Deep Reinforcement Learning With Discrete Normalized Advantage Functions for Resource Management in Network Slicing

Chen Qi, Yuxiu Hua, Rongpeng Li, Zhifeng Zhao, and Honggang Zhang

***Abstract*— Network slicing promises to provision diversified services with distinct requirements in one infrastructure. Deep reinforcement learning (e.g., deep $\mathcal{Q}$-learning, DQL) is assumed to be an appropriate algorithm to solve the demand-aware inter-slice resource management issue in network slicing by regarding the varying demands and the allocated bandwidth as the environment *state* and the *action*, respectively. However, allocating bandwidth in a finer resolution usually implies larger action space, and unfortunately DQL fails to quickly converge in this case. In this letter, we introduce discrete normalized advantage functions (DNAF) into DQL, by separating the $\mathcal{Q}$-value function as a state-value function term and an advantage term and exploiting a deterministic policy gradient descent (DPGD) algorithm to avoid the unnecessary calculation of $\mathcal{Q}$-value for every state-action pair. Furthermore, as DPGD only works in continuous action space, we embed a k-nearest neighbor algorithm into DQL to quickly find a valid action in the discrete space nearest to the DPGD output. Finally, we verify the faster convergence of the DNAF-based DQL through extensive simulations.**

***Index Terms*— Network slicing, resource management, deep reinforcement learning.**

I. Introduction

NETWORKS are becoming increasingly agile and flexible to provision diversified services with distinct requirements on latency and rate. Specifically, network slicing, which belongs to one of cutting-edge technologies in the 5G era, allows infrastructure providers to offer "slices" of resources (computational, storage and networking) with specified service license agreements (SLAs) [1]–[4]. However, in order to fully reap the desired merits like slice-level protection, envyfreeness, and load-driven elasticity [3], [4], end-to-end network slicing still faces a lot of technical challenges. For example, taking account of the limited spectrum, the slice-level protection could guarantee superior quality of experience (QoE) but also incurs degradation in spectrum efficiency (SE). Therefore, one typical question naturally arises like that how to intelligently allocate the spectrum to slices according to the dynamics of service request from mobile users in a coherent manner [5], so as to obtain satisfactory QoE in each slice at the cost of acceptable SE.

In order to address the aforementioned problem, one potential solution is to consider the (deep) reinforcement learning (RL). As a non-nascent algorithm, RL has been widely applied in the field of cognitive radio [6] and green communications [7]. The recent well-known application success in Go [8] further proves the feasibility to utilize neural networks (NN) to approximate the value functions in classical RL with case-testified convergence stability, and triggers tremendous research attention in the communications and networking area to solve resource allocation issues in some specific fields like power control, green communications, cloud radio access networks, mobile edge computing and caching [3]. But, a common problem in these works is that researchers usually assume a rather limited small discrete action space to ensure the necessary convergence rate. For example, [3] realized the spectrum allocation per slice on the unit of MegaHertz and accordingly design a DRL framework with tens of possible actions. But, such a coarse-grained spectrum allocation solution inevitably decreases the SE when some slice has very few service activities. In a word, it urgently needs a rethink on DRL to better avoid the curse of dimensionality and quickly converge in larger action space.

Overall speaking, this letter aims to answer how to allocate the limited spectrum on a finer-grained resolution across slices based on an improved DRL. In particular, we revolutionize the calculation and approximation of the $\mathcal{Q}$-value function in the deep $\mathcal{Q}$-learning (DQL) as follows:

- Inspired by the normalized advantage functions (NAF) model [9], [10], we design a discrete NAF (DNAF) NN to separately approximate a state-value function term $V(\boldsymbol{s})$ and an advantage term $A(\boldsymbol{s}, \boldsymbol{a})$, where $\boldsymbol{s}$ and $\boldsymbol{a}$ denote a state and an action respectively. Moreover, we have $Q(\boldsymbol{s}, \boldsymbol{a}) = V(\boldsymbol{s}) + A(\boldsymbol{s}, \boldsymbol{a})$. Hence, the common part of the $\mathcal{Q}$-value function could be learnt across all actions.
- We utilize a deterministic policy gradient descent (DPGD)-based $\mathcal{Q}$-learning [16] to replace the classical statistical policy gradient descent-based DQL, so as to directly yield the most suitable action for a specific state.
- In order to solve the issue that a DPGD method ignores the discreteness of the action space, we firstly output a proto action with the largest $\mathcal{Q}$-value in the virtual continuous action space and then scramble it with an extra noise term. Finally, we embed a k-nearest neighbor (k-nn) algorithm to quickly select the closest discrete action.

The remainder of the letter is organized as follows: Section II talks about some necessary algorithmic background and formulates the system model. Section III gives the details of the DNAF based $\mathcal{Q}$-Learning, while Section IV presents the

Manuscript received May 14, 2019; accepted June 9, 2019. Date of publication June 14, 2019; date of current version August 12, 2019. This work was supported by National Natural Science Foundation of China (No. 61701439, 61731002), Zhejiang Key R&D Plan (No. 2019C01002), the Fundamental Research Funds for the Central Universities. The associate editor coordinating the review of this letter and approving it for publication was N. Lee. *(Corresponding author: Rongpeng Li.)*

The authors are with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China (e-mail: qichen7c@zju.edu.cn; 21631087@zju.edu.cn; lirongpeng@zju.edu.cn; zhaozf@zju.edu.cn; honggangzhang@zju.edu.cn).

Digital Object Identifier 10.1109/LCOMM.2019.2922961

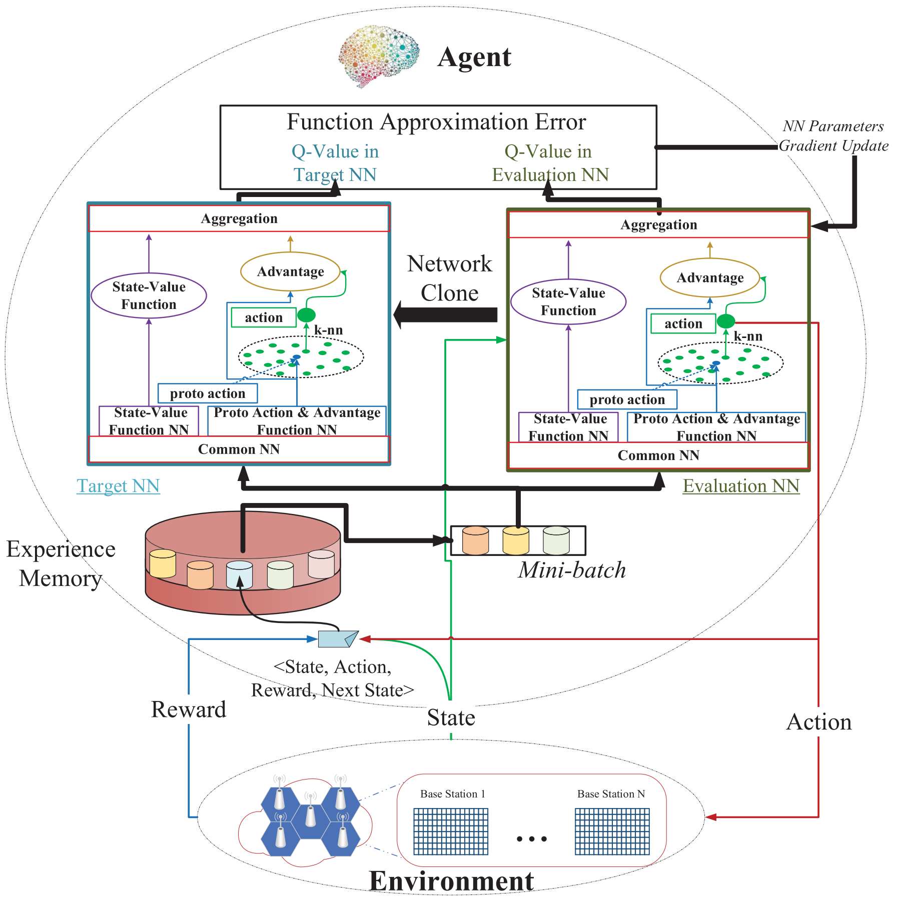

Fig. 1. An illustration of DNAF-based $\mathcal{Q}$-learning for resource management in network slicing.

related simulation results. Finally, Section V concludes the letter with a summary.

II. Mathematical Background and System Model

A. Mathematical Background

RL tries to find a strategy π, which maps a state $\boldsymbol{s} \in \mathbb{S}$ (i.e., the varying traffic per slice) to an action $\pi(\boldsymbol{s})$, (i.e., $\boldsymbol{a} \in \mathbb{A}$, allocated bandwidth per slice) to maximize the discounted accumulative reward starting from the state $\boldsymbol{s}^{(0)} = \boldsymbol{s}$. Formally, this accumulative reward is called as a state-value function, which can be calculated by [11]

$$V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R(\boldsymbol{s}, \pi(\boldsymbol{s}) | \boldsymbol{s}^{(0)} = \boldsymbol{s})\right] \tag{1}$$

where the positive parameter γ is the discount factor that maps the future reward to the current state. Given the diminishing importance of future cost than the current one, γ is smaller than 1.

$\mathcal{Q}$-learning is an RL technique to obtain the strategy π. Specifically, a $\mathcal{Q}$-learning agent attempts to learn the value of taking a specific action for a given state, i.e. Q-value, by constantly updating Q value in a temporal difference (TD) manner as

$$Q(\boldsymbol{s}, \boldsymbol{a}) \leftarrow Q(\boldsymbol{s}, \boldsymbol{a}) + \alpha\big(R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \max_{\boldsymbol{a}' \in \mathbb{A}} Q(\boldsymbol{s}', \boldsymbol{a}') - Q(\boldsymbol{s}, \boldsymbol{a})\big) \tag{2}$$

where α is the learning rate and $\boldsymbol{s}'$ denotes the state of the environment after taking action $\boldsymbol{a}$ at state $\boldsymbol{s}$.

In recent years, [8] proposes to use NNs to approximate $\mathcal{Q}$-value function [12] so as to solve an RL problem with a tremendous state dimension. Mathematically, DQL trains an NN with parameters $\boldsymbol{\theta}$ by minimizing the loss function between the real $\mathcal{Q}$-value function $Q(\boldsymbol{s}, \boldsymbol{a})$ and an NN-approximated one $Q^+(\boldsymbol{s}, \boldsymbol{a}|\boldsymbol{\theta})$, which can be formulated as

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}'} L(\boldsymbol{\theta}') = \arg\min_{\boldsymbol{\theta}'} \left(Q(\boldsymbol{s}, \boldsymbol{a}) - Q^+(\boldsymbol{s}, \boldsymbol{a}|\boldsymbol{\theta}')\right)^2 \tag{3}$$

Commonly, $\boldsymbol{\theta}$ could be achieved by a gradient-based approach as

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla L(\boldsymbol{\theta}) \tag{4}$$

In addition, there are some tricks that can improve the performance of DQL, such as replay buffer [13], target network [14], prioritized replay [15], etc.

B. System Model

We consider an access network scenario in Fig. 1 consisting of multiple base stations (BSs), where there exists a list of existing slices $1, \cdots, N$ sharing the aggregated bandwidth W and having fluctuating demands $\boldsymbol{d} = (d_1, \cdots, d_N)$. We aim to maximize the expectation of the utility function $\mathbb{E}\{R(\boldsymbol{d}, \boldsymbol{w})\}$ by dynamically adjusting the allocated bandwidth

$\boldsymbol{w} = (w_1, \cdots, w_N)$ to each slice, where the notation $\mathbb{E}(\cdot)$ denotes to take the expectation of the argument. Moreover, the utility function is defined as the weighted sum of SE and QoE satisfaction ratio. Mathematically,

$$R = \zeta \cdot \text{SE} + \beta \cdot \text{QoE} \tag{5}$$

where ζ and β denotes the importance of SE and QoE. Our goal is to allocate the bandwidth to slices according to the traffic variations within each slice, that is,

$$\begin{aligned}
&\arg\max_{\boldsymbol{w}} \mathbb{E}_t\{R(\boldsymbol{d}, \boldsymbol{w})\} \\
&\quad = \arg\max_{\boldsymbol{w}} \mathbb{E}\big\{\zeta \cdot \text{SE}(\boldsymbol{d}, \boldsymbol{w}) + \beta \cdot \text{QoE}(\boldsymbol{d}, \boldsymbol{w})\big\} \\
&\qquad \text{s.t.:}\ \ \boldsymbol{w} = (w_1, \cdots, w_N) \\
&\qquad\qquad w_1 + \cdots + w_N = W \\
&\qquad\qquad w_i = k \cdot \Delta, \quad \forall i \in [1, \cdots, N] \\
&\qquad\qquad \boldsymbol{d} = (d_1, \cdots, d_N) \\
&\qquad\qquad d_i \sim \text{Certain Traffic Model}, \quad \forall i \in [1, \cdots, N]
\end{aligned} \tag{6}$$

where t denotes the temporal index, k is an integer and Δ is the minimum allocated bandwidth per slice. Notably, $\boldsymbol{d}(t)$ depends on both $\boldsymbol{d}(t-1)$ and $\boldsymbol{w}(t-1)$, since the maximum sending capacity of servers belonging to one service is tangled with the provisioning capabilities for this service. For example, the TCP sending window size is influenced by the estimated channel throughput.

The key challenge to solve (6) lies in the volatile demand variations without having known a priori due to the traffic model. Hence, DQL promises to be an appropriate solution to solve this problem. But, DQL converges slowly for large action space, since DQL needs to predict $\mathcal{Q}$-values for each state-action pair. Unfortunately, the size of action space $|\mathbb{A}|$ increases exponentially along with the decrease of Δ and the increase of N, since we have

$$|\mathbb{A}| = \binom{\lfloor \frac{W}{\Delta} \rfloor - 1}{N - 1} \tag{7}$$

Therefore, it is inevitable to revolutionize the classical DQL.

III. The DNAF Based $\mathcal{Q}$-Learning

Researchers in [9] and [10] has suggested NAF as a potential solution to DQL with continuous action space by decomposing $\mathcal{Q}$-value into a state-value function V and an advantage function A. Since the discreteness of action space for resource management in network slicing is different from the continuity in [9] and [10], it is quite meaningful to re-investigate its effectiveness here.

Inspired by [9], [10], we build two separate NNs for the state-value function $V(\boldsymbol{s}|\boldsymbol{\theta}^V)$ parameterized by $\boldsymbol{\theta}^V$ and the advantage function $A(\boldsymbol{s}, \boldsymbol{a}|\boldsymbol{\theta}^A)$ parameterized by $\boldsymbol{\theta}^A$, on top of a common NN collecting some general information, that is,

$$Q(\boldsymbol{s}, \boldsymbol{a}|\boldsymbol{\theta}^Q) = V(\boldsymbol{s}|\boldsymbol{\theta}^V) + A(\boldsymbol{s}, \boldsymbol{a}|\boldsymbol{\theta}^A) \tag{8}$$

Moreover, the advantage function approximation leverages the DPGD algorithm [16] to obtain the proto action $\boldsymbol{\mu}(\boldsymbol{s}|\boldsymbol{\theta}^{\boldsymbol{\mu}})$ by

Algorithm 1 The DNAF Based $\mathcal{Q}$-Learning

```
1: Randomly initialize a normalized $\mathcal{Q}$ network $Q(\boldsymbol{s}, \boldsymbol{a}|\boldsymbol{\theta}^Q)$,
   where $\boldsymbol{\theta}^Q$ is a concatenation of $\boldsymbol{\theta}^V$ and $\boldsymbol{\theta}^A$.
2: Initialize a target network $\mathcal{Q}'$ with weight $\boldsymbol{\theta}^{Q'} \leftarrow \boldsymbol{\theta}^Q$.
3: Initialize replay buffer $\mathbb{R} \leftarrow \varnothing$.
4: Initialize an episode index $t = 0$.
5: repeat
6:    At episode $t$, the agent observes the state $\boldsymbol{s}_t$.
7:    The agent calculates a proto-action $\hat{\boldsymbol{a}}_t = \boldsymbol{\mu}(\boldsymbol{s}_t|\boldsymbol{\theta}^{\mu}) + \mathcal{N}_t$
      and determines the closest action $\boldsymbol{a}_t = g_1(\hat{\boldsymbol{a}}_t)$ by (10).
8:    The agent receives the reward $R(\boldsymbol{s}_t, \boldsymbol{a}_t)$ and observes a
      new state $\boldsymbol{s}_{t+1}$ for the environment.
9:    The agent stores the episode experience $(\boldsymbol{s}_t, \boldsymbol{a}_t, R_t, \boldsymbol{s}_{t+1})$
      in $\mathbb{R}$.
10:   The agent samples a mini-batch $\mathbb{R}_{\text{mbatch}}$ of experiences
      from $\mathbb{R}$.
11:   The agent sets $y_i = R_i + \gamma V'(\boldsymbol{s}_{i+1}|\boldsymbol{\theta}^{Q'})$ and gets
      estimated $\mathcal{Q}$-value $Q_i = Q(\boldsymbol{s}_i, \boldsymbol{a}_i|\boldsymbol{\theta}^Q))$ by (5) and (6),
      $\forall(\boldsymbol{s}_i, \boldsymbol{a}_i, R_i, \boldsymbol{s}_{i+1}) \in \mathbb{R}_{\text{mbatch}}$.
12:   The agent updates the weights $\boldsymbol{\theta}^Q$ for the evaluation
      network by leveraging gradient descent algorithm to
      $\frac{1}{|\mathbb{R}_{\text{mbatch}}|}\sum_i((y_i - Q_i)^2)$.
13:   The agent clones the $\mathcal{Q}$ network to the target network $\mathcal{Q}'$
      every $C$ episodes by assigning the weights $\mathcal{Q}'$ as $\boldsymbol{\theta}^{Q'} =$
      $\boldsymbol{\theta}^Q$.
14:   The episode index is updated by $t \leftarrow t + 1$.
15: until A predefined stopping condition (e.g., the gap
    between $y_i$ and $Q_i$, the episode length, etc) is satisfied.
```

an parameterized by $\boldsymbol{\theta}^{\boldsymbol{\mu}}$ by

$$\begin{aligned}
A(\boldsymbol{s}, \boldsymbol{a}|\boldsymbol{\theta}^A) = -\frac{1}{2}(\boldsymbol{a} - \boldsymbol{\mu}(\boldsymbol{s}|\boldsymbol{\theta}^{\mu}))^T \Lambda(\boldsymbol{s}|\boldsymbol{\theta}^L) \\
\times \Lambda(\boldsymbol{s}|\boldsymbol{\theta}^L)^T (\boldsymbol{a} - \boldsymbol{\mu}(\boldsymbol{s}|\boldsymbol{\theta}^{\mu}))
\end{aligned} \tag{9}$$

where $\Lambda(\boldsymbol{s}|\boldsymbol{\theta}^L)$ is a low-triangular matrix whose entries come from a linear output layer of an NN parameterized by $\boldsymbol{\theta}^L$ [9]. Hence, $\boldsymbol{\theta}^A$ is a concatenation of $\boldsymbol{\theta}^{\boldsymbol{\mu}}$ and $\boldsymbol{\theta}^L$.

Since the action $\boldsymbol{\mu}(\boldsymbol{s}|\boldsymbol{\theta}^{\boldsymbol{\mu}})$ might be an invalid action in the discrete action space $\mathbb{A}$, we first calculate the proto action $\hat{\boldsymbol{a}}$[1] by adding $\boldsymbol{\mu}(\boldsymbol{s}|\boldsymbol{\theta}^{\boldsymbol{\mu}})$ with a noise term. Notably, the added noise in selecting action procedure could be regarded as an alternative means to the ϵ-greedy strategy in classical RL for state exploration. Afterwards, k-nn is applied to find the closest valid action, that is,

$$g_k(\boldsymbol{a}) = \arg\min_{\boldsymbol{a}' \in \mathbb{A}} \|\boldsymbol{a}' - \hat{\boldsymbol{a}}\|_2 \tag{10}$$

In other words, the function $g_k(\cdot)$ is a k-nn mapping from a continuous space to a discrete set and returns k valid actions closest to the proto action. For $k > 1$, it also means to obtain the k nearest actions that maximize $\mathcal{Q}$-value [17]. The case $k = 1$, which belongs to the key focus in this letter, is equivalent to simply select the nearest action.

We incorporate the aforementioned methods into the DQL and have the DNAF-based $\mathcal{Q}$-learning in Algorithm 1.

[1]As depicted in Fig. 1, the proto action is plotted in the blue circle while the valid actions are presented in green ones.

TABLE I
A BRIEF SUMMARY OF KEY SETTINGS FOR TRAFFIC GENERATION PER SLICE

	VoLTE	Video	URLLC
Bandwidth	10 MHz		
Scheduling	Round robin per slot (0.5 ms)		
Slice Band Adjustment (Q-Value Update)	1 second (2000 scheduling slots)		
Channel	Rayleigh fading		
User No. (100 in all)	46	46	8
Distribution of Inter-Arrival Time per User	Uniform [Min = 0, Max = 160ms]	Truncated Pareto [Exponential Para = 1.2, Mean = 6 ms, Max = 12.5 ms]	Exponential [Mean = 180 ms]
Distribution of Packet Size	Constant (40 Byte)	Truncated Pareto [Exponential Para = 1.2, Mean = 100 Byte, Max = 250 Byte]	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

IV. SIMULATION RESULTS AND NUMERICAL ANALYSES

In this part, we compare the convergence rate of the DNAF-based DQL and classical DQL. We simulate in one single BS scenario with three types of services (i.e., VoLTE, video, ultra-reliable low-latency communications (URLLC)) as in [3], [18] and correspondingly have three slices. Moreover, we attempt to allocate 10-MegaHertz bandwidth to these three slices and set the minimal bandwidth allocation resolution is 0.2-MegaHertz, thus leading to 1176 valid actions. Meanwhile, the network slice stops sending new packets to one user if 5 packets in the caching buffer for this user have not been successfully delivered or expired (e.g., exceeding the tolerant delay for that slice). Otherwise, each network slice sends traffic to its user following the settings in Table I.

Fig. 2 gives an illustration of the reward variations with respect to the episode index. Here, the reward is defined in (5). As for the DNAF-based DQL, inspired by the noise settings in [13], we first assume the noise obeys normal distribution and is multiplied by an attenuating coefficient, which gradually decays with the number of iterations and ultimately fixes at zero after 3000 iterations. It can be observed from Fig. 2 that regardless of the values of α, the DNAF-based DQL could converge after 4000 - 6000 episodes, while the classical DQL still changes dramatically with no sign of convergence even after 10000 episodes. Therefore, it could safely come to the conclusion that the DNAF-based DQL could converge more rapidly than the classical DQL. On the other hand, Fig. 2 also provides the performance result of the equal-allocation strategy where we intuitively allocate the bandwidth according to the number of slices and verified that the DNAF-based DQL yield superior performance than the equal-allocation strategy. Furthermore, Fig. 3 gives the cumulative reward of the DNAF-based DQL, classical DQL, equal-allocation

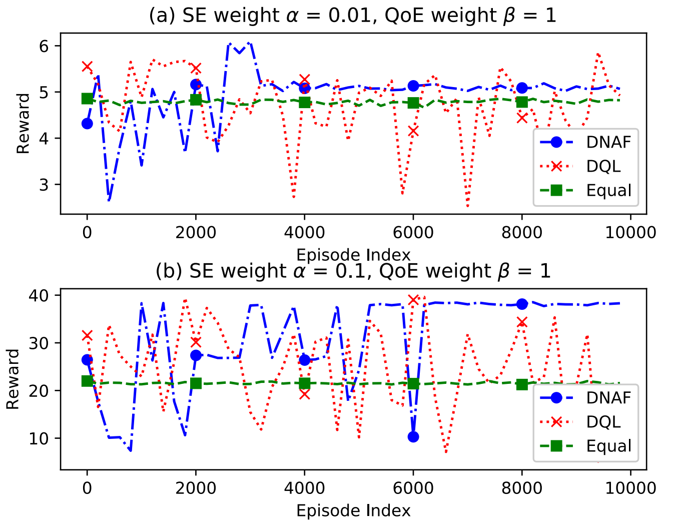

Fig. 2. The convergence rate of the DNAF-based DQL and classical DQL.

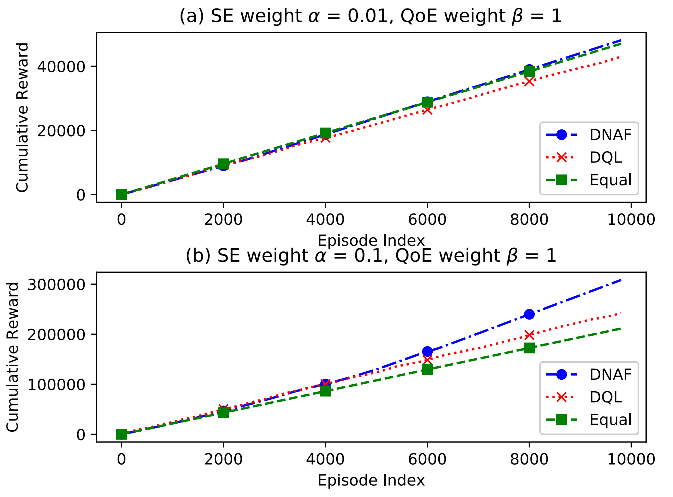

Fig. 3. The cumulative reward of the DNAF-based DQL, classical DQL, equal-allocation strategy.

strategy. It can be found in Fig. 3 that the DNAF-based DQL could obtain superior performance than the equal-allocation strategy while DQL yield some inferior performance than the equal-allocation strategy due to its slow convergence rate.

Fig. 4 presents the SE and QoE satisfaction ratio applying the learnt policy after 10000 episodes. It can be observed that when $\alpha = 0.01$, which implies that SE is on a par with QoE satisfaction ratio, the DNAF-based DQL could yield superior performance on SE and QoE satisfaction ratio simultaneously than the classical DQL. On the other hand, when α takes a larger value (i.e., 0.1) to put more focus on SE, compared than the classical DQL, the DNAF-based DQL learns a policy giving significantly higher SE but degrades the QoE satisfaction ratio for some slices. Notably, some evaluation metrics produced by the DNAF-based DQL policy are not always superior (or even inferior) to the classical DQL, since it is still a very challenging research topic to design RL with multiple conflicting rewarding metrics. In this letter, we simply choose the weighted sum of two conflicting metrics (i.e., SE and QoE) as the reward in RL. Despite the intuitiveness of this direct

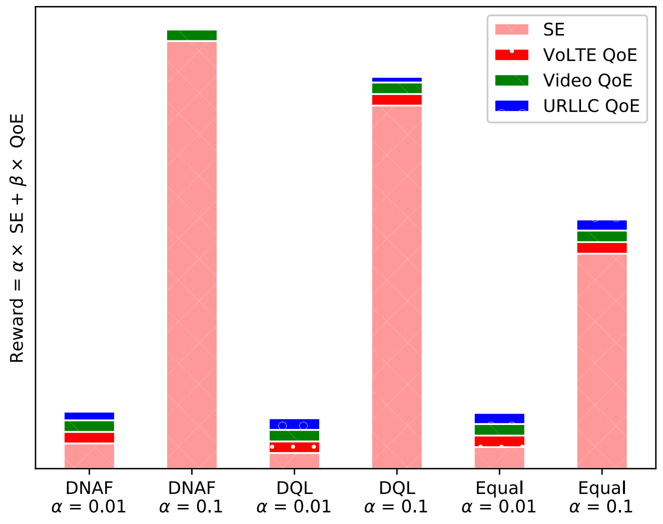

Fig. 4. The snapshot of SE and QoE satisfaction ratio applying the learnt policy after 10000 episodes.

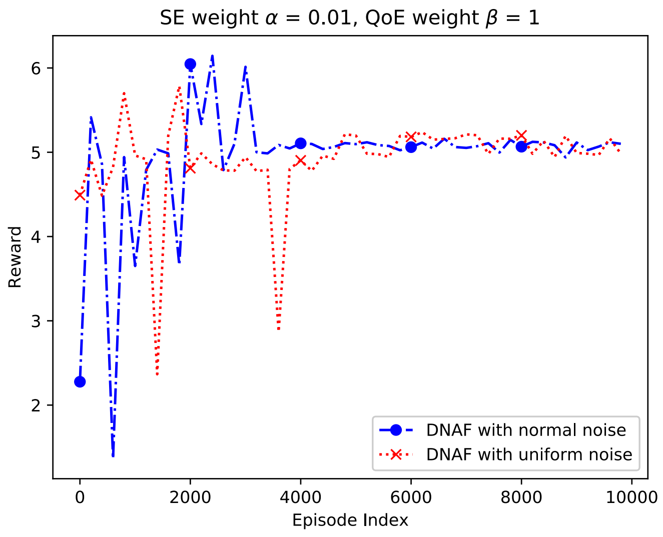

Fig. 5. The reward of the DNAF-based DQL with the action noise obeying normal distribution and uniform distribution.

summation, our simulation results have demonstrated that we cannot guarantee to simultaneously obtain superior performance for both SE and QoE. Therefore, we have left this inspiring and interesting topic as our future works.

Fig. 5 shows the comparison between normal distributed noise and uniform distributed ones, respectively. It can be observed that both cases could lead to convergent learning policy and exhibit trivial performance difference.

V. Conclusion

In this letter, we have discussed how to accelerate the convergence rate of the classical DQL in large action space, so as to satisfy the requirements for finer-resolution resource management in network slicing. In particular, we have applied the DNAF into DQL, by separating the $\mathcal{Q}$-value function as a state-value function term and an advantage term and exploiting a DPGD algorithm to avoid the unnecessary calculation of $\mathcal{Q}$-value for every state-action pair. Furthermore, we have embedded a k-nn algorithm into DQL to quickly find a valid action in the discrete action space. We have also verified that compared than the classical DQL, the DNAF-based DQL exhibits faster convergence and superior performance. Hence, we believe our works could contribute to enhancing the applicability of DQL in network slicing. However, there still exist some research issues to be solved, in particular the need to further improve DQL to guarantee minimal slice SLAs, capably adapt non-stationary traffic demands, and smartly design the reward for multi-conflicting metrics.

References

[1] X. Zhou *et al.*, "Network slicing as a service: Enabling enterprises' own software-defined cellular networks," *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 146–153, Jul. 2016.
[2] R. Li *et al.*, "Intelligent 5G: When cellular networks meet artificial intelligence," *IEEE Wireless Commun.*, vol. 24, no. 5, pp. 175–183, Oct. 2017.
[3] R. Li *et al.*, "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74429–74441, 2018.
[4] J. Zheng and G. de Veciana, "Elastic multi-resource network slicing: Can protection lead to improved performance," Jan. 2019, *arXiv:1901.07497*. [Online]. Available: https://arxiv.org/abs/1901.07497
[5] S. Vassilaras *et al.*, "The algorithmic aspects of network slicing," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 112–119, Aug. 2017.
[6] X. Chen *et al.*, "Stochastic power adaptation with multiagent reinforcement learning for cognitive wireless mesh networks," *IEEE Trans. Mobile Comput.*, vol. 12, no. 11, pp. 2155–2166, Nov. 2013.
[7] R. Li *et al.*, "TACT: A transfer actor-critic learning framework for energy saving in cellular radio access networks," *IEEE Trans. Wireless Commun.*, vol. 13, no. 4, pp. 2000–2011, Apr. 2014.
[8] V. Mnih *et al.*, "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
[9] S. Gu *et al.*, "Continuous deep Q-learning with model-based acceleration," Mar. 2016, *arXiv:1603.00748*. [Online]. Available: http://arxiv.org/abs/1603.00748
[10] Z. Wang *et al.*, "Dueling network architectures for deep reinforcement learning," Nov. 2015, *arXiv:1511.06581*. [Online]. Available: http://arxiv.org/abs/1511.06581
[11] R. S. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, U.K.: Cambridge Univ. Press, 1998. [Online]. Available: http://webdocs.cs.ualberta.ca/sutton/book/ebook/
[12] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.
[13] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," Sep. 2015, *arXiv:1509.02971*. [Online]. Available: http://arxiv.org/abs/1509.02971
[14] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, Mar. 2016, pp. 1–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=3016100.3016191
[15] T. Schaul *et al.*, "Prioritized experience replay," Nov. 2015, *arXiv:1511.05952*. [Online]. Available: http://arxiv.org/abs/1511.05952
[16] D. Silver *et al.*, "Deterministic policy gradient algorithms," in *Proc. ICML*, Jun. 2014, pp. 1–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=3044805.3044850
[17] G. Dulac-Arnold *et al.*, "Deep reinforcement learning in large discrete action spaces," Dec. 2015, *arXiv:1512.07679*. [Online]. Available: http://arxiv.org/abs/1512.07679
[18] NGMN. (Oct. 2007). *NGMN Radio Access Performance Evaluation Methodology*. [Online]. Available: https://www.ngmn.org/fileadmin/user_upload/NGMN_Radio_Access_Performance_Evaluation_Methodology.pdf