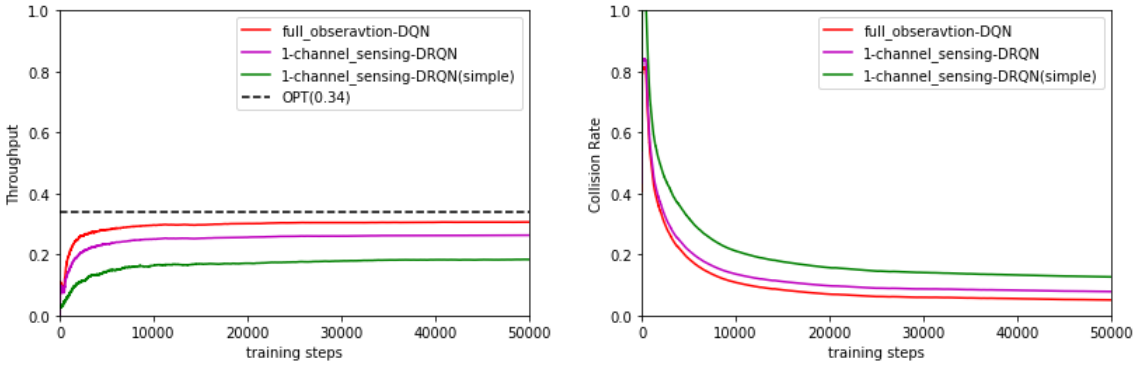
(c) Throughput Loss of different kinds of PUs due to Collisions with Learning Agent

Figure 3.3: Performance in General Environment

that the full-observation DRQN can nearly reach this optimal throughput, although it must sense all ten channels during each observation interval to do so. Further, it can be seen that the myopic node, despite having full observations of all ten channels, suffers in terms of performance. The myopic policy is found to be insufficient for this complex environment. The reason for the sub-optimal performance is two-fold. First, the myopic algorithm needs to know the specific Markov transfer matrices, which must be estimated. Second, even if the estimation process is accurate, some scenarios, such as the Type I stochastic nodes, do not satisfy the assumptions made by the myopic node. Thus, the myopic algorithm does not have universal applicability. The learning nodes have a clear advantage in this respect.

Further, compared to the fixed sensing nodes, the flexible sensing nodes exhibit a small performance loss. This is because the flexible pattern node always selects the most valuable one or two channels to sense. In other words, the flexible pattern nodes save substantial sensing resources while suffering a relatively small performance loss. This would appear to be an acceptable tradeoff. Meanwhile, the flexible DRQN node's collision rate (8%) is slightly higher than the full observation DQN. Here, it should be noted that most of the collisions come from random selection (i.e., from exploration to avoid sub-optimal performance), while some collisions also result from imperfect feedback. In the learning stage of single agents, the SU will affect the performance of the PU to a certain extent. Since, in any given timeslot, the agent can only collide with one PU, we show the plot of throughput Loss of different kinds of PUs due to collision with Learning Agent.

As we mentioned before, the collision rate also depends on the reward structure, which we will examine in more detail shortly. Here, We set the successful transmission reward to be much higher than the collision penalty to improve channel utilization efficiency. Meanwhile, the penalty chosen for waiting is also somewhat 'aggressive'. Thus, the DRQN node will take some risky actions, which causes the collision rate to increase. However, from another viewpoint, the DRQN node only sacrifices a 5% increase in collision rate while reducing



(a) Throughput in Extreme Environment

(b) Collision Rate in Extreme Stochastic Environment

Figure 3.4: Performance in Extreme Stochastic Environment

the sensing by a factor of ten, which is undoubtedly a worthwhile tradeoff. Comparing the performance of the two kinds of proposed DRQN nodes, we find that both 1-channel and 2-channel nodes eventually reach nearly the same performance, albeit with different convergence speeds. (The throughput of the 2-channel sensing DRQN node after convergence is higher by 0.008). However, the 1-channel sensing flexible DRQN node obviously saves resources. Thus, it is another tradeoff that the agent can select different numbers of channels for observation. However, it is interesting to note that the DRQN node can learn and predict the patterns of the various nodes while only observing a single channel if chosen intelligently.

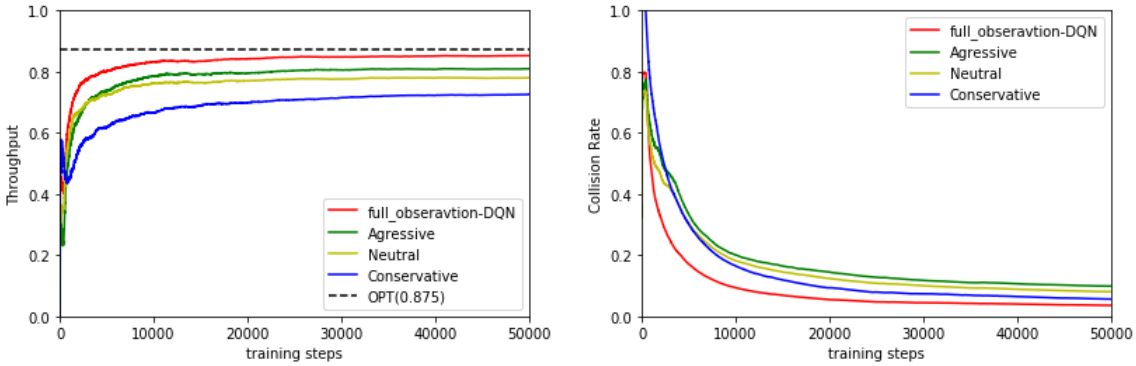
Comparing the previously proposed MEQN [82] with our proposed DRQN nodes, the MEQN's convergence speed is faster as one might expect (the MEQN converges in about 90000 steps, while the DRQN converges at about 170000 steps). However, the performance of the MEQN is worse than the DRQN after convergence. The reason is that MEQN or EQN is designed for fast training, but our DRQN with LSTM is designed for near-optimal performance. Thus, in the presence of specific convergence requirements, the MEQN may be the better choice, while our proposed approach concentrates more on the final perfor-

mance. Additionally, examining Fig.3.3, we find that the flexible 1-channel sensing DRQN node with a simple action policy can achieve almost the same performance level as the full approach. This may suggest that using a simple action strategy can achieve the same level of performance with a simpler overall structure since the transmission decision need not be part of learning but instead left to a simple policy after sensing.

In order to examine this issue further, we compared three learning nodes (1-channel sensing flexible DRQN node, full-observation pattern DQN node, and 1-channel sensing flexible DRQN node with simple action policy) in an extreme environment: Channels #1 to #5 contain five hidden nodes, while channels #6 to #9) contain four fixed nodes, and channel #10 contains a Type II stochastic node. Thus, in this system, the probability that all channels are simultaneously occupied is about 66%.

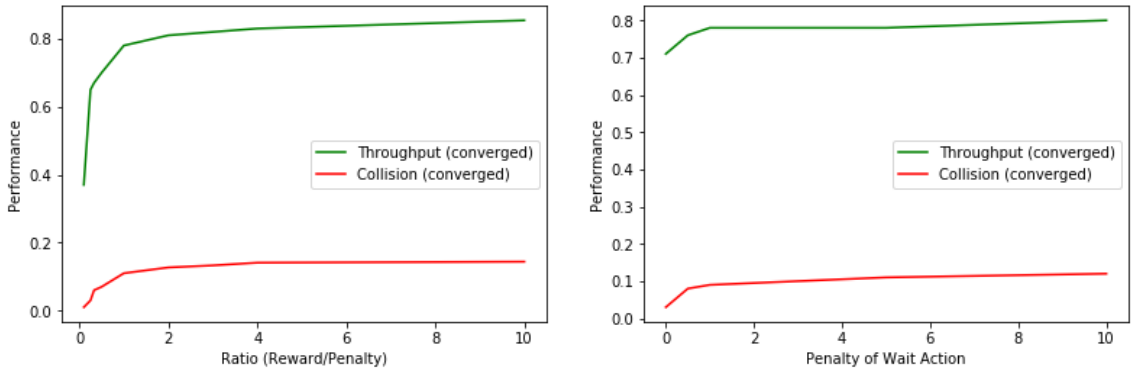
In Fig.3.4, we compare the performance of these three learning nodes in this environment. We can see that our proposed node's performance is a little worse than the full observation (albeit with only 10% of the overall channel observations required). Additionally, we find that the 1-channel DRQN that uses the NN to make transmission decisions is much better than the simple strategy. In fact, the more sophisticated approach provides not only 50% higher throughput but also nearly half the collision rate. This is because the more sophisticated learning node predicts which channel should be sensed and the corresponding behavior. Due to imperfect feedback from the channel and the fact that some nodes are hidden, if there is only one available channel at a time, the simple strategy is not able to effectively avoid collisions. However, the more sophisticated 1-channel DRQN node can avoid collisions to a greater extent. Meanwhile, the full observation DQN node has an easier learning problem to solve due to the exploitation of a full set of observations. Further, the more channels sensed simultaneously, the higher the probability that the agent can find an open channel with reliable observations.

As we mentioned above, the behavior of the learning nodes is strongly affected by the



(a) Performance of Throughput (b) Performance of Collision Rate

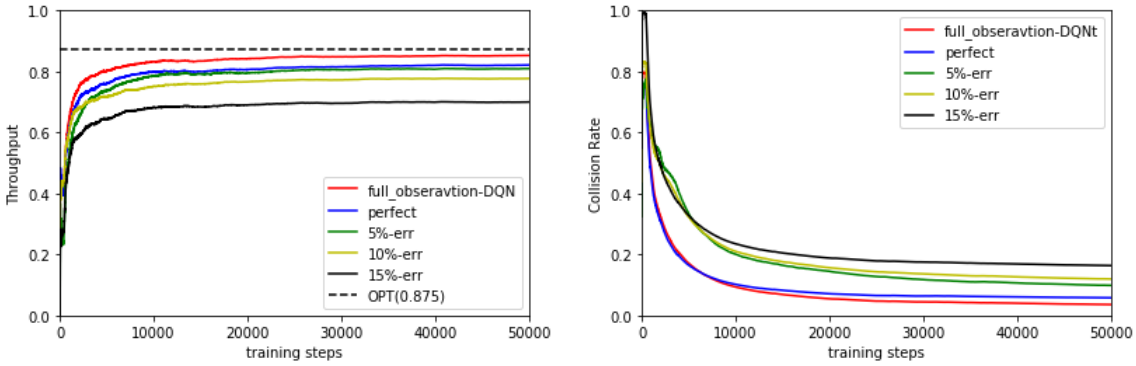
Figure 3.5: Relationship between reward structure and action tendency



(a) Average Throughput and Collision Rate as the Ratio of ACK Reward to NACK Penalty is varied (b) Average Throughput and Collision Rate as the Wait Penalty is varied

Figure 3.6: Relationship between reward structure and action tendency

reward settings. In Fig.3.5, we examine this issue more carefully. In the plots, there are three different reward settings: 1). 'Aggressive' (used in the previous simulations); 2). 'Neutral'; 3) 'Conservative'. Among these three settings, the main difference is the relationship between the penalty and reward. In other words, if the reward (e.g., +100) is much higher than the penalty (e.g., −50) and has a relatively high penalty for waiting (e.g., −20), the agent will take more risky actions (i.e., risk more collisions). Thus, it's behavior is "aggressive". In contrast, if the reward (e.g., +50) is much lower than the penalty (e.g., −100) and has a relatively low penalty (e.g., −5) for waiting, its behavior is "conservative." The 'neutral' is a bit more difficult to choose (This is actually an interesting issue that we plan to investigate more in the future). If we set the reward equal to the penalty, there are many more 'wait' actions than expected. Therefore, in the simulation, we set the reward to be slightly higher than the penalty and set the penalty for waiting to be between 'aggressive' and 'conservative'. To this point, we have not developed theoretical relationships to associate with these behaviors. Instead, these settings are based on experience. In the paper, we use our judgment based on the desired values of throughput, collision rate, and waiting rate. Based on the simulation results shown in Fig.3.5, we find that: 1) An 'aggressive' reward setting improves the spectrum utilization rate (i.e., increases throughput), but at the cost of a higher collision rate; 2). A 'conservative' reward structure provides the lowest collision rate but increases the rate of 'waiting', which can be negative; 3). A 'Neutral' reward structure provides a middle ground, as we would expect. Thus, it is clear that we can adjust the ratio of the reward and penalty to meet the relative performance requirements. Here, we also give more quantitative guidance on choosing the reward structure. The first plot is the final throughput (and collision rate) v.s. the ratio of the reward (for ACK) to the penalty (for NACK). In Fig.3.6 (ratios of 0.1,0.25,0.33,0.5,1,2,3,4,10), We can see that collision rate and throughput both saturate quick after a ratio of 2. Additionally, we also added a plot of the throughput (and collisions) versus the penalty for waiting (using a 1 : 1 ratio for



(a) Performance of Throughput (b) Performance of Collision Rate

Figure 3.7: Performance in imperfect feedback from General Environment

the ACK/NACK reward/penalty.This impact is shown in Fig.3.6. We can see here that the throughput (and, to a lesser extent, the collision rate) is relatively insensitive to the wait penalty.

In practical systems, because of noise and/or fading, nodes will not receive perfect feedback from the environments. Thus, it is meaningful to understand if imperfect feedback from the environment significantly impacts the performance of the learning nodes. In the simulation, to test robustness, we test four levels of feedback error: perfect, 5% error rate, 10% error rate, and 15% error rate. Further, we only test the 1-channel flexible sensing DRQN node here. It should be noted that the errors in the system of the feedback are assumed to be i.i.d.

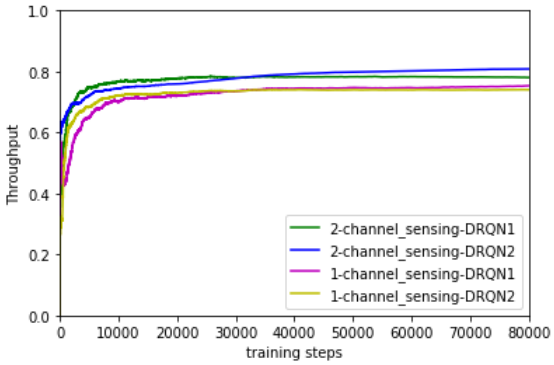
In Fig.3.7, we compare the impact of different error rates using the same parameters as in Fig.3.3. We see that the effect of imperfect feedback is relatively limited for error rates up to 10%. That is, for error rates less than or equal to 10%, the DRQN node's performance is robust and can achieve nearly as good as that of perfect feedback. It is found that even if the observations have a relatively small number of errors, it does not affect the extraction and utilization of knowledge. However, it should be stated that the DRQN node's robustness also has its limits. We see that performance drops significantly at an error rate of 15%.

Additionally, if the error rate exceeds 20%, the prediction performance becomes significantly worse and unstable. One way to improve the performance is by increasing the number of channels sensed simultaneously.

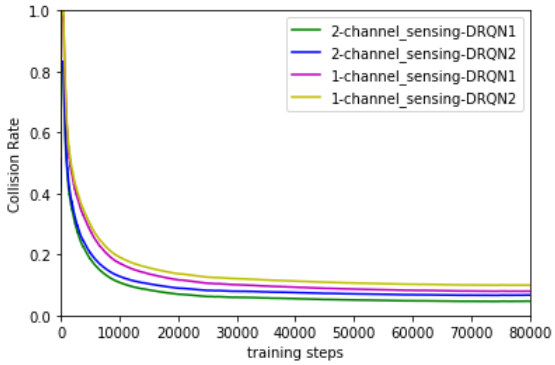
Based on the above simulation results, we conclude that the proposed DRQN nodes can learn the different nodes' spectral occupancy patterns/behaviors and correctly determine which channels should be sensed at the following time slot. Moreover, the performance reaches near-optimal performance in fairly complex environments. Meanwhile, from the results, we also conclude that different reward settings determine the general behavior of the node. Finally, the proposed DRQN nodes have some tolerance in handling imperfect environment feedback. The reason that the proposed DRQN has such performance is due to the unique structure of combining the DNN with an LSTM. The structure allows the agent to learn the environment's dynamic behavior from its history and makes fairly accurate (although not perfect) predictions of future behavior, allowing the node to make accurate sensing (and transmission) decisions.

3.3.3 The Impact of Multiple Learning Nodes

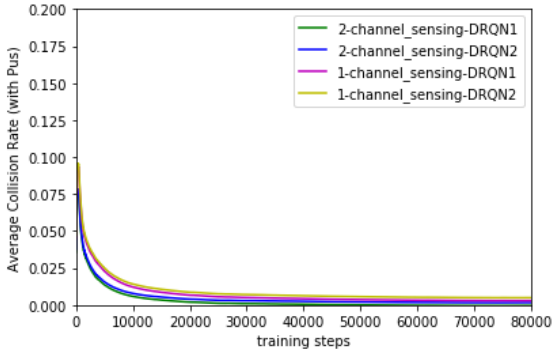
Based on the previous simulations, we conclude that the proposed DRQN node can perform well in the presence of multiple PUs whose behavior can be learned. In this subsection, we examine the performance of multiple coexisting agents. This issue is sometimes referred to as the 'multi-agent problem'. We simplify the problem by examining two coexisting DRQN nodes. The environment we assumed here is as same as what we use in the previous subsection. In Fig.3.8 we present the results from three different simulations. In the first, there are two learning agents, each using one channel for sensing. In the second simulation, we again include two learning agents in the environment, but each chooses two channels to sense



(a) Performance of Throughput



(b) Performance of Collision Rate



(c) Performance of Average Collision Rate (with PUs)

Figure 3.8: Performance of multi-agent

at each time slot. From the figure, we find that having two SUs in the environment does (slightly) impact performance: For two 2-channel SUs, the throughput of each DRQN is reduced by about 3%. For two 1-channel SUs, throughput is reduced by about 9%. The collision rate is unchanged compared to previous single-channel learning node simulations. Meanwhile, in the learning stage of multiple agents, the SU will affect the performance of the PU to a certain extent. However, our results show that an individual PU will experience a collision rate that starts below 10% at the beginning of training, drops quickly below 2% and settles at a value much less than 1% (see Figure 3.8). However, more 'waiting' occurs in the multiple-agents scenario and the overall throughput of the SU's increases. Thus, the results show that our DRQN nodes can learn to avoid each other while exploiting open channels in the environment. At the same time, by analyzing the action plots, it can be found that most of the collisions come from random actions. These random actions stem from exploration and the penalty for waiting too often. The agents have to make more random and risky actions instead of 'stable' actions when a second SU is attempting to occupy open channels. Another thing that is found is that despite the fact that we use an 'aggressive' reward setting, the agents in the multiple-agent scenarios tend to 'wait' more than in the single SU environment. However, we do find that the agents can learn to exist in the same environment without causing instability. By analyzing the structure of DRQN, we find that this fact is mainly due to (1) although these DRQN nodes have the same structure, the initialization of the weights of each layer are different (randomly initialized); (2) the 'routes' they select in the stochastic gradient descent algorithm are also different, leading to different final behaviors; (3) due to the exploration (random actions) rate and the randomness of memory replay, the sequential data they use for learning are also different; (4) compared to the PUs in the environment, the other agent appears more like a type of hidden node. Thus, the agents prefer more' waiting' to avoid collisions with the 'hidden node'.

3.3.4 Dynamic Environment

One issue with the use of learning for dynamic spectrum access is that the communication environment is often not static. In a dynamic environment, the learning agent must continually learn the new behavior of the environment. Therefore, to determine whether the proposed DRQN node can adapt to a dynamic environment, the proposed DRQN node was tested in an environment with changing characteristics. Specifically, we assume that the overall dynamic environment has five sub-environments. Note that all stochastic nodes in this study are assumed to be Type II. Initially, the channels are occupied by two hidden nodes (channels #1 and #5), three fixed node channels (#2 to #4), and five stochastic nodes (channels #6 to #10). We name this sub-environment 'env1'. Then, two of the channels' patterns exchange: channel #5 is occupied by a stochastic node, and channel #10 is occupied by a hidden node. We term this sub-environment 'env2'. Subsequently, four other channels' dynamic patterns change: channels #3 and #4 are occupied by stochastic nodes, but channels #8 and #9 are occupied by fixed nodes ('env3'). After some time, the environment changes back to the second environment ('env2'). Finally, the environment returns to its initial state ('env1').

From the simulation results of the previous section, the DRQN node typically needs more than 40000 steps to reach optimal performance. Thus, in this scenario, we set the duration of each environment to be slightly less than this minimum convergence time. It should be noted that the DRQN node does not know the environmental switching time or that it will switch. However, here we set the initial exploration rate to 10%, and using epsilon-greedy, we reduce the exploration rate to 5%.

Note also that for illustration purposes, in the performance plots, we reset the collision rate and throughput at each change of environment. From Fig 3.9, it can be seen that: first, the DRQN node can quickly adapt to the dynamic environment. Meanwhile, comparing

the convergence speed in different sub-environments, we see that after the first environment (env1), the speed in convergence rate in the following environments is accelerated. Especially comparing the speed of the first sub-environment and the last one, as well as the second sub-environment and fourth one, we find that convergence is much improved when we return to a sub-environment that has been seen before. This issue illustrates that the DRQN node can deal with a dynamic environment and hints that pre-training could improve the learning node's performance.

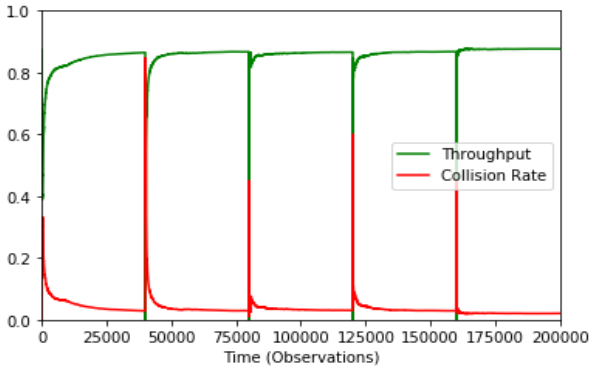


Figure 3.9: Performance in Dynamic Environment

3.3.5 DRQN with Cache Information and Multiple Channel Transmission Capability

In the previous sections, we made two fundamental assumptions: (1) that agents can exploit only a single channel at a time, and (2) since a new packet arrives during each slot, packets are dropped if the agent does not transmit. However, dropping such a large number of packets is unacceptable. Thus, in this section, we will test agents that include (a) a cache (i.e., they can store packets) and (b) the ability to transmit on more than one channel simultaneously. This work assumes a fixed cache size of 200 in our learning nodes. Additionally, to utilize the

cache state as a means to improve the throughput, the reward feedback from the environment is connected to the cache state. However, instead of directly inputting the specific value of cache to the DRQN, we define the 'state' of cache as a more general approach (see Section 3.2.4 and Table 3.3). Clearly, different cache state definitions will lead to slightly different performance, but the following results are representative.

In the testing environment, we assume the feedback of the environment has a 5% error rate, and there are ten available independent channels, two of which (#1 and #5) are occupied by hidden nodes, fixed nodes occupy three channels (#2 to #4), and stochastic nodes occupy five channels (#6 to #10). Here, since the Type II stochastic node is more general, we only simulate this kind of stochastic node. In other words, all stochastic nodes in these simulations are Type II. Further, we will test four different types of learning nodes. In order to illustrate the performance of the proposed learning nodes, we first test a full observation DQN node without a cache limit. We examine this node because it can serve as a near-optimal upper limit. Meanwhile, a DQN with a cache and a DRQN without a cache is also included as a control group.

It should be noted that all of the SU nodes are capable of transmitting on multiple channels simultaneously. Correspondingly, the reward is directly related to the number of channels the node successfully uses at one time. In the section, the reward per channel is 100, and the penalty per channel is -100 (e.g., if a node transmits on three channels but one of them experiences a collision, the reward is $200 - 100 = 200$). Further, due to the cache state, a collision has three different penalties. Specifically, we use three fractions of the penalty: 80% of normal penalty, 60% of normal penalty, and 40% as described in Table 3.4.

Initially, due to the ability to transmit on multiple channels, the assumption is that all learning nodes can select two channels to sense at each time step. Thus, according to the

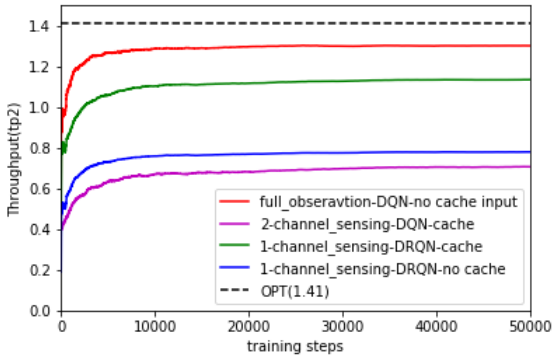


Figure 3.10: The Impact of Cache Input on Throughput Performance

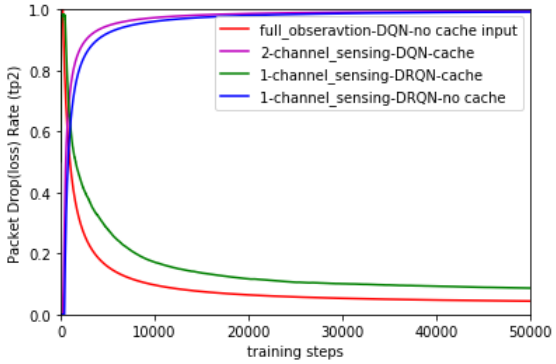


Figure 3.11: The Impact of Cache Input on Packet Drop (loss) Rate Performance

environment, the optimal throughput η_{opt} of a SU is approximately 1.41:

$$\eta \approx (1 \times C_5^1 \times (1/3)^1 \times (2/3)^4 + 2 \times \left(1 - C_5^1 \times (1/3)^1 \times (2/3)^4 - (2/3)^5\right) \approx 1.41) \quad (3.2)$$

. Where C_N^k is the number of combinations of N items taken k at a time. In Fig.3.10 and Fig.3.11, we compare two main performance metrics: throughput and packet drop (loss) rate[2]. From the results in Fig 3.10 and 3.11 we can make the following observations about the performance when delay/cache state is considered: 1.) The full observation DQN has near-optimal performance and thus fully exploits its knowledge, and is only slightly worse

[2]Due to the existence of the cache, a lost packet only occurs if a new packet arrives and the cache is full. Also, a packet is only removed from the cache if it is successfully transmitted.

than optimal because of the non-zero exploration rate; 2.) Under the same neural network structure, inputting the cache state improves the performance (DRQN with cache information outperforms DRQN without cache information); 3.) The DRQN structure outperforms the DQN structure with the same available information; 4.) Even if the DRQN node does not use the cache state, the performance is still better than the DQN with cache information. Stated another way, in general, we find that (a) because the DRQN with cache input can determine the relationship between the various rewards and the cache state, it has better performance than the same structure without using the cache state; (b) The DRQN's structure can utilize previous information and predict which channel should be sensed at the following time step, while the DQN's structure cannot; (c) Although (based on the throughput plots) all of the nodes can learn the behavior of PUs to varying degrees and achieve good throughput, only the DRQN with cache information (and the full-observation DQN) can effectively reduce the packet drop rate. This means that the DRQN with cache information can learn the relationship between the cache state and the dynamic rewards and use that relationship to reduce delay. Without cache information, the nodes only learn to avoid the PUs without effectively reducing the cache size. However, the DRQN with cache information avoids collisions while decreasing 'useless waiting' to improve performance. First, compared to 1-channel sensing DRQN, the 2-channel DQN with cache is very bad. The main reason is that although the 2-channel DQN with cache information can sense multiple channels, its ability to predict which channels to sense isn't outstanding. Our analysis shows that most of its sensing predictions are poor, leading to a high collision rate. Thus the packet drop rate is close to 1, and the throughput performance is lower. Therefore, the DRQN-with-cache is the best-performing structure in terms of both throughput and packet drop rate.

In Fig 3.12, we provide the collision rate plots. Here, because the full observation DQN's collisions mostly occur during the exploring process, its collision rate is a lower bound. Compared to the overall collision rate, the DRQN-cache node is slightly higher than the full

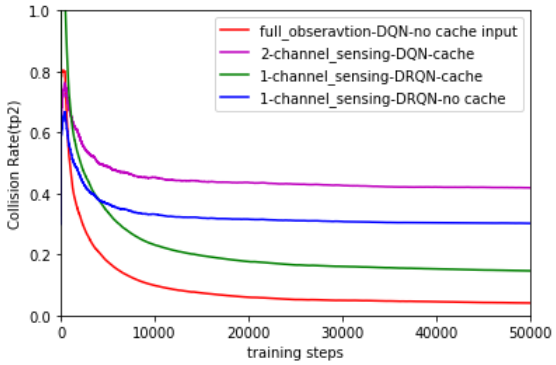


Figure 3.12: The Impact of Cache Input on Collision rate Performance

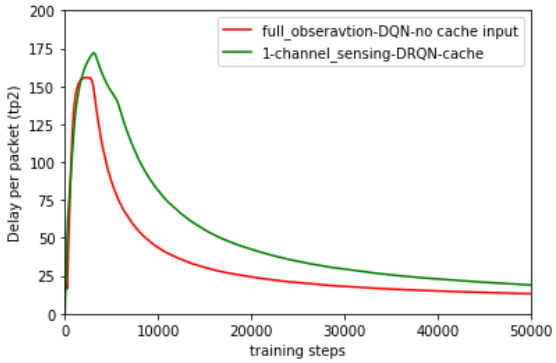


Figure 3.13: The Impact of Cache Input on Delay Performance

observation DQN. However, because of the random actions resulting from the exploration process and the random action which occurs after significant waiting, our node has some collisions with the fixed node (about 3.5%). Thus, if we only count the collisions with the stochastic nodes, our DRQN-cache performance is close to the full observation DQN. It is found that the DRQN-cache can learn the pattern of the stochastic nodes, although it is unable to completely avoid collisions.

Finally, because we have a fixed-size cache, we also examine the cache state as well as the delay per packet. According to Fig 3.11, the DRQN without cache information and the DQN with cache information almost always have a full cache. Thus, the delayed performance of both learning nodes is not meaningful. Therefore, we only plot DQN with full observation and DRQN with cache information here. From Fig 3.13, it can be seen that the DRQN-cache

node can maintain a finite delay by learning the impact of the cache state. The node does well because it learns to transmit multiple packets when possible to reduce the cache size, although it can not fully avoid collisions. Thus, combining an appropriate cache state with a corresponding reward setting can improve the agents' performance and prevent a high packet drop rate while maintaining an acceptable throughput and collision rate.

3.4 Conclusion

In the chapter, we have considered a general and complex dynamic spectrum access problem where the system has several primary nodes dynamically transmitting on specific channels, and secondary nodes exploit the idle periods based on partial observations of the spectrum without any prior knowledge of the primary nodes' behaviors. In the previous chapter, fixed partial observation patterns were used to obtain knowledge of the environment while a DRL agent determined the transmission policy. In order to improve resource utilization efficiency, we used a DRQN to determine the optimal sensing and corresponding action policy via online learning. We showed that the approach achieves near-optimal throughput and collision rate performance with a fraction of the needed observations/sensing. Moreover, we also showed that the proposed DRQN node could handle a more general stochastic environment than a standard myopic policy. Further, the results have shown that the DRQN is robust when facing imperfect environment feedback. Third, we examined the relationship between DRQN's behavior and the reward settings, showing that modifying the relationship between rewards and penalties can make the node more or less aggressive (i.e., it can trade throughput for collisions.) The proposed node behaves well in the presence of multiple learning agents and dynamic environments. Finally, in order to control packet delay and the possibility

of dropped packets, we introduced cache (or buffer) information into the DRQN input. In the simulations, we have shown that the proposed DRQN node learns to exploit multiple available channels in order to reduce the cache size and control packet delay. Further, it was shown that without cache information, attempting to optimize throughput does not control packet delay.

In the chapter, we assumed that the learning nodes always had packets to transmit (i.e., an arrival rate of 1), but, in reality, the number of packets arriving at each time slot follows a specific distribution. Thus, in future work, we plan to examine various packet arrival rates. Second, the assumed timeslot structure we applied in the paper has some flaws (e.g., the requirement of a sensing period every time slot). Thus, in our future work, we will examine a more general timeslot structure. In such a structure, if the node believes there is will be no available channel in the next timeslot, it could choose to 'sleep' rather than sense. Thus, in the future, we will also introduce a sleep mode in the DRQN.

Summarizing the work described in the previous two chapters, we have shown that our proposed DRQN structure can determine a near-optimal access method, no matter the sensing/observation mode used. However, to this point, we have ignored an equally important issue: learning efficiency, or in other words, how fast the performance converges. In practical scenarios, it is unacceptable if the user needs to wait too long to obtain acceptable performance. Thus, in the following chapter, our research will focus on improving learning efficiency while still maintaining strong performance.

Chapter 4

Accelerating Learning Convergence in Dynamic Spectrum Access

indent Through the work presented in the first two chapters, we demonstrated that the proposed DRQN solution could learn policies that achieve near-optimal performance using different (even optimally small) observation patterns. However, we must also consider practical indicators aspects. Among them is learning efficiency or, in other words, convergence speed. For example, in practical scenarios, an approach is untenable if the user-desired needs to wait an unacceptably long time to get the desired performance. Therefore, in this chapter, our research focuses on finding solutions that can improve learning efficiency.

Generally speaking, the goal of DSA is to allow secondary users (SUs) (or unlicensed users) to exploit the idle spectrum resources of the primary user (PU) network in a way that limits the interference caused to the PUs. In order to enable DSA, SUs should be capable of 'S.I.T' - Sensing, Identifying opportunities, and Tolerating errors. Due to the defects of traditional approaches, we applied DRL to DSA. In the chapter, we focus on convergence speed of our progressive approach from the previous two chapters. In contrast to most work that has applied reinforcement learning to DSA, our goal is for the agent to learn a near-optimal access policy with accelerated convergence. As described in our previous work [64, 65], and other works [24, 56, 64]), converge speed is a significant but little-studied topic. Therefore, in this chapter, we mainly focus on how to improve the convergence speed.

As in the previous chapters, we study an uncoordinated multi-channel access problem with discrete channels under partial observations. In our assumed model, each channel has three possible states (occupied, idle, and unknown). Each PU transmits according to some predetermined schedule and is assumed to avoid collisions with other PU's. Meanwhile, the PUs ignore the activity of the SUs. Further, each SU has to sense/observe some channels and choose one channel (or possibly multiple channels) on which to transmit or 'wait' until later. If the SU chooses to transmit and the selected channel(s) remains idle, the transmission is successful (i.e., we assume that interference dominates performance). Otherwise, there is a collision, and the transmission fails. Because of the limited resource, SUs can observe some channels' states with imperfect feedback each timeslot instead of all channels. In work, our provided approaches are mainly from three different perspectives, namely, the learning frequency (parallel learning structures or multiple learning [66]), the inherited learning (general transfer learning [67, 68]), and the new learning approaches (Meta-learning [69, 70]). At the same time, it should be noted that the scenarios in which these three different methods can be used are different. The results demonstrate that three different approaches can accelerate the converge speed and guarantee the sub-optimal (and even optimal) performance under partial observation constrain.

4.1 Related Work

Performing dynamic spectrum access with the help of the DRL algorithm is an efficient method for handling the spectrum management issue for mobile traffic users, and many methods have been implemented in this research direction so far. In our topic, there are mainly two key core issues: partial observation and accelerated converge speed. [56] ex-

amines correlated channels in DSA. However, when channels are correlated, the number of possible states in the system decreases, reducing the learning difficulty. [59] and [60] provide a solution that combines an RNN with a DQN to solve such DSA problems and make some breakthroughs. In [59], the DQN is trained for all users at a single unit (e.g., the cloud) which means the nodes update their DQN weights by communicating with the central unit. However, in our problem, the learning nodes are unable to communicate with each other even through a central unit. Moreover, the assumption of offline training and the number of channels in [59] are very different than what is considered here. [60] combines a reservoir computing algorithm (one kind of RNN) with a DQN to solve POMDPs when learning nodes can't communicate with each other, but it is not a clear spectrum prediction method, and the performance could be improved in terms of both convergence speed and collision rate. [87] classifies distributed-DSA technology and offers an alternative where non-cooperative utility for dynamic-spectrum-access in a distributed way. [88] described communication between multi players with each other and the maximization of the shared utility. A DRL-based spectrum-sensing policy for one user interacting with an external factor as an environment is described in [56]. The multi-user scenario for this case is considerably different in environmental characteristics, channel utility, and algorithm adopted. Meanwhile, in [89, 90, 91], all works focus on how to reach the optimal (sub-optimal) related performance. Although all the mentioned works made some breakthroughs in different issues, few researchers have made progress on the problem of convergence speed. In order to overcome our second core issue, we have to consider some other ideas.

In order to speed up the online learning process, the ideas of 'separation' (divide large tasks into different smaller tasks), 'sharing' (share necessary knowledge while maintaining privacy), and 'combination' (combine multiple small tasks as needed) is essential. In [92, 93, 94], the researchers provide lots of promising ideas, such as federated learning and parallel learning. And in this chapter, we utilize these core ideas to accelerate the online learning process in

DSA under partial observation. Meanwhile, according to [95**?**], Meta-learning is another potential approach to speed up the learning process in the long term viewpoint. Thus, in the following work, we will implement these three main methods to accelerate the online learning converge speed.

4.2 System Model

4.2.1 Problem Formulation

As we stated above, we formulate the problem as follows: our goal is to guarantee the fundamental core issue that maximizes reward (throughput) while minimizing collisions. Meanwhile, another core issue is accelerating converge speed. Throughput is directly related to successful transmissions. Thus, we seek to maximize the number of successful transmissions while minimizing collisions with a faster converge speed. We do this using Deep Recurrent Q-learning, which estimates the Q-value of each state-action pair by DRQN [64]. Further, in order to accelerate the converge speed, the main structure we utilized is DRQN, but at the same time introduced other structures or methods to achieve the goal, namely, parallel learning structures or multiple learning, general transfer learning, and Meta-learning.

Thus, we consider a dynamic spectrum access scenario with a mesh network of $2N$ primary radio nodes, including N transmitters and N receivers. The N primary transmitters choose to transmit on one of K channels based on their action pattern. This work describes the PU node as a two-state Markov chain. This PU type is more general, and the corresponding

transition probability of the two-state Markov chain on the nth channel is:

$$P_n = \begin{pmatrix} p_{00}^n & p_{01}^n \\ p_{10}^n & p_{11}^n \end{pmatrix} \tag{4.1}$$

where, $p_{ij} = Pr\left(state_j | state_i\right)$. In this work, we assume that the $p_{00} = 0.8$ and $p_{11} = 0.9$. Thus, the probability of the channel being idle is about 0.333, and the probability of the channel being occupied is about 0.667. Moreover, we note that PUs may be 'hidden' in that the transmitter cannot observe them, although they can cause interference to the receiver. We also assume no duplex collisions (i.e., collisions between up/down transmissions of the same communications link) occur. Collisions only occur when two different transmitter nodes attempt to occupy the same channel simultaneously. We further assume that the SUs can only observe some channels' states due to the limited resource. Thus, in this paper, we aim for the agent(s) to learn a near-optimal transmission strategy with a faster converge speed. Meanwhile, because of the dynamic environment, it is assumed that the state (i.e., channel occupancy) can change at each time step. If the node transmits and the corresponding channel is idle in that time step, the transmission succeeds, and the SU receives a positive reward. The result of transmission is assumed to be known using standard ACK/NACK messages. Due to varying channel quality (i.e., temporal fading), observations include feedback errors (imperfect observation error). The reward for successful transmission can be fixed (based on single-channel transmission) or variable (based on multi-channel transmission). Meanwhile, SUs can also choose to wait (i.e., not transmit). However, because we do not want SUs to wait too often or miss available opportunities, we introduce a continuous waiting penalty mechanism (receiving a small reward in the hopes of receiving a higher future reward but reduced by waiting consecutive time slots). Ultimately, the goal is to learn a policy that maximizes the expected long-term reward based on the chosen learning structure.

4.2.2 Action Space and Reward Structure

Our main problem is faster convergence while guaranteeing optimal or sub-optimal performance in dynamic spectrum environments. Therefore, to simplify this problem and consider the condition of limited energy, the channel is still observed by scanning (only a part of the channel states are observed each time, but all the channel states will be observed in a few timeslots). At the same time, multiple rates may be required in some cases, so the cache and corresponding cache state (see Table 3.3) will also be introduced. In general, the user's behavior is mainly reflected in the selection of 1 or 2 channels for communication at a timeslot. It should be clear that the reward setting and the cache state used in our paper is what we used in our simulations. Obviously, different reward settings of combinations and definitions of cache state may lead to different behavior patterns. Thus, the relation between reward settings with behavior in quantity is a topic worthy of further study.

First, we introduce the reward structure without considering the cache. Based on the ACK-/NACK feedback, the initial reward is 150 per successful transmission. For a variable rate node that is capable of transmitting on multiple channels simultaneously, the reward is increased for each successful transmission across channels (e.g., successfully transmitting on two channels results in a reward of 300). If the transmission fails, the reward is -50 per channel. Meanwhile, nodes can also choose to wait (i.e., not transmit) to receive a higher future reward. Because we do not want nodes to wait too often, the reward for waiting is set to 5 but decreases by 5 for each consecutive slot where waiting is chosen with a minimum reward of -50.

However, when we talk about the cache, the reward is different and complex. As shown in Table 3.4. Different cache states have corresponding rewards when they receive different

Table 4.1: Cache State Definition

STATE	**0**	**1**	**2**
	Empty	**(0,25%size]**	**(25%size,50%size]**
STATE	**3**	**4**	**5**
	(50%size,75%size]	**(75%size,100%size)**	**Full**

feedback. When the cache state 0 as the case where there are no packets in the cache. Thus, in this state, 'wait' is obviously the best choice, and any other action is clearly incorrect. Thus, any other action receives a penalty of -100. When the cache state is 1, 2, and 3, due to the different cache usage rooms, the reward is adjusted by the demand, which is the release cache state. For example, if the state is 2, although there is still available cache space, we do not want to 'wait' often. Therefore, 'wait' has no reward (or even a small penalty) and decreases to a medium penalty if there is waiting in consecutive time slots. When the cache state is 3, the cache space does not have much room for future packets. Thus, the packets stored in the cache need to be transmitted. Thus, 'wait' has a medium penalty (-20). Further, because we want our learning node to transmit as often as possible, the collision penalty is reduced (-80). The cache space is nearly depleted when the cache state is 4. Although it may have space for packets, we do not want to 'wait'. Thus, 'wait' has a higher penalty (-50). Meanwhile, we encourage the learning nodes to take an aggressive approach, resulting in a lower collision penalty (-40). When the cache state is 5, the cache is full. If no packet can be transmitted, it will be discarded, and the packet drop (loss) rate will increase. Thus, 'wait' is unacceptable and should incur a high penalty. Because of the full state, a more aggressive strategy needs to be taken. Thus, the penalty of collision is small (-40).

Table 4.2: Reward Setting

	Action		
State	Wait	Success	Failure
0	Medium Reward(+50)	Penalty(-100)	Penalty(-100)
1	Small(+1) or No reward	Reward	Penalty
2	No reward or small penalty1(-1)	Reward	Penalty
3	medium penalty1(-20)	Reward	medium Penalty3
4	medium penalty2(-40)	Reward	medium Penalty4
5	high penalty(-100)	Reward	small Penalty3

4.3 Advanced approaches for Accelerating Learning Process and Simulation Results

In this section, we investigates the performance of three advanced approaches for accelerating the DRQN's learning process via simulation using TensorFlow [78]. Here, we assume that all communication environments have 10 discrete frequency channels. And the DRQN structure we used here is shown in Fig.4.1. The normal size of the DRQN has 4 hidden layers: three fully connected layers (each layer containing 30, 50 and 25 units respectively) and one RNN (40 hidden cells) layer. The activation function applied here is ReLU (Rectified Linear Unit) function. Adam algorithm [79] and mini-batch are used for updating the weights. Finally, an adaptive ε greedy algorithm [40] (initially set to 0.1 and is decreased by 0.00005 every time step until reaching 0.05) is used to trade exploration and exploitation.

4.3.1 Parallel Learning (Multiple Learning)

Technically, the parallel learning we used here is one kind of distributed learning scheme that local learners act solely as local data collectors and do not require the global model through any feedback from the aggregator. Parallel learning can scale up the algorithm and accelerate the learning period. During the learning process, one available training set (the

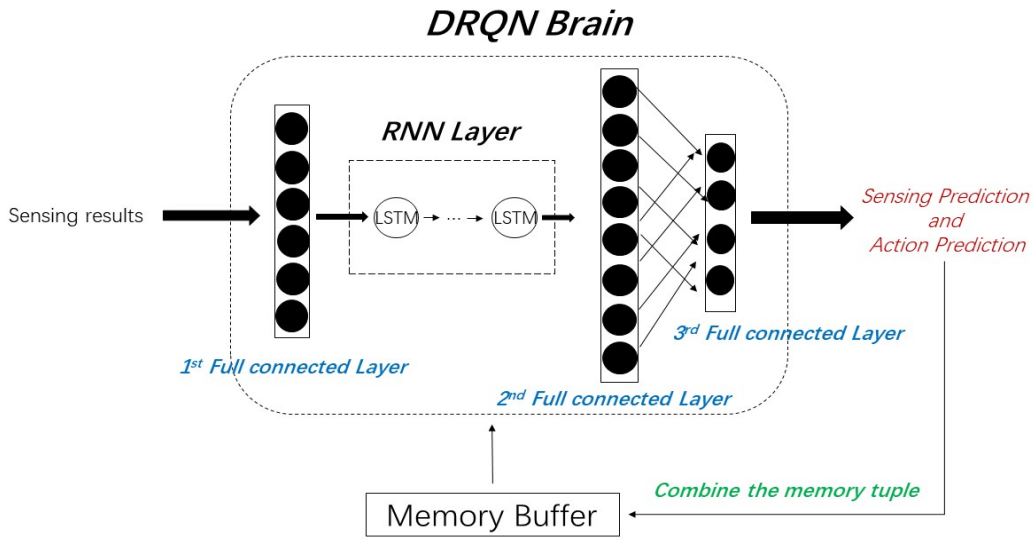


Figure 4.1: The proposed DRQN structure

overall observations in the memory box) at a central parameter server is divided into some subsets of data (the average number of data samples per learning unit is way larger than the number of learning unit participating in the training process) and assigned to a group of learning units (training neural networks). Obviously, the subsets of data have the same underlying distribution. Subsequently, the training process is performed in parallel, and the parameters are fed back to the parameter server. This type of learning is performed in data centers where the worker machines obtain data from shared storage and hence, unlike federated learning, will end up having samples from the same distribution. After parallel learning, they also need to 'combine' the knowledge of the multiple learning units. It is distributed ensemble learning, also known as committee-based learning, is a learning approach in which multiple learners are combined to improve the overall performance. In general, the goal of such learning methods is to learn from a mixture of learning units rather than improving a global model using a naturally distributed dataset through a federation of local learners with communication constraints. The overall process is shown in the Fig.4.2.

To illustrate the performance of parallel learning, we design the parameter combinations

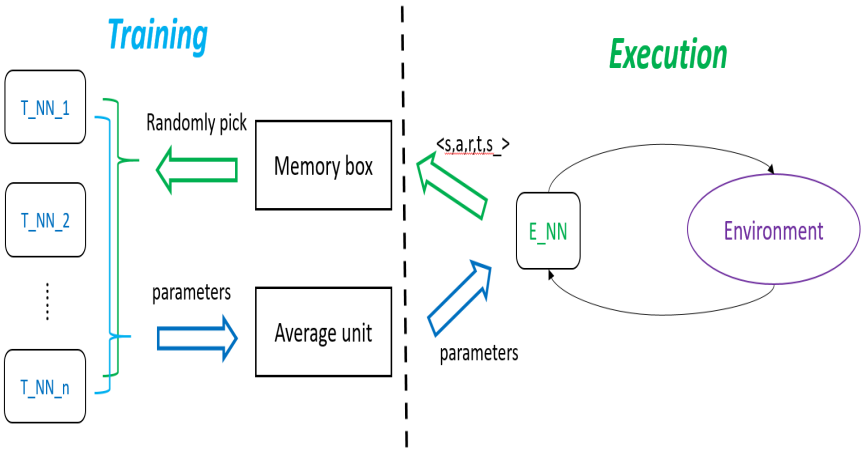
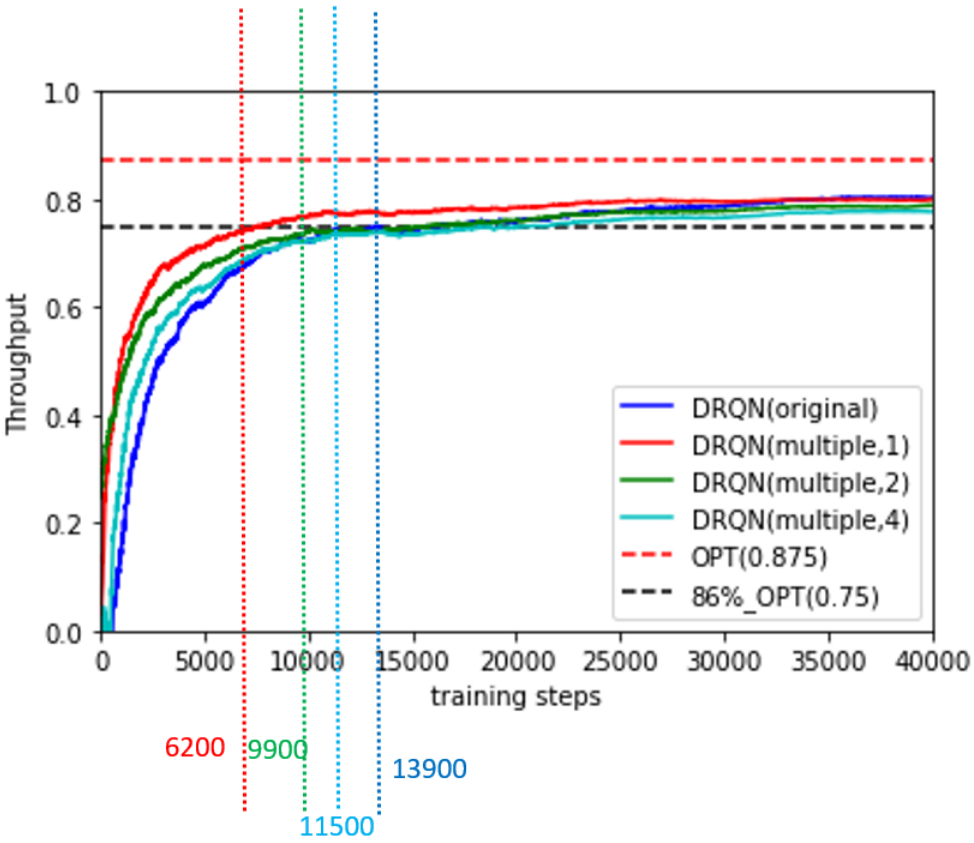


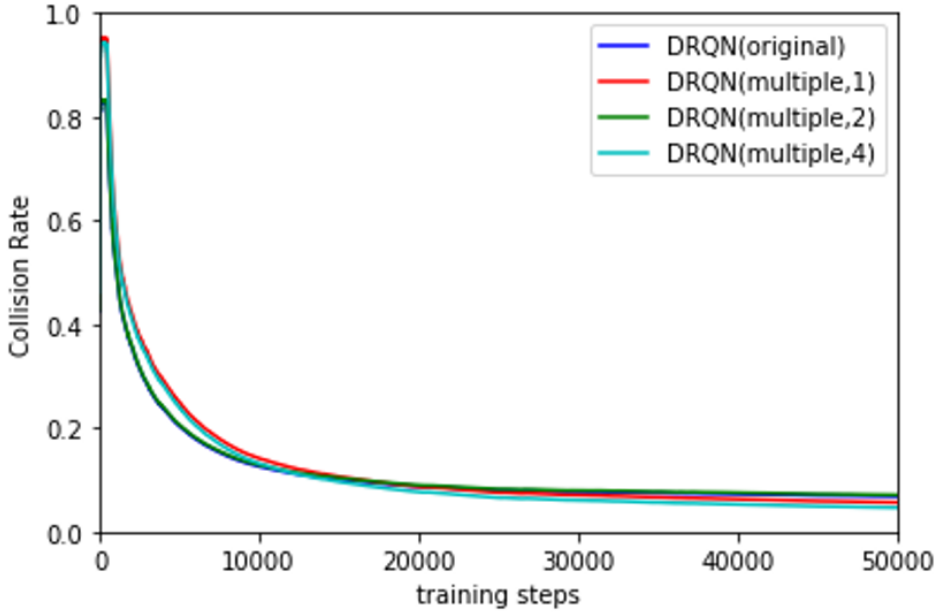
Figure 4.2: The Parallel Learning structure

in Table 4.3. In this work, We mainly consider three factors, the number of neural networks, the structure (size) of the neural network, and the frequency of learning (e.g., 1/2 means learning every two timeslots). It is worth noting that the computational complexity of DRQN is mainly reflected in the computational complexity of RNN. Therefore, we mainly change the overall complexity by adjusting the structure of the RNN. For example, we roughly think that the RNN with 40 LSTM cells is complex two times that of an RNN with 20 LSTM cells when the number of DRQN is the same. The testing environment is a stochastic spectrum environment with ten discrete independent channels. Each channel's character follows Markov transfer matrix (as shown in Section 4.2.1). Thus, based on the probability distribution, the optimal throughput is 0.875. Judging how convergence begins is a relatively subjective criterion. In this article, we choose 86% of the optimal performance as the 'converge flag.' The reason is that the performance gap will not exceed 10% in any two timeslots. Meanwhile, the agent is able to choose only one channel or wait for every timeslot.

From Fig.4.3, it can be seen that under the condition of the same number (four DRQN) of DRQNs, the higher the learning frequency, the faster the learning speed will converge. Compared to the agent without PL, the PL agent with four times the computational com-



(a) The Impact of Parallel Learning on Throughput Performance



(b) The Impact of Parallel Learning on Packet Collision Performance

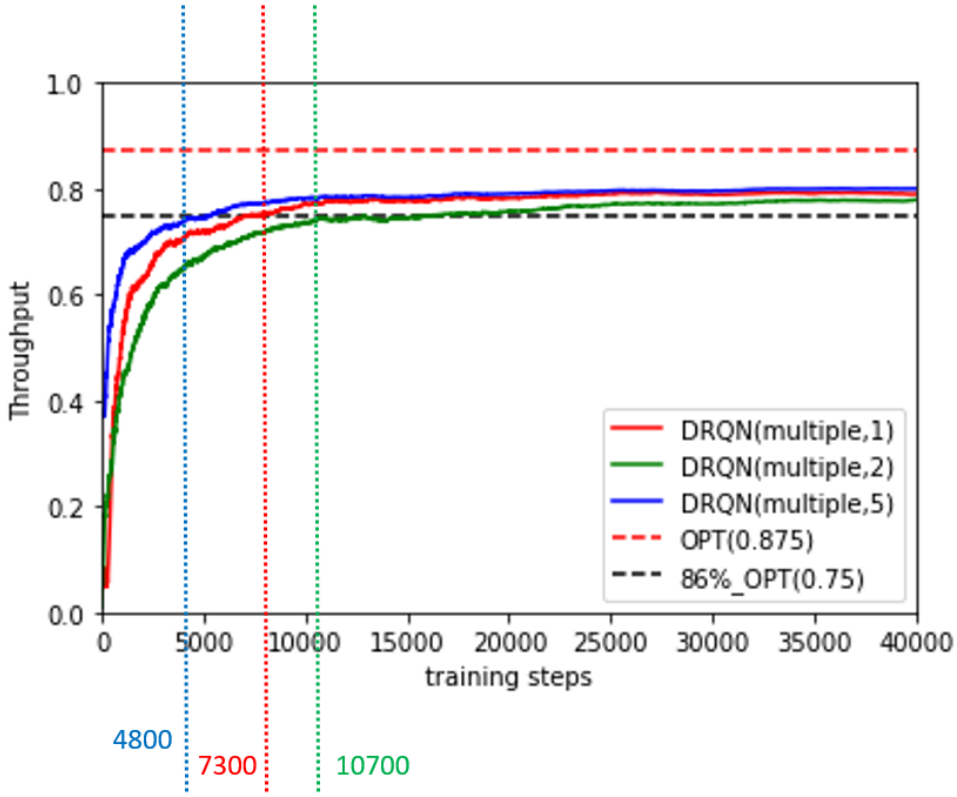
Figure 4.3: Throughput and Collision Rate Performance of Parallel Learning

Table 4.3: Simulation setting of Parallel Learning

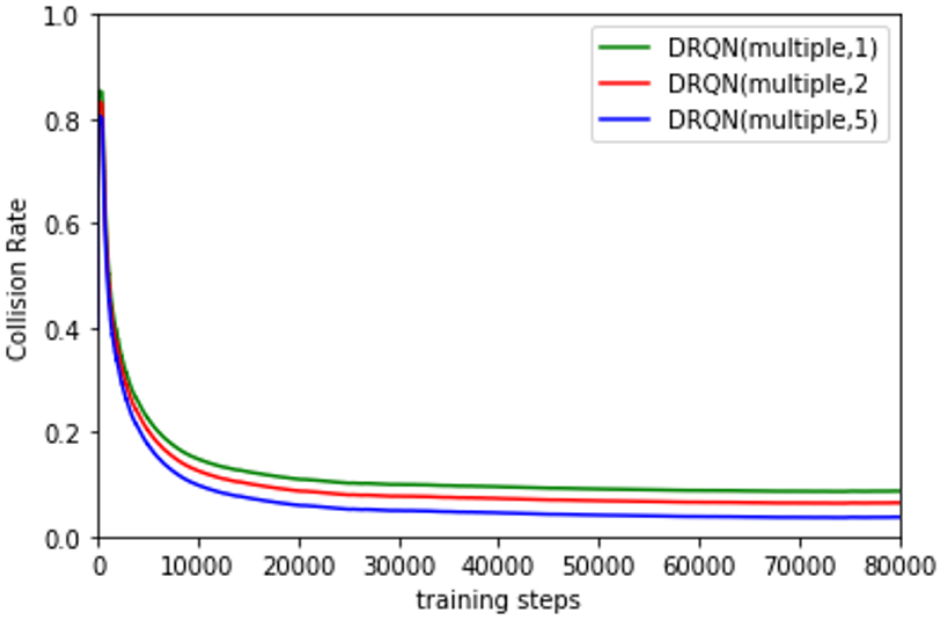
Number of NNs	Size of NNs	Learning Frequency	Computation Complexity
1	normal	1	1x (Fig.4.3)
2	normal	1/2	1x (Fig.4.3)
4	normal	1/2	2x (Fig.4.3)
4	normal	1	4x (Fig.4.3)
4	half	1	2x (Fig.4.4)
4	half	1/2	1x (Fig.4.4)
10	half	1	5x (Fig.4.4)

plexity can already achieve two times the convergence speed. Even the PL with two times the computational complexity can improve the convergence speed to 1.4 times. In addition, under the same computational complexity (1x complexity), the agent with PL can also have a certain speedup. Thus, 1. At the same time, the more learning networks, the faster the convergence speed can be achieved, but the computational complexity brought will also increase; 2. The learning frequency will also affect the learning effect. This is because more learning structures and higher learning frequencies can obtain more knowledge among units (and there will be more errors inevitably), but use the equalizer to balance the results of the overall learning network. To a certain extent, it will dilute the error and never further improve the learning efficiency.

From Fig.4.4, it can be seen that under the condition of the same size (half DRQN) of DRQNs, the number of DRQN and learning frequency still have an influence on the converge speed. More interestingly, compared with the green color in the Fig.4.3, under the same computational complexity (two times), the combination of high learning frequency (red color in the Fig.4.4) and the small neural network has a faster convergence rate than the combination of low learning frequency and large neural network. Meanwhile, the ability of PL to increase the learning rate by adding more NNs has limitations. This is because, in the case of a fixed environment, after the size of the neural network reaches a certain scale, the improvement of its learning efficiency is not obvious, but the learning frequency can



(a) The Impact of Parallel Learning on Throughput Performance



(b) The Impact of Parallel Learning on Packet Collision Performance

Figure 4.4: Throughput and Collision Rate Performance

continue to improve the efficiency. Also, more learning networks come with higher computational complexity, so balancing benefits and expenses in PL is a concern.

To sum up, Parallel Learning is a direct and effective way to accelerate convergence. However, it has an unavoidable flaw: the computational complexity will increase with the complexity of the task. Moreover, in practical problems, the environment is dynamic. Thus in the following approaches, we not only need to consider the effect of improving the converge speed but also consider practical problems such as computational complexity and dynamic environment.

4.3.2 General Transfer Learning

Transfer learning is a promising method to tackle the issue of converge speed and data scarcity. Transfer learning aims to improve the training effect of the target domain by transferring knowledge from different but similar source domains. There are four main transfer learning: Instance-based, mapping-based, network-based, and adversarial-based. As we mentioned, in real-world communication, the system environment is constantly changing, i.e., PUs and SUs are always moving in and out of the current environment. For example, if an SU enters a new environment or the current environment changes, it is unrealistic to learn the environment from the beginning due to various constraints. Thus, in order to overcome the emergence of limitations, the capabilities of (deep) transfer learning have been applied. Those methods can effectively solve the problem that the existing data are not similar enough for transfer learning. Moreover, those methods can reduce the size of the target domain data set required by model training and contribute to the rapid convergence of the target training model [96]. Thus, in the subsection, the approach of general transfer learning has been applied.

First, we introduce the simulated system. As shown in Fig.4.5, we separate learning and ex-

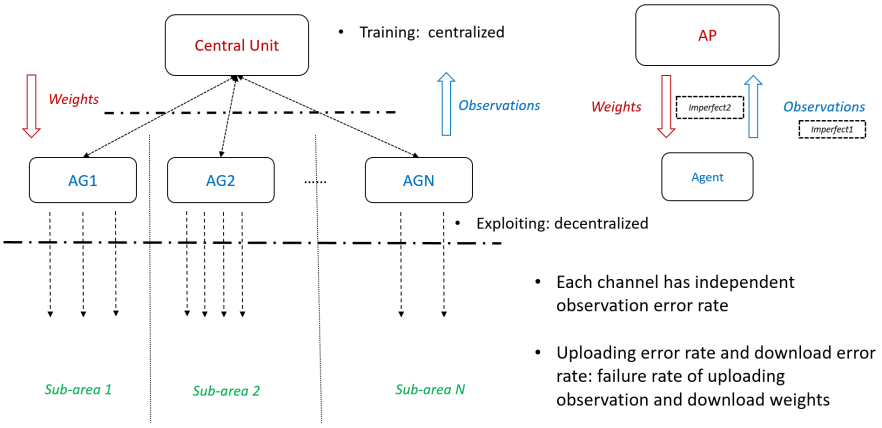


Figure 4.5: The Structure of Parallel Learning

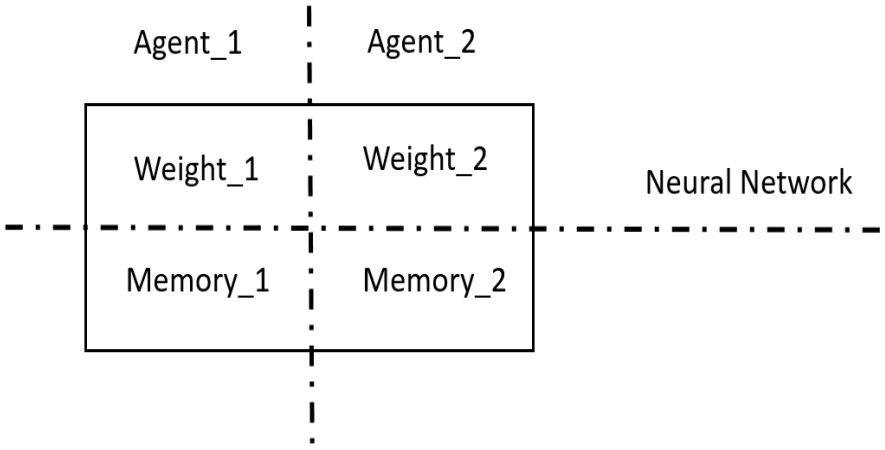


Figure 4.6: The Structure of Central Unit

ecution. A central unit in the environment is used to have a shared sizeable neural network. And there is enough space to allocate to the SUs in the environment for storing their data and training the parameters (as shown in Fig.Fig.4.6). SUs are in different environments, upload observations (partial observations), and download the corresponding training models. Of course, in order to make the problem more practical, we introduce two imperfect assumptions: 1. Each observed channel is an independent observation error (5%); 2. Uploads and downloads have a probability of failure(5%). If the failure occurs, the central unit or agent

will wait until the next serving timeslot. Moreover, each sub-environment changes dynamically (PUs' state), and when a new SU enters the specific environment, the central unit can identify the area and transmit the corresponding information. Again, the testing environment is a stochastic spectrum environment with ten discrete independent channels. Each channel's character follows Markov transfer matrix (as shown in Section 4.2.1). Meanwhile, each channel can be transferred to a legacy action pattern with 10% when the environment changes. We also introduce the character of hidden nodes (some channels are considered hidden nodes.). Thus, the throughput performance in the simulation will be replaced by a successful access rate. Here, it should be clear that, due to the continuous wait penalty, when the reward of 'wait' becomes negative, the 'wait' is considered a failure.

In Fig.4.7, there are three different environments (three independent APs), and three agents.

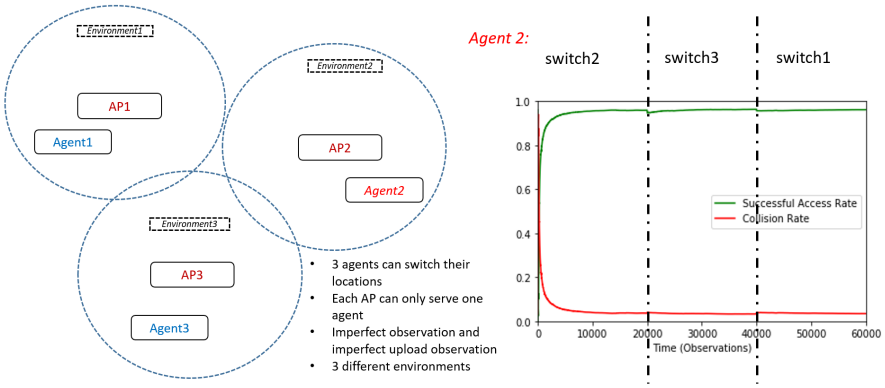


Figure 4.7: Transfer Learning in Stationary environment

We assume that each AP can only serve one agent, three agents can switch their locations, and the imperfect assumptions. In the environment, the agent will switch their location (for dynamic propose) with a pattern like 1->2->3->1. To make the research question more practical, we also introduce 3' frequencies':Uploading frequency(1/2, every two timeslots upload once), downloading frequency(1/4), and training frequency(1/2). A random action period is needed at the beginning. Due to the location of three agents is relatively fixed,

thus, we only show one agent's (agent No.2) performance. From the Fig. 4.7, it can be seen that when the agent can download the corresponding weights, it can very quick converge due to the environment is stationary.

However, the first simulation is too simple to illustrate the transfer learning. In Fig.4.8,

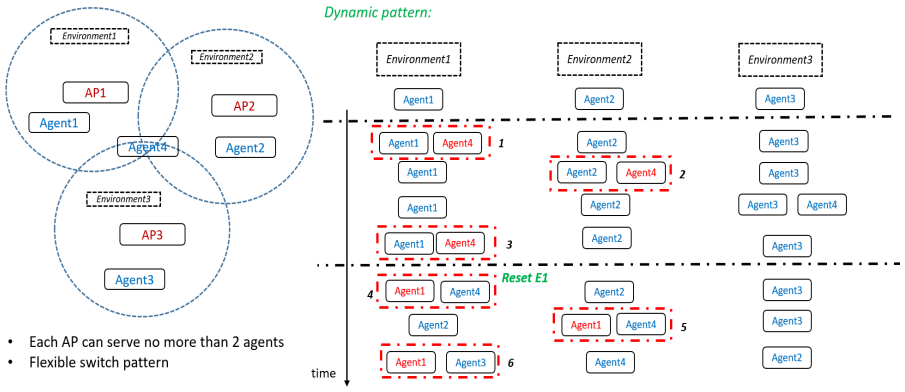


Figure 4.8: Transfer Learning in Dynamic environment with Multiple Agent

there are three different environments (three independent APs), and three agents. We assume that each AP can serve no more than two agents, agents can switch their locations (shown in Fig.4.8), and the imperfect assumptions. To make the research question more practical, we also introduce 3' frequencies':Uploading frequency(1/2, every two timeslots upload once), downloading frequency(1/4), and training frequency(1/2). A random action period is needed at the beginning. Because the AP can serve multiple agents simultaneously, the scenarios become complex. Here, we should notice five cases: 1. one 'old' agent, one set of trained weights, and one new come. We assume that the new one will download the weights and update from its own observation with an initial exploring rate; 2. one 'old' agent, two sets of weights, and a new one comes. We assume that the new one will download another weight and update from its own observation with an initial exploring rate; 3. no' old' agent, no set of weights, two newcomers. We assume that all the new agents learn the environment with initial exploring rates. And they Upload and download the information

separately; 4. no' old' agent, one set of weights, two newcomers. We assume that all the new agents download the same weights and learn the environment separately; 5. no' old' agent, two sets of weights, two newcomers. We assume that the agents can download a different set of weights and update them by observing initial exploring rates. Thus, in the simulation, we run six cases in which case1 runs two times (show in Fig.4.8, black number markers).

From Fig.4.9, We can find that transfer learning can help speed up the learning process

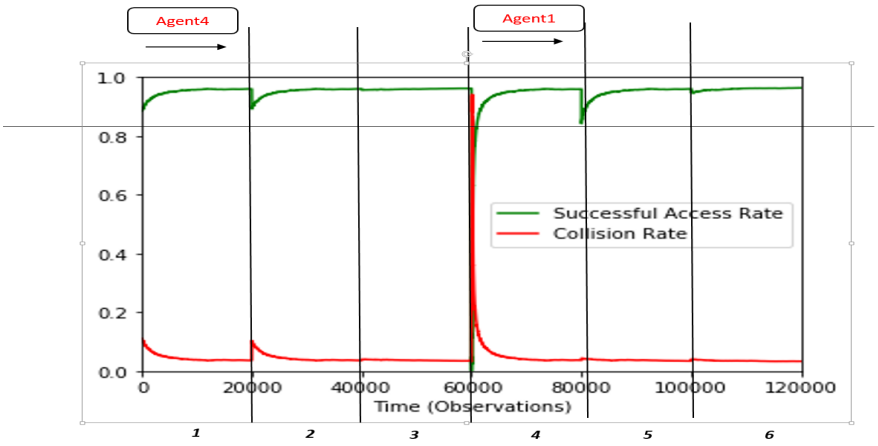


Figure 4.9: Performance of Transfer Learning in Dynamic environment with Multiple Agent

for a single agent but also help improve the convergence speed in multi-agent situations. However, another question introduced here is whether we need to repeat the exploration process if a set of parameters can be downloaded when a SU enters a new environment. In other words, whether an exploration process in a new environment will improve the overall performance. Therefore in the following simulation, we will make a comparison, that is, the performance comparison with and without the exploration process.

In this simulation, our dynamic system environment is a slowly changing process, i.e., every period (10000 timeslots), only one channel in the system will change its state, and the way of change is shown in Fig. 4.10. In the simulation, we mainly compare two different transfer learning methods: with and without exploration. From Fig.4.11, these two different approaches have different characteristics. The transfer learning without an exploration

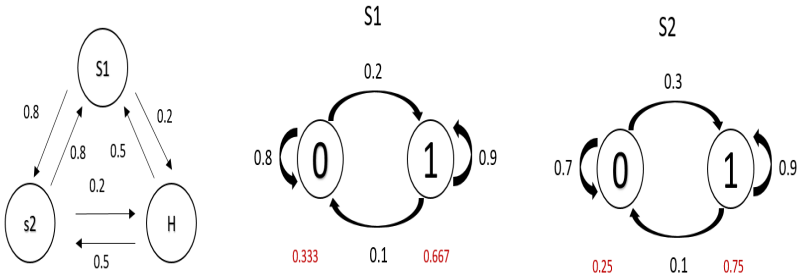


Figure 4.10: Channel Change Pattern in Exploration Process Issue

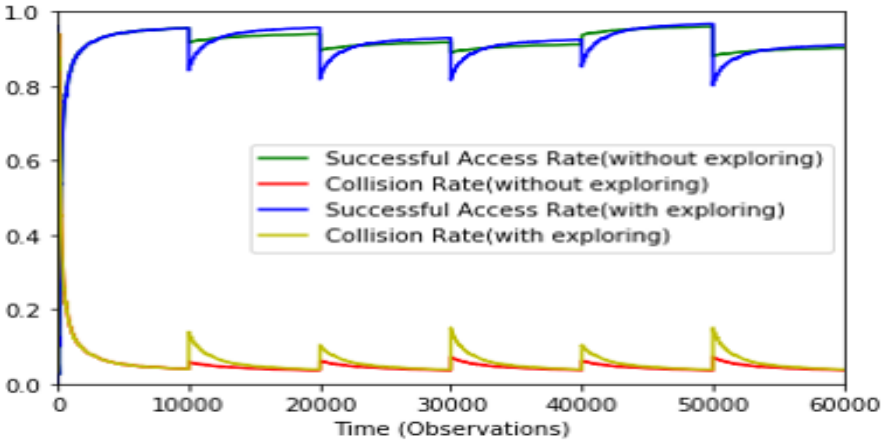


Figure 4.11: Performance Comparison of Exploration Process Issue

process has a faster converge speed, but the final performance (within limited timeslots) has some loss. On the contrary transfer learning with the exploration process has a little slower converge speed, but the final performance (within limited timeslots) is better. From another viewpoint, transfer learning with the exploration process may reach the optimal faster in the long term.

Finally, we explore the information transfer and integration capabilities of transfer learning. The way transfer learning transmits information is a topic worthy of study. In a large area covered by an AP, due to the large range and limited energy, the SU can only observe the state of part of the channel, and the remaining channels will become states of hidden nodes.

For example, in Fig. 4.12, the area covered by the AP is roughly divided into three parts, and the observed channel states of each region are different. Suppose agent3 is a new agent, and its observed information is as shown. At the same time, as a comparison, we prevent a SU that can observe the entire channel s(no hidden node in the observation result) as a comparison.

From Fig. 4.13, the left side of the figure is the result of completely autonomous learning by

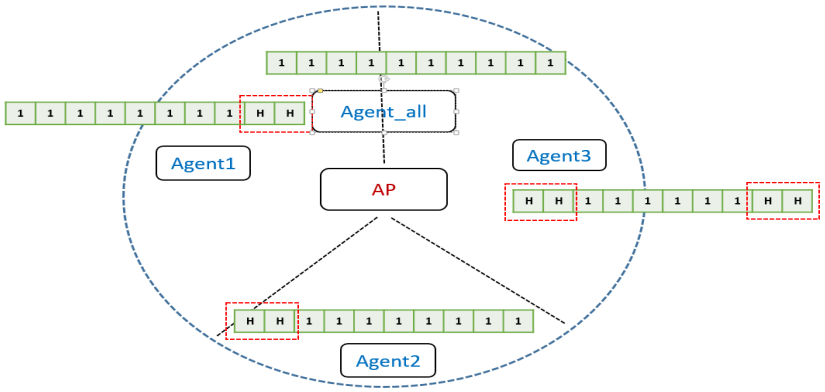


Figure 4.12: Channel Change Pattern in Exploration Process Issue

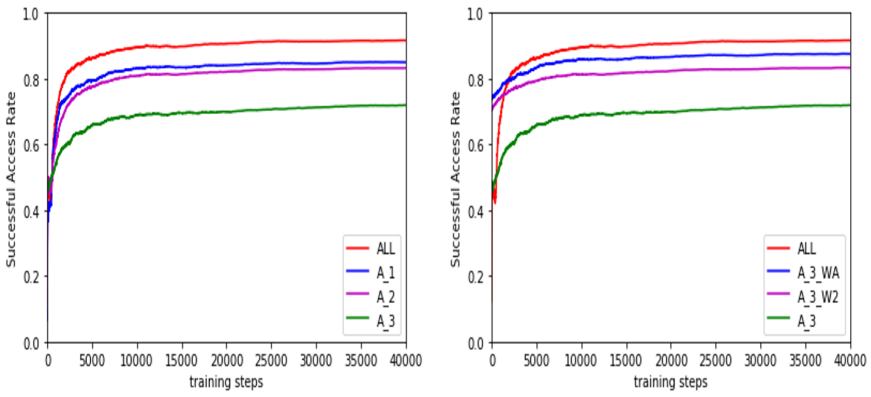


Figure 4.13: Performance Comparison of Exploration Process Issue

the new SU (agent3). Due to the increase in the number of hidden nodes, the final learning effect will also be affected. However, looking at the figure on the right, two transfer learning

methods are introduced here: 1. AP randomly passes an existing parameter to agent3, and then agent3 updates the parameter through its observation results ($A_{3W}2$ in the figure). 2. The AP uses the equalizer to integrate (equalize) two sets of existing parameters and pass them to agent3, and then agent3 updates the parameters ($A_{3W}A$ in the figure) through its own observations. By comparison, it can be found that transfer learning can improve the convergence speed compared with fully autonomous learning (A_3 in the figure).The reason is that although the observable channel states will be different, there are still parts of the information that are the same, so using the current parameters can save training time on the same parts. Secondly, using balanced transfer learning can improve the convergence speed and, even at the same time, the performance is relatively better. This is because, after the equalization parameters, almost all the state knowledge of the channel is more or less possessed. In other words, although the signal state information of the partial area will be 'diluted' after equalization, these parameters will still contain relevant information to help new users train.

In general, compared with the previous parallel learning, general transfer learning rapidly improves the convergence speed while maintaining relatively stable computational complexity and performance. But transfer learning also has some unavoidable problems. The first is that the ability to improve convergence speed is relatively limited (although the effect of improvement is rapid). Secondly, because it involves the sharing of some information, in real scenarios, the privacy of user information may have a negative impact. The last is the requirement of the network structure that a central unit is required as an interactive transfer station for necessary information (NN's weights). Therefore, in the following work, we will focus on further improving the convergence speed in the long term, which will not be limited by the network structure like the central unit, and can guarantee users' privacy.

4.3.3 Meta-learning

Meta-learning is most commonly understood as learning to learn, which refers to improving a learning algorithm over multiple learning episodes. In contrast, conventional ML improves model predictions over multiple data instances. Compared to the base learning (see Fig. 4.14) that one learning algorithm solves one task, meta-learning is an outer (or upper/meta) algorithm that updates the inner learning algorithm such that the model it learns improves an outer objective [70]. Thus, meta-learning is achieved by conceptually dividing learning into two levels: (a) The innermost level by which specific knowledge for specific tasks is acquired (e.g., fine-tuning an already acquired model for the consideration of a new data set). (b) The outer-most level we reach to across-task knowledge (e.g., optimization of efficient transfer between tasks). A meta-learning system should include a learning sub-system. Experience must be gained by a process that is based on knowledge acquired from metadata, which has been obtained from previously completed learning tasks. In other words, Meta-learning is conducted on learning episodes sampled from a task family, leading to a base learning algorithm that performs well on new tasks sampled from this family.

A common view of meta-learning is to learn a general-purpose learning algorithm that

	Goal	Input	output	function
Machine learning	Establish map (f) from training set (X) to labels set (Y)	training set (X)	labels set (Y)	f (weights for specify task)
Meta learning	From multiple tasks (T) and corresponding data (D), built function (F). F can output a f for new task	multiple tasks (T) and corresponding data (D)	f	F

Figure 4.14: Comparison for Machine Learning and Meta learning

can generalize across tasks and ideally enable each new task to be learned better than the last. Thus, such a task is associated with a data set D. The following equation presents the

optimal parameters of the model:

$$\theta^* = \arg_{\theta}^{\min} E_{DP(D)} \left[L_{\theta} \left(D \right) \right] \tag{4.2}$$

Here, each data set is considered a subset of data D, divided into a training set T and a forecasting set V. $D = \langle T, V \rangle$.These two kinds of data set is used for validation and testing. Thus, Learning how to learn thus becomes:

$$\min_{\theta} \underset{Tp(T)}{E} L \left(D; \theta \right) \tag{4.3}$$

where, $L \left(D; \theta \right)$ is loss function, At same time, it measures the performance of a model trained using θ on dataset D. 'How to learn'. $p \left(T \right)$ is a distribution of tasks. To solve this problem in practice, we often assume access to a set of source tasks sampled from $p \left(T \right)$. The source train and validation datasets are often called support and query sets. The meta-training step of 'learning how to learn' can be written as:

$$\theta^* = \arg \max_{\theta} \log p \left(\theta \left| D \right. \right) \tag{4.4}$$

In our proposed work, due to online learning, meta-learning may not fully solve the dynamic spectrum issue. Thus, we also should combine the DRQN with meta-learning. During the learning process or meta-learning process, we use Model-Agnostic Meta-Learning (MAML) [97]. The neural network is trained by using a few examples to adapt the model to new tasks faster. MAML is a general optimization and task-agnostic algorithm, and it is used to train the parameters of a model for fast learning with a small number of gradient updates. The process is shown in Fig. 4.15

First, we perform a basic Meta Learning test, as shown in Fig. 4.16. S indicates that the state of the channel follows the Markov transfer process. L indicates that the legacy node

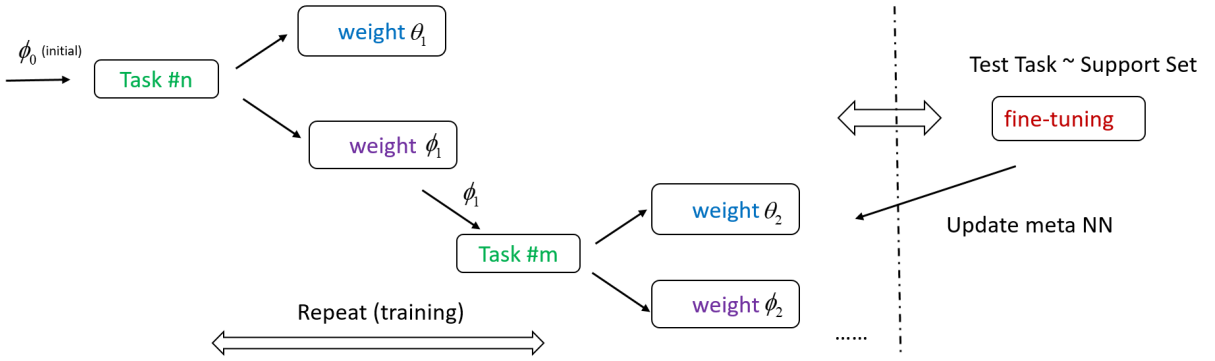


Figure 4.15: Diagram of Model-Agnostic Meta-Learning (MAML)

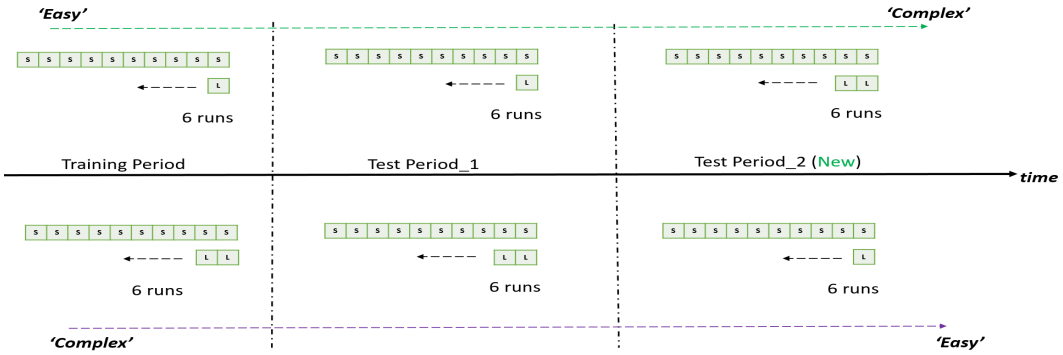


Figure 4.16: The Environment of Learning Order Issue

occupies the channel. If there is an L in the picture, which means that every time you run, there is a random position of S replaced by L. If there are two L in the figure, which means that every run, there are two S randomly replaced by L. The process at the top of the figure is that the training is a simple scene, the first test is a simple scene, and the second test is a complex scene that has not been experienced. Similarly, the process below is just the opposite. Finally,

From Fig.4.17, meta-learning can accelerate the learning process 'step by step'. Meanwhile, compared to transfer learning, the ultimate converge speed is much faster, but meta-learning need enough runs. The reason is that meta-learning is also learning. Thus, it should take enough training to reach the corresponding stable performance. Second, with the continuous update of meta-learning, the required convergence time will decrease faster, even when en-

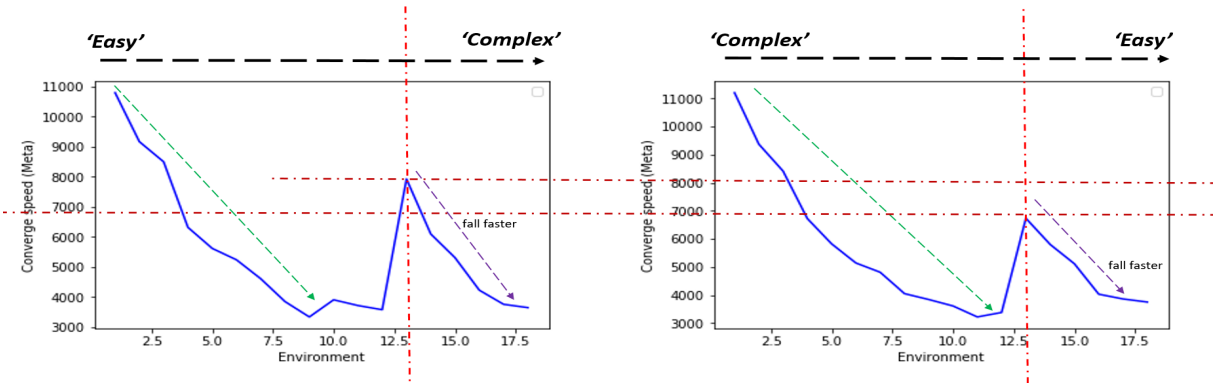


Figure 4.17: Performance of Meta-Learning in Learning Order Issue

countering a relatively unfamiliar environment. The reason is that although it is a relatively unfamiliar environment, meta-learning can learn and extract the corresponding knowledge and use it in this environment because it still has similar characteristics as before, thereby further accelerating the learning process.

Another issue worth studying is the symmetry environment. Strictly speaking, the sym-

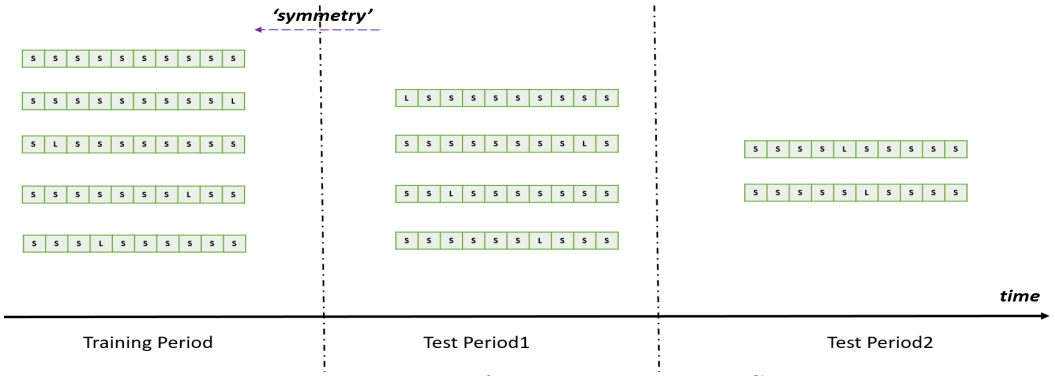


Figure 4.18: Environment of Meta Learning in Symmetry Issue

metry environment is relatively new to the user because the channel characteristics are distributed differently. So, in the next test, we will test the symmetry environment. As shown in Fig.4.17, the test period1 is the symmetry environment of the training period. The test period2 is a relatively new environment.

From Fig.4.19, meta-learning can accelerate all learning processes. Meanwhile, the accelerating process is faster when the agent meets the symmetry environment. Thus, it gives a

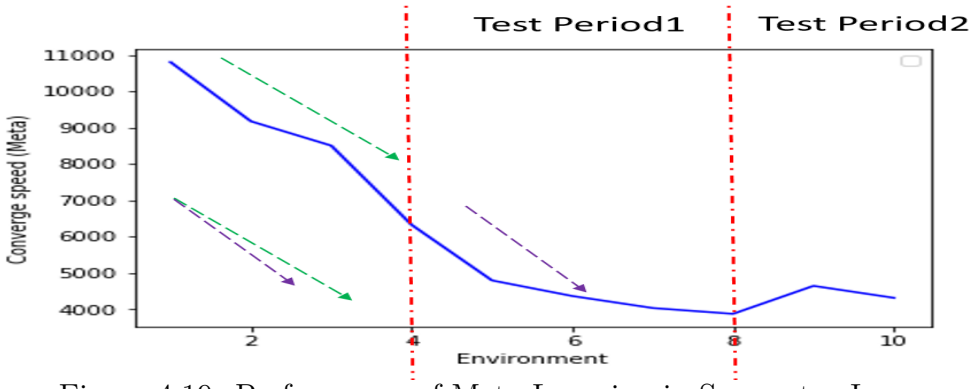


Figure 4.19: Performance of Meta Learning in Symmetry Issue

very important conclusion. When encountering a large and complex system, choosing the training environment with symmetry can effectively accelerate the user's learning and mastery of the entire system, thereby improving learning efficiency.

In the last simulation, we will compare the performance of two different main approaches,

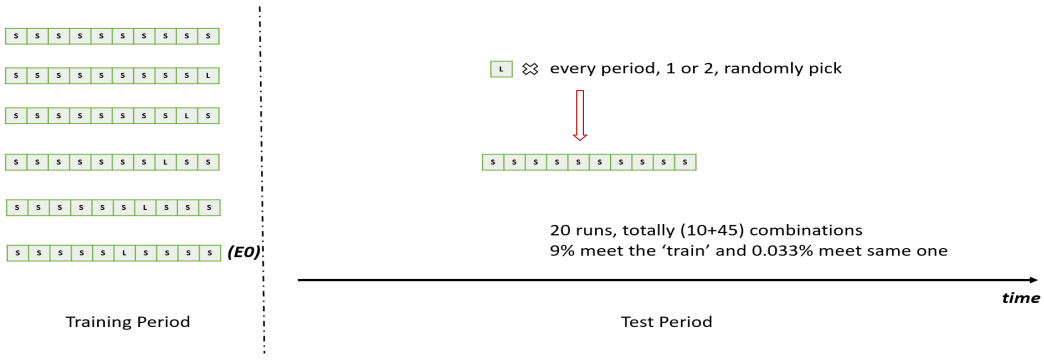


Figure 4.20: General Dynamic Environment

General Transfer Learning and Meta-learning, in a dynamic environment (shown in Fig.4.20).

From Fig.4.21, general transfer learning can accelerate the learning process faster, but its performance is worse than meta-learning. Although meta-learning's acceleration process is relatively slow, it is continuously accelerating; in other words, meta-learning's adaptability to environmental changes will be better. But the time needed to achieve stable performance is longer. Therefore, in general, both methods are feasible and characteristic. General transfer learning is better if you need to speed up the learning process quickly. But in the long

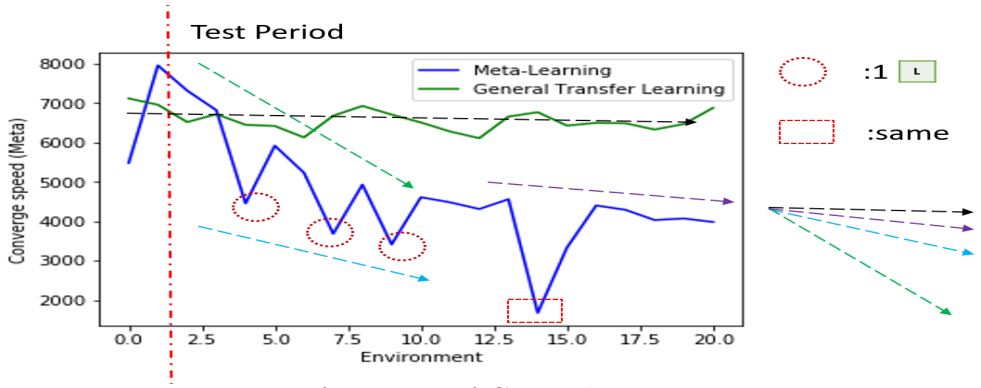


Figure 4.21: Performance of General Dynamic Environment

run, meta-learning will be better.

4.4 Conclusion

In this chapter, we considered a general and stochastic dynamic spectrum access problem where the system has several primary nodes, which follow the Markov transfer matrix, and secondary nodes exploit the idle periods based on partial observations of the spectrum without any prior knowledge of the primary nodes' behaviors. In previous chapters, our contribution concentrated on the performance of throughput and collision rate or smart sensing strategy. However, another issue, the converge speed, we seldom notice. Therefore, in this work, we mainly focus on finding effective approaches to accelerate the learning process with guaranteed performance. In work, we find different approaches from three different perspectihttps://www.overleaf.com/projectves, namely, the learning frequency (parallel learning structures or multiple learning), the inherited learning (general transfer learning), and the new learning approaches (Meta-learning). At the same time, it should be noted that the scenarios in which these three different methods can be used are different. The results demonstrate that three different approaches can accelerate the converge speed and guar-

antee the sub-optimal (and even optimal) performance under partial observation constrain. According to the simulations, meta-learning is the best choice for accelerating the learning process overall. But, if the system requires a faster accelerating learning process, general transfer learning is better. Parallel learning is the most straightforward way to implement the issue of accelerating learning. However, it needs much more resources for computing and storage.

It should be noted here that there are still many other ways to speed up the learning process, and even meta-learning at the same time, different structures will have different effects. Therefore, future work is to investigate more efficient solutions.

Chapter 5

Summary and Conclusions

Dynamic Spectrum Access (DSA) has strong potential to address the need for improved spectrum efficiency. Unfortunately, traditional DSA approaches such as simple "sense-and-avoid" approaches fail to provide sufficient performance in many scenarios. Thus, the combination of sensing with deep reinforcement learning (DRL) has been shown to be a promising alternative to previously proposed simplistic approaches. DRL does not require the explicit estimation of transition probability matrices or prohibitively large matrix computations, unlike traditional reinforcement learning methods. Further, since many learning approaches cannot solve the resulting online Partially-Observable Markov Decision Process (POMDP), we proposed the use of Deep Recurrent Q-Networks (DRQN) to determine the optimal channel access policy via online learning. The fundamental goal of this dissertation is to develop DRL-based solutions to address this POMDP-DSA problem.

Chapter 2 described a DRQN approach to determine an optimal access strategy in complex dynamic spectrum access scenarios where the system has several PUs dynamically accessing multiple channels and SUs which have no prior knowledge of the PUs' behavior. Instead, SUs base spectrum access decisions on partial observations of the spectrum and a learned strategy. In an environment with stochastic nodes, we have also examined the myopic policy, which is the optimal access policy when nodes follow a Markov chain model and the statistical information is known. Moreover, we have considered a cache-enabled DRQN-based learning approach. The system has several primary nodes dynamically accessing multiple

channels. Through simulations, we have shown that, although the DRQN SU is relatively slow to converge, it is able to achieve nearly optimal performance even without any prior knowledge. Even in extreme stochastic environments, the DRQN-based SU can still achieve excellent performance. Further, we examined the DRQN in the presence of nodes that follow fixed-pattern channel switching.

In chapter 2, our proposed approaches utilize the fixed sensing/observation pattern, which may not use sensing resources optimally. Thus, Chapter 3 developed a solution that uses a DRQN to determine the *optimal sensing and corresponding transmission policy* via online learning. We showed that the approach achieves near-optimal throughput and collision rate performance with a fraction of the needed observations/sensing resources. Moreover, we also showed that the proposed DRQN node could handle a more general stochastic environment, unlike a standard myopic policy. Further, the results have shown that the DRQN is robust when facing imperfect feedback from the environment. Third, we examined the relationship between DRQN's behavior and the reward settings, showing that modifying the relationship between rewards and penalties can make the node more or less aggressive (i.e., it can trade throughput for collisions.) The proposed node behaves well in the presence of multiple learning agents and dynamic environments. Finally, to control packet delay and the probability of dropped packets, we introduced cache (or buffer) information into the DRQN input. In the simulations, we have shown that the proposed DRQN node learns to exploit multiple available channels in order to reduce the cache size and control packet delay. Further, it was shown that without cache information, attempting to optimize throughput does not control packet delay.

In Chapters 2-3, our work concentrated on the performance of throughput and collision rate and the sensing strategy. However, another important issue, the convergence speed, was not emphasized. Therefore, in Chapter 4, we focused on finding effective approaches to accelerate learning without losing performance. In the chapter, we examine three different approaches,

namely, parallel learning (or multiple learning), inherited learning (general transfer learning), and meta-learning. At the same time, it should be noted that the scenarios in which these three different methods can be used are different. The results demonstrate that all three approaches can accelerate learning (i.e., reduce the convergence time) while providing near-optimal performance. Based on simulation results, meta-learning is found to be the best approach for accelerating the learning process from a long-term perspective. However, in the short-term, if the system requires faster learning, general transfer learning may be the better option. However, parallel learning is the most straightforward way to implement accelerated learning, although it needs more resources for computation and storage. Overall, we feel that meta-learning is the best solution for improving learning efficiency in a dynamic environment. At the same time, the results also demonstrate that different learning structures influence the performance of learning efficiency in slightly different ways.

Bibliography

[1] Zaher Dawy, Walid Saad, Arunabha Ghosh, Jeffrey G Andrews, and Elias Yaacoub. Toward Massive Machine Type Cellular Communications. *IEEE Wireless Communications*, 24(1):120–128, 2017.

[2] Godfrey Anuga Akpakwu, Bruno J Silva, Gerhard P Hancke, and Adnan M Abu-Mahfouz. A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges. *IEEE Access*, 6:3619–3647, 2017.

[3] Ibrar Yaqoob, Ejaz Ahmed, Ibrahim Abaker Targio Hashem, Abdelmuttlib Ibrahim Abdalla Ahmed, Abdullah Gani, Muhammad Imran, and Mohsen Guizani. Internet of Things Architecture: Recent Advances, Taxonomy, Requirements, and Open Challenges. *IEEE Wireless Communications*, 24(3):10–16, 2017.

[4] Federico Boccardi, Robert W Heath Jr, Angel Lozano, Thomas L Marzetta, and Petar Popovski. Five Disruptive Technology Directions for 5G. *ArXiv Preprint ArXiv:1312.0229*, 2013.

[5] Shancang Li, Li Da Xu, and Shanshan Zhao. 5G Internet of Things: A Survey. *Journal of Industrial Information Integration*, 10:1–9, 2018.

[6] Jeffrey G Andrews, Stefano Buzzi, Wan Choi, Stephen V Hanly, Angel Lozano, Anthony CK Soong, and Jianzhong Charlie Zhang. What Will 5G Be? *IEEE Journal on Selected Areas in Communications*, 32(6):1065–1082, 2014.

[7] Sundeep Rangan, Theodore S Rappaport, and Elza Erkip. Millimeter-Wave Cellular

Wireless Networks: Potentials and Challenges. *Proceedings of the IEEE*, 102(3):366–385, 2014.

[8] Mamta Agiwal, Abhishek Roy, and Navrati Saxena. Next Generation 5G Wireless Networks: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials*, 18(3):1617–1655, 2016.

[9] Robert W Heath, Nuria Gonzalez-Prelcic, Sundeep Rangan, Wonil Roh, and Akbar M Sayeed. An Overview of Signal Processing Techniques for Millimeter Wave MIMO Systems. *IEEE Journal of Selected Topics in Signal Processing*, 10(3):436–453, 2016.

[10] Deverajan Ganesh Gopal and Sekaran Kaushik. Emerging Technologies and Applications for Cloud-based Gaming: Review on Cloud Gaming Architectures. In *Emerging Technologies and Applications for Cloud-Based Gaming*, pages 67–87. IGI Global, 2017.

[11] Mingzhe Chen, Ursula Challita, Walid Saad, Changchuan Yin, and Mérouane Debbah. Artificial Neural Networks-based Machine Learning for Wireless Networks: A Tutorial. *IEEE Communications Surveys & Tutorials*, 21(4):3039–3071, 2019.

[12] 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.

[13] Suzhi Bi, Rui Zhang, Zhi Ding, and Shuguang Cui. Wireless Communications in the Era of Big Data. *IEEE Communications Magazine*, 53(10):190–199, 2015.

[14] Engin Zeydan, Ejder Bastug, Mehdi Bennis, Manhal Abdel Kader, Ilyas Alper Karatepe, Ahmet Salih Er, and Mérouane Debbah. Big Data Caching for Networking: Moving from Cloud to Edge. *IEEE Communications Magazine*, 54(9):36–42, 2016.

[15] Seok-Hwan Park, Osvaldo Simeone, and Shlomo Shamai Shitz. Joint Optimization of Cloud and Edge Processing for Fog Radio Access Networks. *IEEE Transactions on Wireless Communications*, 15(11):7621–7632, 2016.

[16] Alessio Zappone, Marco Di Renzo, and Mérouane Debbah. Wireless Networks Design in the Era of Deep Learning: Model-based, Ai-based, or Both? *IEEE Transactions on Communications*, 67(10):7331–7376, 2019.

[17] Y Xu, J Yu, and RM Buehrer. Dealing with Partial Observations in Dynamic Spectrum Access: Deep Recurrent Q-Networks. In *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*, pages 865–870. IEEE, 2018.

[18] Timothy J O¡¯Shea and Jakob Hoydis. An Introduction to Machine Learning Communications Systems. *ArXiv Preprint ArXiv:1702.00832*, 2017.

[19] Eliya Nachmani, Elad Marciano, Loren Lugosch, Warren J Gross, David Burshtein, and Yair Be¡¯ery. Deep Learning Methods for Improved Decoding of Linear Codes. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):119–131, 2018.

[20] Qing Zhao and Brian M Sadler. A Survey of Dynamic Spectrum Access. *IEEE Signal Processing Magazine*, 24(3):79–89, 2007.

[21] 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.

[22] Sudhir Srinivasa and Syed Ali Jafar. Cognitive Radios for Dynamic Spectrum Access-the Throughput Potential of Cognitive Radio: A Theoretical Perspective. *IEEE Communications Magazine*, 45(5), 2007.

[23] Ekram Hossain, Dusit Niyato, and Zhu Han. *Dynamic Spectrum Access and Management in Cognitive Radio Networks*. Cambridge University Press, 2009.

[24] Oshri Naparstek and Kobi Cohen. Deep Multi-User Reinforcement Learning for Distributed Dynamic Spectrum Access. *IEEE Transactions on Wireless Communications*, 18(1):310–323, 2018.

[25] Qing Zhao, Bhaskar Krishnamachari, and Keqin Liu. On Myopic Sensing for Multi-Channel Opportunistic Access: Structure, Optimality, and Performance. *IEEE Transactions on Wireless Communications*, 7(12), 2008.

[26] Keqin Liu and Qing Zhao. Indexability of Restless Bandit Problems and Optimality of Whittle Index for Dynamic Multichannel Access. *IEEE Transactions on Information Theory*, 56(11):5547–5567, 2010.

[27] Omer Melih Gul. Average Throughput of Myopic Policy for Opportunistic Access over Block Fading Channels. *IEEE Networking Letters*, 1(1):38–41, 2019.

[28] Nima Akbarzadeh and Aditya Mahajan. Dynamic Spectrum Access under Partial Observations: A Restless Bandit Approach. In *2019 16th Canadian Workshop on Information Theory (CWIT)*, pages 1–6. IEEE, 2019.

[29] Ethem Alpaydin. *Introduction to Machine Learning*. MIT Press, 2009.

[30] Jiawei Han, Jian Pei, and Micheline Kamber. *Data Mining: Concepts and Techniques*. Elsevier, 2011.

[31] Michael I Jordan and Tom M Mitchell. Machine Learning: Trends, Perspectives, and Prospects. *Science*, 349(6245):255–260, 2015.

[32] Pat Langley and Herbert A Simon. Applications of Machine Learning and Rule Induction. *Communications of the ACM*, 38(11):54–64, 1995.

[33] Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from Data*, volume 4. AMLBook New York, NY, USA:, 2012.

[34] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-Supervised Learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[35] Sun-Chong Wang. Artificial Neural Network. In *Interdisciplinary Computing in Java Programming*, pages 81–100. Springer, 2003.

[36] Robert Hecht-Nielsen. Theory of the Backpropagation Neural Network. In *Neural Networks for Perception*, pages 65–93. Elsevier, 1992.

[37] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-Art in Artificial Neural Network Applications: A Survey. *Heliyon*, 4(11):e00938, 2018.

[38] Russell Beale and Tom Jackson. *Neural Computing - An Introduction*. CRC Press, 1990.

[39] Bernard Widrow, David E Rumelhart, and Michael A Lehr. Neural Networks: Applications in Industry, Business and Science. *Communications of the ACM*, 37(3):93–106, 1994.

[40] 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.

[41] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level Control Through Deep Reinforcement Learning. *Nature*, 518(7540):529, 2015.

[42] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*, volume 1. MIT Press Cambridge, 1998.

[43] Yuxuan Wang and DeLiang Wang. A Deep Neural Network for Time-domain Signal Reconstruction. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4390–4394. IEEE, 2015.

[44] Ahmet M Elbir, Kumar Vijay Mishra, and Yonina C Eldar. Cognitive Radar Antenna Selection via Deep Learning. *IET Radar, Sonar & Navigation*, 2019.

[45] Hongji Huang, Jie Yang, Hao Huang, Yiwei Song, and Guan Gui. Deep Learning for Super-Resolution Channel Estimation and DOA Estimation based Massive MIMO System. *IEEE Transactions on Vehicular Technology*, 67(9):8549–8560, 2018.

[46] Ronan Collobert and Jason Weston. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167. ACM, 2008.

[47] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech Recognition with Deep Recurrent Neural Networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649. IEEE, 2013.

[48] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[50] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face Recognition: A Convolutional Neural-Network Approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997.

[51] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3D Convolutional Neural Networks for Human Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2012.

[52] Song Han, Huizi Mao, and William J Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *ArXiv Preprint ArXiv:1510.00149*, 2015.

[53] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent Neural Network Regularization. *ArXiv Preprint ArXiv:1409.2329*, 2014.

[54] Larry R Medsker and LC Jain. Recurrent Neural Networks. *Design and Applications*, 5:64–67, 2001.

[55] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to Forget: Continual Prediction with LSTM. 1999.

[56] 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.

[57] Matthew Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPS. *CoRR, abs/1507.06527*, 7(1), 2015.

[58] Christopher Schulze and Marcus Schulze. Vizdoom: DRQN with Prioritized Experience Replay, Double-Q Learning, & Snapshot Ensembling. *ArXiv Preprint ArXiv:1801.01000*, 2018.

[59] Oshri Naparstek and Kobi Cohen. Deep Multi-User Reinforcement Learning for Dynamic Spectrum Access in Multichannel Wireless Networks. *ArXiv Preprint ArXiv*, 1704, 2017.

[60] Hao-Hsuan Chang, Hao Song, Yang Yi, Jianzhong Zhang, Haibo He, and Lingjia Liu. Distributive Dynamic Spectrum Access through Deep Reinforcement Learning: A Reservoir Computing based Approach. *IEEE Internet of Things Journal*, 2018.

[61] GM Foody and MK Arora. An Evaluation of Some Factors Affecting the Accuracy of Classification by An Artificial Neural Network. *International Journal of Remote Sensing*, 18(4):799–810, 1997.

[62] Wei Tang, Gang Hua, and Liang Wang. How to Train A Compact Binary Neural Network with High Accuracy? In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017 Feb 13.

[63] Taskin Kavzoglu. Increasing the Accuracy of Neural Network Classification Using Refined Training Data. *Environmental Modelling & Software*, 24(7):850–858, 2009.

[64] Yue Xu, Jianyuan Yu, and R Michael Buehrer. The Application of Deep Reinforcement Learning to Distributed Spectrum Access in Dynamic Heterogeneous Environments with Partial Observations. *IEEE Transactions on Wireless Communications*, 19(7):4494–4506, 2020.

[65] Y Xu, J Yu, and R Michael Buehrer. Cache-Enabled Dynamic Spectrum Access via Deep recurrent Q-networks with Partial Observation. In *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pages 1–2. IEEE, 2019.

[66] Alan Kramer and Alberto Sangiovanni-Vincentelli. Efficient Parallel Learning Algo-

rithms for Neural Networks. *Advances in Neural Information Processing Systems*, 1, 1988.

[67] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A Survey of Transfer Learning. *Journal of Big data*, 3(1):1–40, 2016.

[68] Lisa Torrey and Jude Shavlik. Transfer Learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pages 242–264. IGI global, 2010.

[69] Ricardo Vilalta and Youssef Drissi. A Perspective View and Survey of Meta-Learning. *Artificial Intelligence Review*, 18(2):77–95, 2002.

[70] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-Learning in Neural Networks: A Survey. *ArXiv Preprint ArXiv:2004.05439*, 2020.

[71] Hado Van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-Learning. In *AAAI*, volume 16, pages 2094–2100, 2016.

[72] Jarmo Lundén, Sanjeev R Kulkarni, Visa Koivunen, and H Vincent Poor. Multiagent Reinforcement Learning based Spectrum Sensing Policies for Cognitive Radio Networks. *IEEE Journal of Selected Topics in Signal Processing*, 7(5):858–868, 2013.

[73] Yi Li, Hong Ji, Xi Li, and Victor CM Leung. Dynamic Channel Selection with Reinforcement Learning for Cognitive WLAN over Fiber. *International Journal of Communication Systems*, 25(8):1077–1090, 2012.

[74] 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.

[75] Husheng Li. Learning the Spectrum via Collaborative Filtering in Cognitive Radio Networks. pages 1–12, 2010.

[76] Jan Oksanen, Jarmo Lundén, and Visa Koivunen. Reinforcement Learning based Sensing Policy Optimization for Energy Efficient Cognitive Radio Networks. *Neurocomputing*, 80:102–110, 2012.

[77] Y Xu, J Yu, WC Headley, and RM Buehrer. Deep Reinforcement Learning for Dynamic Spectrum Access in Wireless Networks. In *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*, pages 207–212. IEEE, 2018.

[78] Martn Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A System for Large-Scale Machine Learning. In *OSDI*, volume 16, pages 265–283, 2016.

[79] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *ArXiv Preprint ArXiv:1412.6980*, 2014.

[80] Igor Kadota, Elif Uysal-Biyikoglu, Rahul Singh, and Eytan Modiano. Minimizing the Age of Information in Broadcast Wireless Networks. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 844–851. IEEE, 2016.

[81] Igor Kadota and Eytan Modiano. Minimizing the Age of Information in Wireless Networks with Stochastic Arrivals. *IEEE Transactions on Mobile Computing*, 2019.

[82] Hao-Hsuan Chang, Lingjia Liu, and Yang Yi. Deep Echo State Q-network (DEQN) and Its Application in Dynamic Spectrum Sharing for 5G and Beyond. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[83] Hao Song, Lingjia Liu, Jonathan Ashdown, and Yang Yi. A Deep Reinforcement Learning Framework for Spectrum Management in Dynamic Spectrum Access. *IEEE Internet of Things Journal*, 2021.

[84] Mantas Lukoševičius. A Practical Guide to Applying Echo State Networks. In *Neural Networks: Tricks of the Trade*, pages 659–686. Springer, 2012.

[85] Nan Zhao, Chao Xu, Shun Zhang, Yiping Xie, Xijun Wang, and Hongguang Sun. Status Update for Correlated Energy Harvesting Sensors: A Deep Reinforcement Learning Approach. In *2020 International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 170–175. IEEE, 2020.

[86] Mohammad Hatami, Mojtaba Jahandideh, Markus Leinonen, and Marian Codreanu. Age-Aware Status Update Control for Energy Harvesting IoT Sensors via Reinforcement Learning. In *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1–6. IEEE, 2020.

[87] Xiao Gao, Zheng Dou, and Lin Qi. A New Distributed Dynamic Spectrum Access Model Based on DQN. In *2020 15th IEEE International Conference on Signal Processing (ICSP)*, volume 1, pages 351–355. IEEE, 2020.

[88] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. *Advances in Neural Information Processing Systems*, 29, 2016.

[89] Nan Zhao, Ying-Chang Liang, Dusit Niyato, Yiyang Pei, Minghu Wu, and Yunhao Jiang. Deep Reinforcement Learning for User Association and Resource Allocation in Heterogeneous Cellular Networks. *IEEE Transactions on Wireless Communications*, 18(11):5141–5152, 2019.

[90] Chang Liu, Xuemeng Liu, Derrick Wing Kwan Ng, and Jinhong Yuan. Deep Residual Learning for Channel Estimation in Intelligent Reflecting Surface-Assisted Multi-User Communications. *IEEE Transactions on Wireless Communications*, 21(2):898–912, 2021.

[91] Xiaowen Ye, Yiding Yu, and Liqun Fu. Multi-Channel Opportunistic Access for Heterogeneous Networks based on Deep Reinforcement Learning. *IEEE Transactions on Wireless Communications*, 21(2):794–807, 2021.

[92] Solmaz Niknam, Harpreet S Dhillon, and Jeffrey H Reed. Federated Learning for Wireless Communications: Motivation, Opportunities, and Challenges. *IEEE Communications Magazine*, 58(6):46–51, 2020.

[93] Zhijin Qin, Geoffrey Ye Li, and Hao Ye. Federated Learning and Wireless Wommunications. *IEEE Wireless Communications*, 28(5):134–140, 2021.

[94] Weiting Zhang, Dong Yang, Wen Wu, Haixia Peng, Ning Zhang, Hongke Zhang, and Xuemin Shen. Optimizing Federated Learning in Distributed Industrial IoT: A Multi-Agent Approach. *IEEE Journal on Selected Areas in Communications*, 39(12):3688–3703, 2021.

[95] Joaquin Vanschoren. Meta-Learning. In *Automated Machine Learning*, pages 35–61. Springer, Cham, 2019.

[96] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A Comprehensive Survey on Transfer Learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

[97] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for

Fast Adaptation of Deep Networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.