Deep Reinforcement Learning for Network Slicing

Zhifeng Zhao, Rongpeng Li, Qi Sun, Chi-Lin I, Yangchen Yang, Xianfu Chen, Minjian Zhao, and Honggang Zhang

Abstract

Network slicing means an emerging business to operators and allows them to sell the customized slices to various tenants at different prices. In order to provide better-performing and cost-efficient services, network slicing involves challenging technical issues and urgently looks forward to intelligent innovations to make the resource management consistent with users' activities per slice. In that regard, deep reinforcement learning (DRL), which focuses on how to interact with the environment by trying alternative actions and reinforces the tendency actions producing more rewarding consequences, is emerging as a promising solution. In this paper, after briefly reviewing the fundamental concepts and evolution-driving factors of DRL, we investigate the application of DRL in some typical resource management scenarios of network slicing, which include radio resource slicing and priority-based core network slicing, and demonstrate the performance advantage of DRL over several competing schemes through extensive simulations. Finally, we also discuss the possible challenges to apply DRL in network slicing from a general perspective.

Index Terms

Deep Reinforcement Learning, Network Slicing, Neural Networks, Q-Learning

I. Introduction

The fifth-generation cellular networks (5G) is assumed to be the key infrastructure provider for the next decade, by means of profound changes in both radio technologies and network architecture design [1]–[4]. Besides the pure performance metrics like rate, reliability and

Z. Zhao, R. Li, M. Zhao and H. Zhang are with Zhejiang University, Hangzhou 310027, China, (email: {zhaozf, lirongpeng, mjzhao, honggangzhang}@zju.edu.cn).

Q. Sun and C.-L. I are with China Mobile Research Institute, Beijing 100053, China (email: {sunqiyjy, icl}@chinamobile.com).

C. Yang is with Beihang University, Beijing, 100191, China (email: cyyang@buaa.edu.cn)

X. Chen is with VTT Technical Research Centre of Finland, Oulu FI-90571, Finland (email: xianfu.chen@vtt.fi).

allowed connections, the scope of 5G also incorporates the transformation of the mobile network ecosystem and accommodates heterogeneous services using one infrastructure. In order to achieve such a goal, 5G will fully glean the recent advances in the network virtualization and programmability [1], [2], and provide a novel technique named *network slicing* [1], [5]–[7]. Network slicing tries to get rid of the current, relatively monolithic architecture like the forth-generation cellular networks (4G) and slice the whole network into different parts, each of which is tailed to meet specific service requirement. Therefore, network slicing means an emerging business to operators and allows them to sell the customized network slices to various tenants at different prices. In a word, network slicing could act as a service (NSaaS) [5]. NSaaS is quite similar to the mature business "infrastructure as a service (IaaS)", the benefit of which service providers like Amazon and Microsoft have happily enjoyed for a while. However, in order to provide better-performing and cost-efficient services, network slicing involves more challenging technical issues, since (a) for radio access networks, spectrum is a scarce resource and it is meaningful to guarantee the spectrum efficiency (SE) [8], while for core networks, virtualized functionalities are limited by computing resources as well; (b) the service level agreements (SLAs) with slice tenants usually impose stringent requirements on quality of experience (QoE) perceived by users [9]; and (c) the actual demand of each slice heavily depends on the request patterns of mobile users. Hence, in the 5G era, it is critical to investigate how to intelligently respond to the dynamics of service request from mobile users [7], so as to obtain satisfactory QoE in each slice at the cost of acceptable spectrum or computing resources [4].

On the other hand, partially inspired by the psychology of animal learning, the learning agent in reinforcement learning (RL) algorithm focuses on how to interact with the environment (represented by *states*) by trying alternative *actions* and reinforces the tendency actions producing more rewarding consequences [10]. Besides, reinforcement learning also embraces the theory of optimal control and adopts some ideas like value functions and dynamic programming. However, reinforcement learning faces some difficulties in dealing with large state space, since it is challenging to traverse every state and obtain a value function or model for every station-action pair in a direct and explicit manner. Hence, benefiting from the advances in graphics processing units (GPUs) and the less concern for the computing power, some researchers propose to sample only a fraction of states and further apply neural networks (NN) to train a sufficiently accurate value function or model. Following this idea, Google DeepMind has pioneered the combination of NN and one typical RL algorithm (i.e., $\mathcal{Q}$-Learning), and obtained one deep reinforcement learning (DRL) algorithm with enough

performance stabilities [11], [12].

The well-known success of AlphaGo [11] and following exciting results to apply DRL to solve resource allocation issues in some specific networks like cognitive radio networks, cache-enabled interference alignment wireless networks [13] and cloud radio access networks [14], have aroused intensive research interest to extend DRL to the field of network slicing. However, given the challenging technical issues in network slicing, it is critical to carefully investigate the application performance of DRL in the following aspects:

- The basic concern is whether or not the application of DRL is feasible. More specifically, does DRL produce satisfactory QoE results while bringing acceptable spectrum or computing resources?
- The research community has proposed some schemes for the resource management in network slicing scenarios. For example, the resource management could be conducted by either following a meticulously designed prediction algorithm, or equally dividing the available resource into each slice. The former implies another reasonable solution, while the latter saves a lot of computational cost. Hence, a comparison between DRL and these interesting schemes is also necessary.

In this paper, we will try to talk about these issues.

II. From Reinforcement Learning to Deep Reinforcement Learning

In this part, we give a brief introduction over RL or more specifically $\mathcal{Q}$-Learning, and then talk about the motivation to evolve from $\mathcal{Q}$-Learning to Deep $\mathcal{Q}$-Learning.

A. Reinforcement Learning

RL learns how to interact with the environment to achieve maximum cumulative return (or average return), and has been successfully applied in the fields like robot control, self driving, and chess playing for years. Mathematically, RL follows the typical concept of Markov decision process (MDP), while the MDP is a generalized framework for modeling decision-making problems in cases where the result is partially random and affected by the applied decision. An MDP can be formulated by a 5-tuple as $M = \langle S, A, P_{s,a}, R, \gamma \rangle$, where S and A denote a finite state space and action set, respectively. $P(s'|s, a)$ indicates the probability that the action $a \in A$ under state $s \in S$ at slot t leads to state $s' \in S$ at slot $t+1$. $R(s, a)$ is an immediate reward after performing the action a under state s, while $\gamma \in [0, 1]$ is a discount factor to reflect the diminishing importance of current reward on future ones. Usually, the goal of MDP is to find a policy $a = \pi(s)$ that determines the selected action a

under state s, so as to maximize the value function, which is typically defined as the expected discounted cumulative reward by the Bellman equation:

$$\begin{aligned} V^{\pi}(\hat{s}) &= E_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k}R(s^{(k)},\pi(s^{(k)}))|s^{(0)}=\hat{s}\right] \\ &= E_{\pi}\left[R(\hat{s},\pi(\hat{s}))+\gamma\sum_{s'\in\mathcal{S}}P(s'|\hat{s},\pi(\hat{s}))V^{\pi}(s')\right], \end{aligned} \tag{1}$$

Dynamic programming could be exploited to solve the Bellman equation when the state transition probability $P(s'|s,a)$ is known apriori with no random factors. But inspired by both control theory and behaviorist psychology, RL aims to obtain the optimal policy π^{*} under circumstances with unknown and partially random dynamics. Since RL does not have explicit knowledge over whether it has come close to its goal, it needs the balance between exploring new potential actions and exploiting the already learnt experience. So far, there has been some classical RL algorithms like $\mathcal{Q}$-learning, actor-critic method, SARSA, TD(λ), etc. Given by the detailed methodologies and practical application scenarios, we can classify these RL algorithms according to different criteria:

- *Model-based versus Model-free*: Model-based algorithms imply the agent tries to learn the model of how the environment works from its observations and then plan a solution using that model. Once the agent gains adequately accurate model, it can use a planing algorithm with its learned model to find a policy. However, model-free algorithms means the agent does not directly learn how to model the environment. Instead, like the classical example of $\mathcal{Q}$-learning, the agent estimates the Q-values (or roughly the value function) of each state-action pair and derives the optimal policy by choosing the action yielding the largest Q-value in the current state. In a word, different from the model-based algorithm, the well-learnt model-free algorithm like $\mathcal{Q}$-learning cannot predict the next state and state before taking the action.
- *Monte-Carlo Update versus Temporal-Difference Update*: Generally, the value function update could be conducted in two ways, that is, the monte-carlo update and the temporal-difference (TD) update. A Monte-Carlo update means the agent updates its estimation for a state-action pair by calculating the mean return from a collection of episodes. But, a TD update approximates the estimation by comparing estimates at two consecutive episodes. For example, $\mathcal{Q}$-learning updates its Q-value by the TD update as $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a))$, where α is the learning rate. Specifically, the term $R(s,a)+\gamma \max_{a'} Q(s',a')-Q(s,a)$ is also named as the TD error, since it captures

the difference between the current (sampled) estimate $R(s,a) + \gamma \max_{a'} Q(s',a')$ and previous one $Q(s,a)$.

- *On-policy versus Off-policy*: The value function update is also coupled with the executed update policy. But, before updating the value function, the agent also needs to sample and learn the environment by performing some non-optimal policy. If the update policy is irrelevant to the sampling policy, the agent is called to perform an off-policy update. Taking the example of $\mathcal{Q}$-learning, this off-policy agent updates the Q-value by choosing the action corresponding to the best Q-value, while it could learn the environment by adopting sampling policies like ϵ-greedy or Boltzmann distribution to balance the "exploration and exploitation" problem [10]. The $\mathcal{Q}$-learning proves to converge regardless of the chosen sampling policy. On the contrary, the SARSA agent is on-policy, since it updates the value function by $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s,a) + \gamma Q(s',a') - Q(s,a))$ where a' and a need to chosen according to the same policy.

B. From $\mathcal{Q}$-Learning to Deep $\mathcal{Q}$-Learning

We first summarize the details of $\mathcal{Q}$-Learning. Generally speaking, $\mathcal{Q}$-Learning belongs to one model-free, TD update, off-policy RL algorithm, and consists of three major steps:

1) Without loss of generality, the agent chooses an action a under state s according to some policy like ϵ-greed. The ϵ-greed policy means the agents chooses the action with the largest Q-value $Q(s,a)$ with a probability of ϵ, and equally chooses the other actions with a probability of $\frac{1-\epsilon}{|A|}$, where $|A|$ denotes the size of the action space.
2) The agent learns the reward $R(s,a)$ from the environment, and the state transitions to s'.
3) The agent updates the Q-value function as $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a))$.

Classical RL algorithms usually rely on two different ways (i.e., explicit table or function approximation) to store the estimated value functions. For the table storage, RL algorithm uses an array or hash table to store the learnt results for each state-action pair. For large state space, it not only requires intensive storage, but also is unable to quickly transverse the complete the state-action pair. On the contrary, due to the curse of dimensionality, function approximation sounds more appealing.

The most straightforward way for function approximation is a linear approach. Taking the example of $\mathcal{Q}$-learning, the Q-value function could be approximated by a linear combination of n orthogonal bases $\boldsymbol{\psi}(s,a) = \{\psi_1(s,a) \cdots \psi_n(s,a)\}$, that is, $Q(s,a) = \theta_0 \cdot 1 + \theta_1 \cdot$

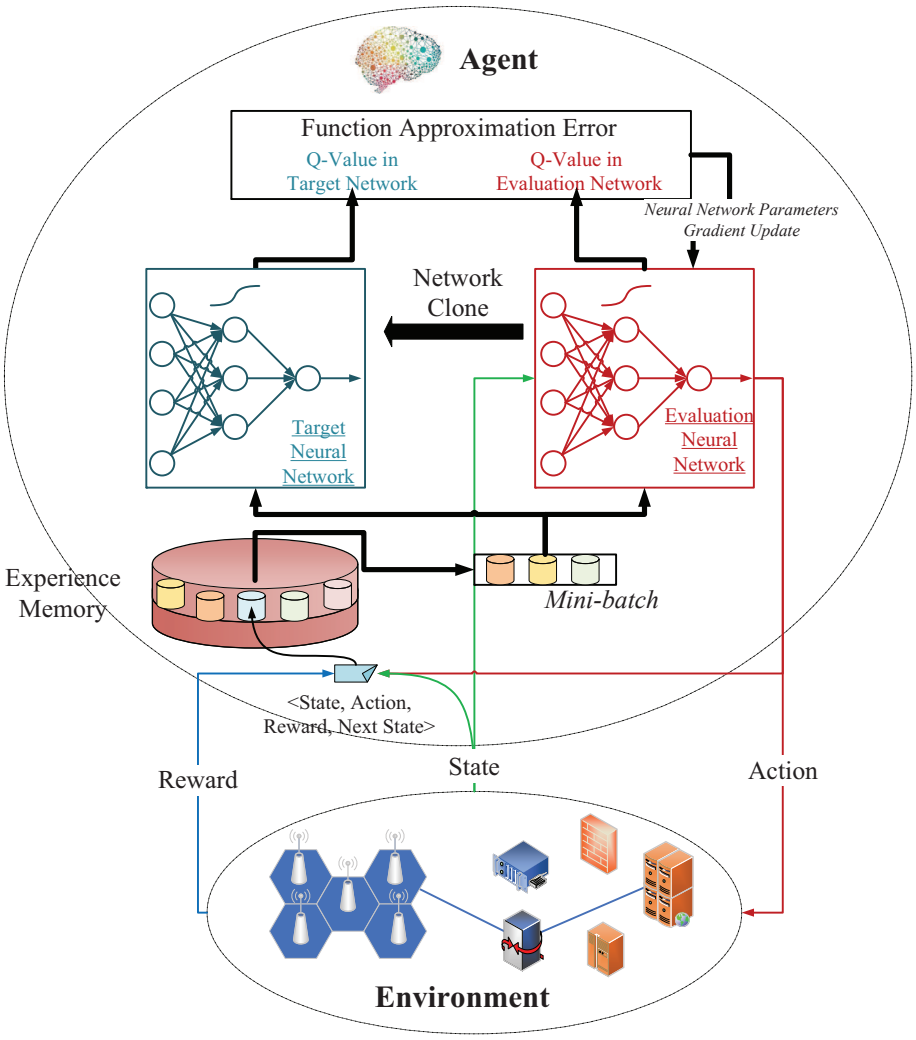


Fig. 1. An illustration of deep $\mathcal{Q}$-learning.

$\psi_1(s,a) + \cdots + \theta_n \cdot \psi_n(s,a) = \boldsymbol{\theta}^T\boldsymbol{\psi}(s,a)$, where θ_0 is a biased term with 1 absorbed into the $\boldsymbol{\psi}$ for simplicity of representation and $\boldsymbol{\theta}$ is a vector with the dimension of n. The function approximation in the $\mathcal{Q}$-learning means that $Q(s,a) = \boldsymbol{\theta}^T\boldsymbol{\psi}(s,a)$ should be as close as the learnt "target" value $Q^+(s,a) = \sum_s P(s'|s,a)\big[R(s,a) + \gamma \max_{a'} Q^+(s',a')\big]$ over all the state-action pairs. Since it is infeasible to transverse all the state-action pairs, the "target" value could be approximated based on the minibatch samples and $Q^+(s,a) \approx R(s,a) + \gamma \max_{a'} Q^+(s',a')$. In order to make $Q(s,a) = \boldsymbol{\theta}^T\boldsymbol{\psi}(s,a)$ approach the "target" value $Q^+(s,a)$, the objective function could be defined as

$$\begin{aligned} L(\boldsymbol{\theta}) &= \frac{1}{2}\big(Q^+(s,a) - Q(s,a)\big)^2 \\ &= \frac{1}{2}\big(Q^+(s,a) - \boldsymbol{\theta}^T\boldsymbol{\psi}(s,a)\big)^2 \end{aligned} \tag{2}$$

The parameter $\boldsymbol{\theta}$ minimizing $L(\boldsymbol{\theta})$ could be achieved by a gradient-based approach as

$\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} - \alpha \nabla L(\boldsymbol{\theta}^{(i)}) = \boldsymbol{\theta}^{(i)} - \alpha\big(Q^{+}(s,a) - \boldsymbol{\theta}^{T}\boldsymbol{\psi}(s,a)\big)\boldsymbol{\psi}(s,a)$. For a large state-action space, the function approximation reduces the number of unknown parameters to a vector with dimension n and the related gradient method further solves the parameter approximation in an computationally efficient manner.

Apparently, the linear function approximation could not accurately model the estimated value function. Hence, some researchers have proposed to replace the approximation $Q(s,a;\boldsymbol{\theta})$ by some non-linear means. In that regard, neural networks is skilled in modeling non-linear functions. Therefore, in AlphaGo, neural networks have been exploited and the loss function in (2) can be re-defined as $L(\boldsymbol{\theta}) = \frac{1}{2}\big(Q^{+}(s,a) - Q(s,a;\boldsymbol{\theta})\big)^2$. Besides, AlphaGo has some novel progress in the following aspects:

- *Experience Replay* [12]: The agent stores the past experience (i.e., the tuple $e_t = \langle s_t, a_t, s'_t, R(s_t,a_t)\rangle$) at episode t into a dataset $D_t = (e_1, \cdots, e_t)$ and uniformly selects some (mini-batch) items from the dataset to update the Q-value neural network $Q(s,a;\boldsymbol{\theta})$.
- *Network Cloning*: The agent uses a separate network $\hat{Q}$ to guide how to select an action a in state s, and the network $\hat{Q}$ is replaced by Q every C episodes. Practical simulation results demonstrate that this network cloning also enhances the learning stability [12].

Both experience replay and network cloning have also motivated to choose the off-policy $\mathcal{Q}$-learning, since the sampling policy is only contingent on previously trained Q-value neural networks and the updating policy relies on the information from the new episodes and is irrespective of the sampling policy.

Finally, we illustrate deep $\mathcal{Q}$-learning in Fig. 1.

III. Resource Management for Network Slicing

Resource management is a permanent topic during the evolution of wireless communication. Intuitively, resource management for network slicing can be considered from several different perspectives.

- *Radio Resource and Virtualized Network Functions*: As depicted in Fig. 2, resource management for network slicing involves both radio access part [15] and core network part with slightly different optimization goals. Due to the limited spectrum resource, the resource management for the radio access puts considerable efforts in allocating suitable symbol length, sub-carrier spacing, cycle prefix length to one slice, so as to maintain acceptable SE while trying to bring appealing rate and small delay. But the widely adopted optical transmission in core networks has decreased the importance

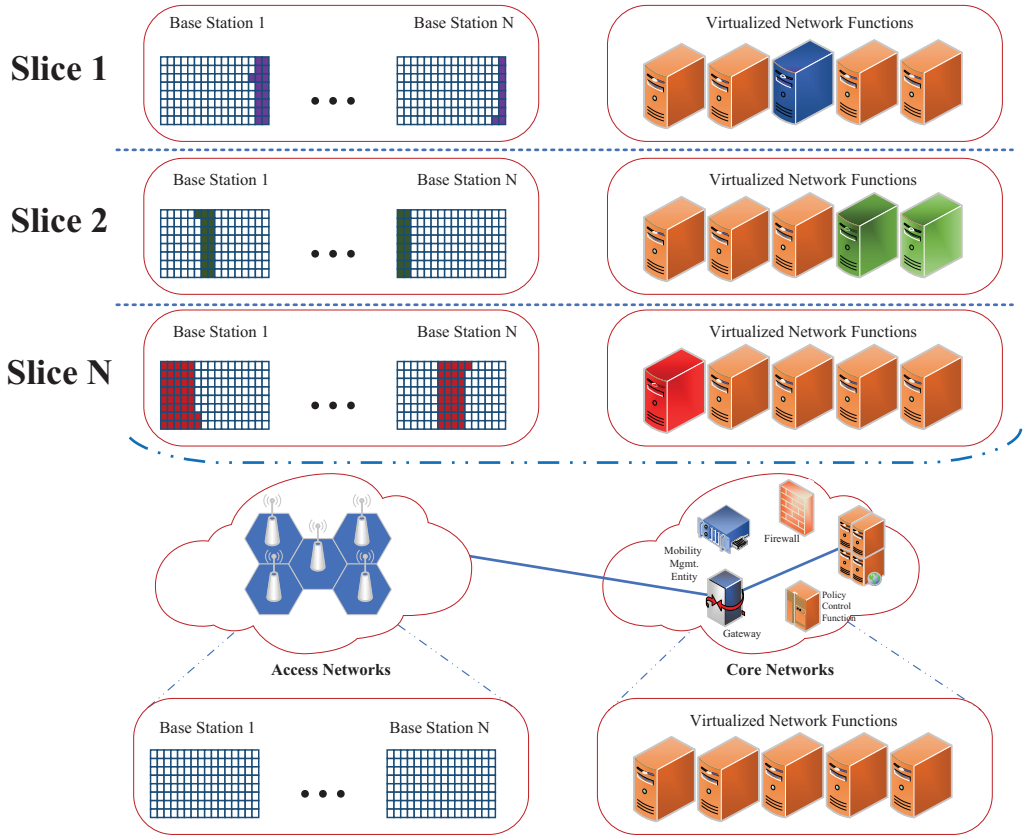


Fig. 2. An illustration of resource management for network slicing.

of SE constraints. Thus, the optimization of core network changes to design appropriate forwarding tables for the packets from one specific slice and focuses heavily on scheduling delay. Fortunately, by balancing the relative importance of resource utilization and QoE satisfaction ratio, the resource management problem could be formulated as $R = \zeta \cdot \text{SE} + \beta \cdot \text{QoE}$, where ζ and β denotes the importance of SE and QoE.

- *Equal or Prioritized Customers for one Tenant*: As a business-initialized concept, network slicing should provide flexible commercial schemes for tenants. Therefore, similar to the cloud computing industry, the basic idea is that one tenant could simply buy one slice with required SLA. On the other hand, the tenant could provide differentiated services and thus leverage several sub-slices with different SLAs to obtain such a goal. In this latter case, it is indispensable to design a prioritized scheduling, so as to effectively exploit the purchasing virtualized resource (i.e., resource utilization) and shorten the waiting time (WT) of customers in different SLA queues. Mathematically, the objective goal could be written as $R = \zeta \cdot \text{RU} + \beta \cdot \text{WT}$, where ζ and β denotes the importance of RU and WT.

TABLE I

A BRIEF SUMMARY OF KEY SETTINGS IN DRL FOR NETWORK SLICING SIMULATIONS

(a) The Mapping from Resource Management for Network Slicing to DRL.

	Radio Resource Slicing	Priority-based Core Network Slicing
State	The number of arrived packets in each slice within a specific time window	The priority and time-stamp of last arrived five flows in each service function chain (SFC)
Action	Allocated bandwidth to each slice	Allocated SFC for the flow at current timestamp
Reward	Weighted Sum of SE and QoE in 3 sliced bands	Weighted sum of average time in 3 SFCs

(b) Parameter settings for radio resource slicing

	VoLTE	Video	URLLC
Bandwidth	10 MHz		
Scheduling	Round robin per slot (0.5 ms)		
Channel	Rayleigh fading		
User No. (100 in all)	46	46	8
Inter-Arrival Time	Uniform [Min = 0, Max = 160ms]	Truncated Pareto [Exponential Para = 1.2, Mean = 6 ms, Max = 12.5 ms]	Exponential [Mean = 180 ms]
Packet Size	40 B	Truncated Pareto [Exponential Para = 1.2, Mean = 100 B, Max = 250 B]	Truncated Lognormal [Mean = 2 MB, Standard Deviation = 0.722 MB, Maximum =5 MB]
SLA: Rate	51 kbps	5 Mbps	10 Mbps
SLA: Latency	10 ms	10 ms	5 ms

Based on the aforementioned discussions, we can safely reach a conclusion that, the objective of resource management for network slicing should take account of several variables (e.g., SE or V_{a1}, QoE or V_{b1}, RU or V_{a2}, WT or V_{b2}, etc) and might be difficult to track. But, the objective could be uniformly formulated as $R = \zeta V_a + \beta V_b$ and considered as the reward of the learning algorithms.

A. Radio Resource Slicing

In this part, we address how to apply DRL for radio resource slicing. Specifically, we simulate within a scenario containing one single BS with three types of services (e.g., VoLTE, video, ultra-reliable low-latency communications (URLLC) for operations), which can be regarded as one typical application scenario in a hospital. There exist 100 registered subscribers with service models summarized in Table I(b). It can be observed from Table I(b), URLLC has less frequent packets compared with the others, while VoLTE requires the smallest bandwidth for its packets.

Our goal aims at optimizing the weighted summation of system SE and slice QoE, where the latter is defined as simultaneously satisfying the rate and latency requirement. We adopt DQL as a promising solution by adopting the mapping in Table I(a) and performs round-robin scheduling method within each slice. We also compare the simulation results with the following three methods, so as to better explain the importance of DQL.

- *Demand-prediction based method*: The method tries to estimate the possible demand by using long short-term memory (LSTM) to predict the number of active users per slice. Afterwards, the bandwidth is allocated by two ways: (1) *DP-No* allocates the whole bandwidth to each slice proportional to the number of predicted users; (2) *DP-BW* performs the allocation by multiplying the number of predicted users by the least required bandwidth and then computing the proportion. Round-robin is conducted within each slice.
- *Hard slicing*: Hard slicing means that each slice is always allocated $\frac{1}{3}$ of the whole bandwidth. Again, round-robin is conducted within each slice.
- *No slicing*: All users are scheduled equally. Round-robin is conducted within all users.

Fig. 3 present the learning process of DQL in radio resource management. In particular, Fig. 3(a)$\sim$3(i) give the initial performance of DQL when the QoE weight is 50 and the SE weight is 0.1, while Fig. 3(j)$\sim$3(r) provide the performance during the last 50 learning updates. From these sub-figures, it can be observed that DQL could not well match the user activity at the very beginning. But after nearly 50000 updates, the gap between the allocated bandwidth and the transmitted packets in reality becomes significantly more narrower. Besides, Fig. 3(s) and Fig. 3(t) show the variations of SE and QoE along with each learning update when the SE weight is 0.1. From both subfigures, a larger QoE weight prefers those actions producing superior QoE performance while bringing certain loss in the system SE performance.

Fig. 4 provides a detailed performance comparison among candidate techniques. Here,

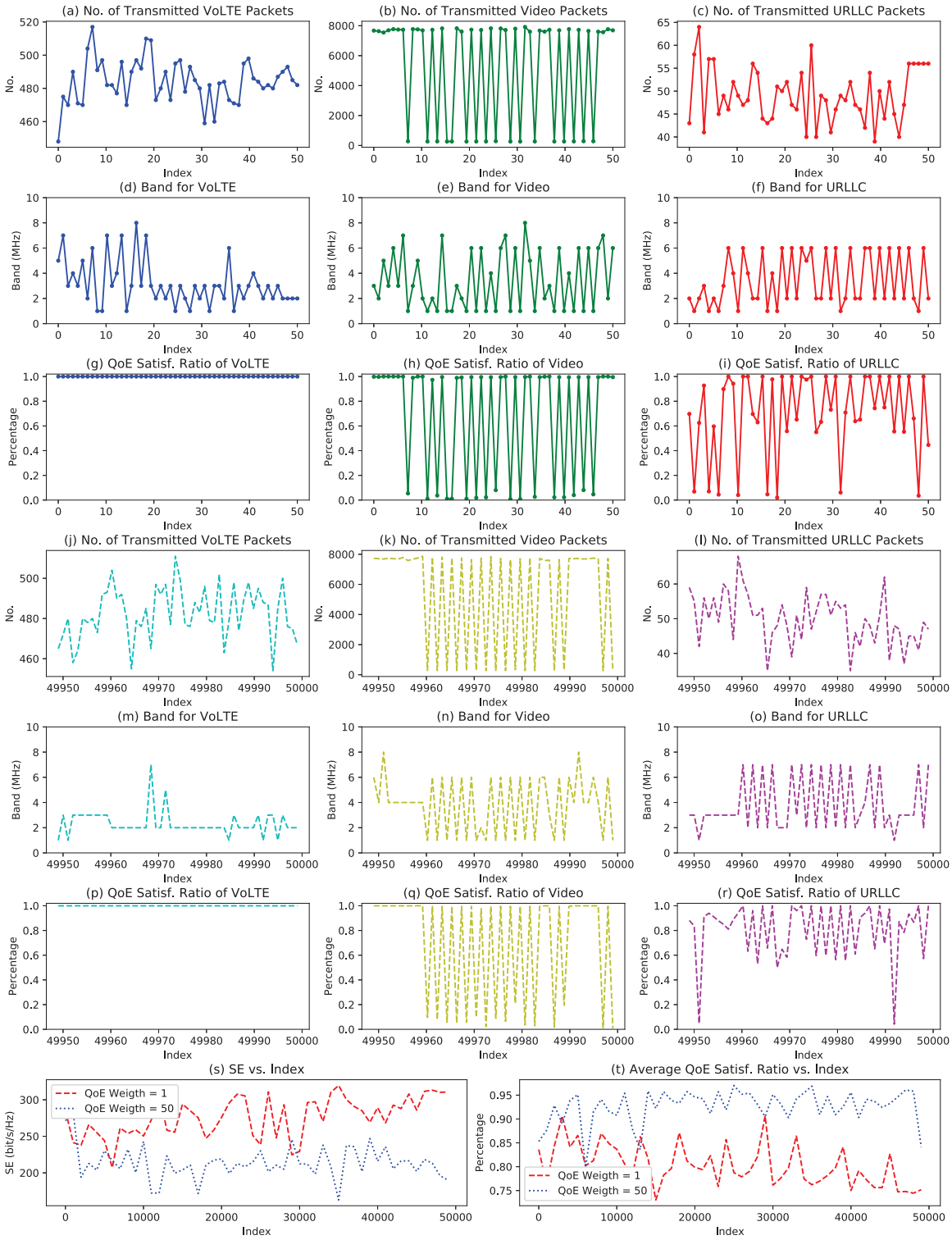


Fig. 3. The Performance of DQL for radio resource slicing w.r.t. the learning steps (QoE Weight = 50).

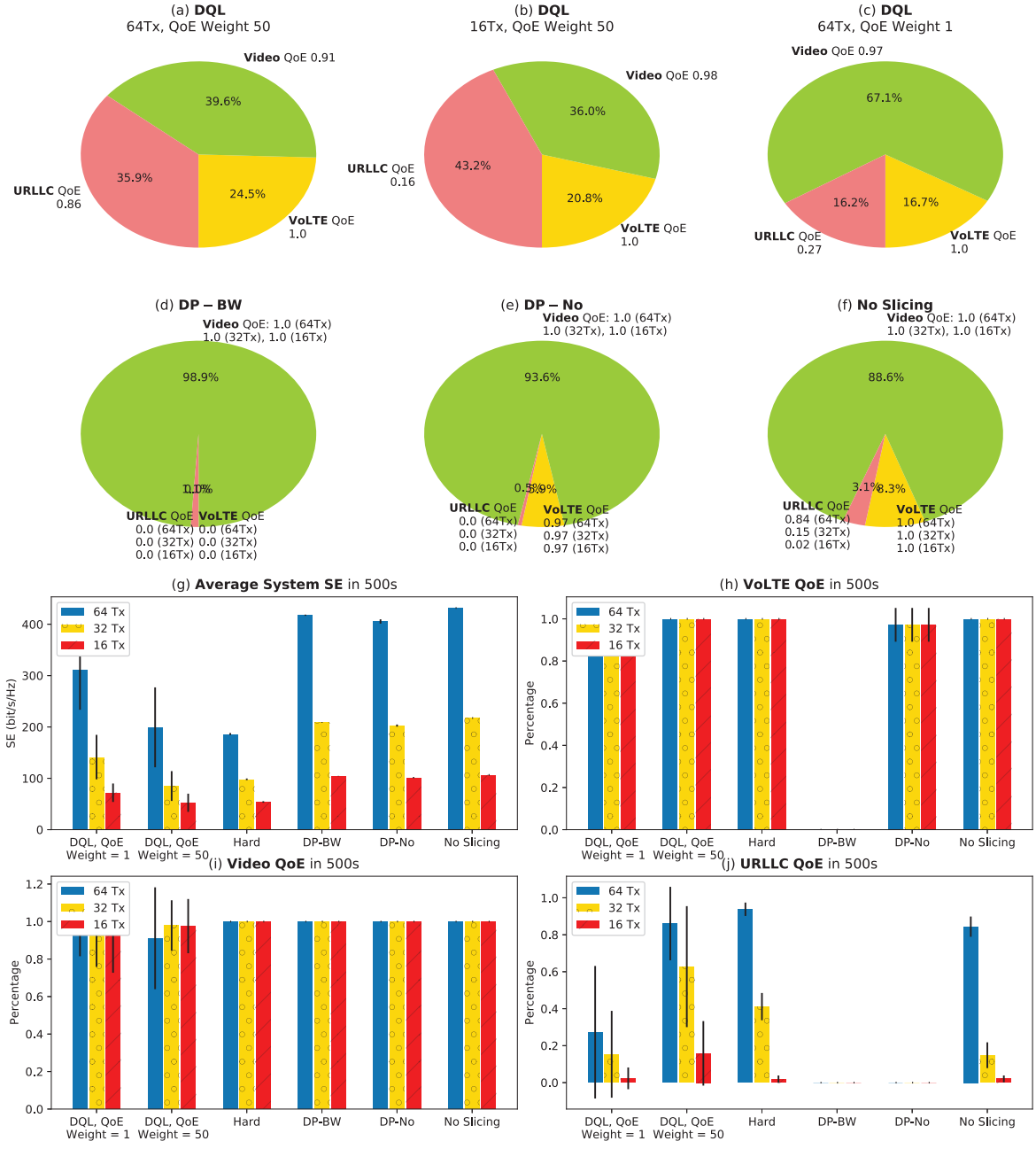


Fig. 4. The performance comparison among different schemes for radio resource slicing.

the related results is averaged by 500 Monte Carlo simulations. From Fig. 4(a)∼ 4(b), a reduction in transmission antennas from 64 to 16, which implies a decrease in network capability and an increase in potential collisions across slices, leads to a re-allocation of network bandwidth inclined to the bandwidth-consuming yet activity-limited URLLC slice. Also, it can be observed from Fig. 4(f), when the downlink transmission uses 64 antennas, "no slicing" performs the best, since the transmission capability is sufficient and the scheduling period is 0.5 ms while the dynamic resource allocation period is 1 ms and thus slower

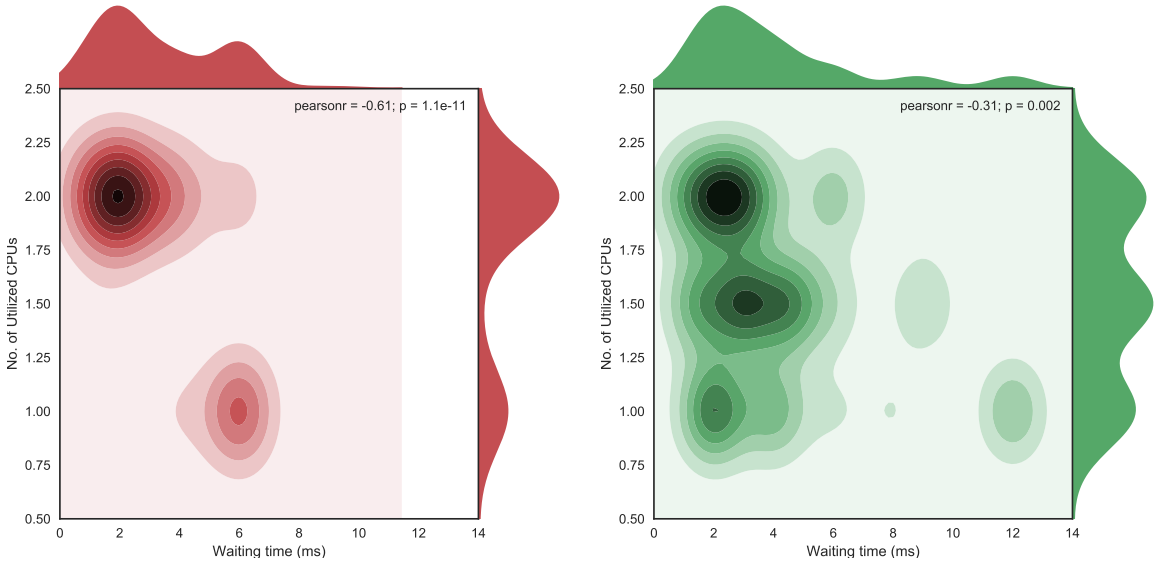
to catch the demand variations. But, when the number of downlink antenna turns to 32, the DQL-driven scheme produces 63% QoE satisfaction ratio for URLLC, while "no slicing" and "hard slicing" schemes only provision 15% and 41% satisfied URLLC packets, respectively. Meanwhile, Fig. 4(d) and Fig. 4(e) intuitively demonstrate the allocation results for the demand-prediction based schemes and show worst performance, since Fig. 3(b) and Fig. 3(t) show the number of video packets dominates the transmission and simple transmitted-packet prediction could not capture the complicated the relationship between demand and QoE. On the other hand, Fig. 4(g) illustrate that this QoE advantage of DQL comes at the cost of a decrease in SE. Recalling the definition of the reward in DQL, if we decrease the QoE weight from 50 to 1, DQL could learn a bandwidth allocation scheme (in Fig. 4(c)) yielding a larger SE yet a smaller QoE. Fig. 4(g) $\sim$ Fig. 4(j) further summarize the performance comparison and validate the DQL's flexibility and advantage in resource-limited scenarios to ensure the QoE per slice.

B. Priority-based Core Network Slicing

Section III-A has discussed how to apply DRL in radio resource slicing, based on "best effort" equal-scheduling mechanism. In this part, we further evaluate the performance of priority-based core network slicing. Specifically, we simulate with a scenario where a tenant buys limited computational resources, which correspond to 3 service function chains (SFCs) by leveraging network functionality virtualization (NFV) on top of it. In spite of possessing the same basic capability, different SFC works at the expenditure of different central processing units (CPUs) and yields different provisioning results (e.g., waiting time). Also, based on the commercial value, this tenant classifies flows from different subscribers into 3 categories (e.g., Category A, B, and C) with decreasing priority from Category A to Category C, and designs the following priority-based scheduling rule, that is, SFC I prioritizes Category A flows over the others, while SFC II equally treats Category A and B users but serves Category C flows with lower priority. SFC III treats all flows equally. Besides, SFCs process flows with equal priority according to the arrival time. The utilized CPU of each SFC depends on the number of its processed flows. Besides, SFC I, II and III cost 2, 1.5, and 1 CPU(s), but incur 10, 15, and 20 milliseconds regardless of the flow size, respectively. Hence, subject to the limited number of CPUs, users will be scheduled to an appropriate SFC, so as to incur acceptable waiting time. Therefore, the scheduling of flows should match and learn the arrival of flows in three categories, and DRL is considered as a promising solution.

Similarly, it is critical to design an appropriate mapping of DRL elements to this slicing issue. As Table I(a) implies, we use a mapping slightly different from that for radio resource slicing. In particular, we abstract the state of DRL as a summary of the category and arrival time of last five flows and the category of the newly arrived flow, while the reward is defined as the weighted summation of processing and queue time of this flow, where a larger weight is adopted to reflect the importance of higher priority. Also, we first pre-train its neural networks by emulating some flows with log-normal distributed inter-arrival time from three categories' users.

We compared the DQL scheme with an intuitive "*no priority*" solution, which allocate the flow to the SFC yielding minimum waiting time. Fig. 5 presents the kernel density estimation of 10000 Monte-Carlo simulation results, where the vertical axis represents the number of utilized CPUs, while the horizontal axis indicates the waiting time of flows. Specifically, the bi-dimensional shading color reflects the number of flows corresponding to the specific waiting time and utilized CPUS. In particular, the darker color implies larger number. It can be observed from the figure, compared with the "no priority", the DQL-empowered slicing results provision flows with smaller average waiting time area (i.e., 10.5% lower than "no priority") and significantly more sufficient CPU usage (i.e., 27.9% larger than "no priority". In other words, DQL could support alternative solutions to exploit the bought computing resources and reduce by first serving the users with higher commercial value.



(a) DQL-based Prioritied Scheduling (b) No Priority Scheduling

Fig. 5. Performance comparison between DQL-based priority scheduling and no priority scheduling for core network slicing.

IV. DQL Application Challenges in Network Slicing

The previous section has demonstrated the feasibility and advantages of DQL for resource management in network slicing. But, network slicing involves many aspects and a successful application of DQL needs some careful considerations.

A. Abstraction of States and Actions

Section III has provided two ways to abstract state and action. One way is to directly count the number of visits in each slice and adjust the allocated bandwidth, while the other tries to record the latest several visits in each slice and further determine the slice for the new visit. Both methods sound practical in the related scenarios. Hence, for new scenarios, it turns into an important issue to choose appropriate abstraction of states and actions, so as to better model the problem and save the learning cost. It remains an open question on how to give some abstraction guidelines.

B. Retrieving Rewards in Time

The simulations in Section III has ideally assumed the accurate acquirement of rewards for a state-action pair. But, such an assumption no longer holds in practical wireless environment, since it takes time for a UE to report the information and the network does not successfully receive the feedback packet due to the varying wireless channel. Also, similar to the case for state and action, the abstraction of reward might be difficult and the defined reward should be as simple as possible, so as to reduce the retrieving overhead.

C. Policy Learning Cost

The quasi-stationary nature of wireless channel and user visits requires a fast policy-learning scheme. However, the current cost of policy training still lacks the necessary learning speed. For example, our pre-training for the priority-based network slicing policy takes two days in an Intel Core i7-4712MQ processor to coverage the Q-value function. Though GPU could speed the training process, the learning cost is still heavy. Therefore, there exists a long way to design a faster policy learning scheme.

V. Conclusion

From the discussion in this article, we found that matching allocated resource to slices with the users' activity demand will be the most critical challenge for effectively realizing network slicing and deep reinforcement learning could be a promising solution. Starting with the

introduction of fundamental concept, we explained the working mechanism and application motivation of deep $\mathcal{Q}$ learning to solve this problem. We further demonstrated the advantage of deep $\mathcal{Q}$ learning in management this demand-aware resource allocation in two typical slicing scenarios including radio resource slicing and priority-based core network slicing through extensive simulations. Our results showed that compared with the demand prediction-based and some other intuitive solutions, deep $\mathcal{Q}$ learning could implicitly incorporate more deep relationship between demand (i.e., user activities) and supply (i.e., resource allocation) in resource-constrained scenarios, and enhance the effectiveness and agility for network slicing. Finally, in order to fulfill the application of deep $\mathcal{Q}$ learning in a broader sense, we talked about some noteworthy issues. We believe deep reinforcement learning could play a crucial role in network slicing in the future.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to Chen Yu and Yuxiu Hua of Zhejiang University for the valuable discussions to implement part of simulation codes.

REFERENCES

[1] K. Katsalis, N. Nikaein, E. Schiller, A. Ksentini, and T. Braun, "Network slices toward 5G communications: Slicing the LTE network," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 146–154, 2017.

[2] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, "NFV and SDN – Key technology enablers for 5G networks," *IEEE J. Sel. Area. Comm.*, vol. 35, no. 11, pp. 2468–2478, Nov. 2017.

[3] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez, "Optimising 5G infrastructure markets: The business of network slicing," in *Proc. IEEE INFOCOM 2017*, Atlanta, GA, USA, May 2017.

[4] R. Li, Z. Zhao, X. Zhou, G. Ding, Y. Chen, Z. Wang, and H. Zhang, "Intelligent 5G: When cellular networks meet artificial intelligence," *IEEE Wireless Commun.*, vol. 5, no. 24, pp. 175 – 183, Oct. 2017.

[5] X. Zhou, R. Li, T. Chen, and H. Zhang, "Network slicing as a service: Enable industries own software-defined cellular networks," *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 146–153, Jul. 2016.

[6] X. Li, M. Samaka, H. A. Chan, D. Bhamare, L. Gupta, C. Guo, and R. Jain, "Network slicing for 5G: Challenges and opportunities," *IEEE Internet Comput.*, vol. 21, no. 5, pp. 20–27, 2017.

[7] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos, "The algorithmic aspects of network slicing," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 112–119, 2017.

[8] N. Zhang, Y. F. Liu, H. Farmanbar, T. H. Chang, M. Hong, and Z. Q. Luo, "Network slicing for service-oriented networks under resource constraints," *IEEE J. Sel. Area. Comm.*, vol. 35, no. 11, pp. 2512–2521, Nov. 2017.

[9] R. Yu, G. Xue, and X. Zhang, "QoS-aware and reliable traffic steering for service function chaining in mobile networks," *IEEE J. Sel. Area. Comm.*, vol. 35, no. 11, pp. 2522–2531, Nov. 2017.

[10] R. Sutton and A. Barto, *Reinforcement learning: An introduction*. Cambridge University Press, 1998. [Online]. Available: http://webdocs.cs.ualberta.ca/~sutton/book/ebook/

[11] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. v. d. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016. [Online]. Available: https://www.nature.com/articles/nature16961

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html

[13] Y. He, Z. Zhang, F. R. Yu, N. Zhao, H. Yin, V. C. M. Leung, and Y. Zhang, "Deep-reinforcement-learning-based optimization for cache-enabled opportunistic interference alignment wireless networks," *IEEE Trans. Veh. Tech.*, vol. 66, no. 11, pp. 10 433–10 445, Nov. 2017.

[14] Z. Xu, Y. Wang, J. Tang, J. Wang, and M. C. Gursoy, "A deep reinforcement learning based framework for power-efficient resource allocation in cloud RANs," in *Proc. IEEE ICC 2017*, Paris, France, May 2017.

[15] A. Aijaz, "Hap-slicer: A radio resource slicing framework for 5G networks with haptic communications," *IEEE System J.*, vol. PP, no. 99, pp. 1–12, 2017.