GAN-based Deep Distributional Reinforcement Learning for Resource Management in Network Slicing

Yuxiu Hua*, Rongpeng Li*, Zhifeng Zhao*, Honggang Zhang*, and Xianfu Chen†
*College of Information Science and Electronic Engineering
Zhejiang University, Zheda Road 38, Hangzhou 310027, China
Emails:{21631087, lirongpeng, zhaozf, honggangzhang}@zju.edu.cn
†VTT Technical Research Centre of Finland, P.O. Box 1100, FI-90571 Oulu, Finland
Email:Xianfu.Chen@vtt.fi

***Abstract*—Network slicing is a key technology in 5G communications system, which aims to dynamically and efficiently allocate resources for diversified services with distinct requirements over a common underlying physical infrastructure. Therein, demand-aware allocation is of significant importance to network slicing. In this paper, we consider a scenario that contains several slices in one base station on sharing the same bandwidth. Deep reinforcement learning (DRL) is leveraged to solve this problem by regarding the varying demands and the allocated bandwidth as the environment *state* and *action*, respectively. In order to obtain better quality of experience (QoE) satisfaction ratio and spectrum efficiency (SE), we propose generative adversarial network (GAN) based deep distributional Q network (GAN-DDQN) to learn the distribution of state-action values. Furthermore, we estimate the distributions by approximating a full quantile function, which can make the training error more controllable. In order to protect the stability of GAN-DDQN's training process from the widely-spanning utility values, we also put forward a reward-clipping mechanism. Finally, we verify the performance of the proposed GAN-DDQN algorithm through extensive simulations.**

***Index Terms*—network slicing, deep reinforcement learning, distributional reinforcement learning, generative adversarial network, 5G**

I. Introduction

The fifth-generation (5G) mobile systems, armed with novel network architecture and emerging technologies, are expected to provide a feasible and efficient scheme to meet existing challenges [1], [2]. Specifically, it is envisioned that 5G systems cater to a wide range of services differing in their requirements and types of devices, and going beyond the traditional human-type communications to include various kinds of machine-type communications [3]. To efficiently accommodate diversified services and use cases over a common underlying physical infrastructure, it becomes a consensus that network slicing is the key enabling technology [1]. Network slicing aims at supporting on-demand tailored services for distinct application scenarios by slicing the whole network into different parts, dynamically and efficiently allocating network resources to each slice in a coherent manner. However, reaping the desired merits is still a challenging technical task [4]. For example, taking account of the limited spectrum, the slice-level protection is in a dilemma as to whether to guarantee superior quality of experience (QoE) or 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 coherently [5], so as to obtain satisfactory QoE in each slice at the cost of acceptable SE.

In order to address the demand-aware resource allocation problem, one potential solution is reinforcement learning (RL). RL is an important type of machine learning where an agent learns how to perform actions in an environment by observing states and obtaining rewards. In RL, the state-action value, $Q(s,a)$, describes the expected return, or discounted sum of rewards when performing action a in state s. Usually, the state-action value can be estimated by classic value-based methods such as SARSA [6] and Q-learning [7] based on Bellman equation. [8] used deep neural networks to approximate Q function, namely deep Q network (DQN), which demonstrated to achieve human-level performance on simple computer games and inspired a research wave of deep reinforcement learning (DRL). Besides modeling the expected return, i.e. $Q(s,a)$, [9] showed that we can describe the distribution of $Q(s,a)$ by the distributional analogue of Bellman equation. From the distributional perspective, [9] proposed C51 algorithm, which takes advantage of Kullback–Leibler (KL) divergence to minimize the difference between the approximate Q distribution and the Q distribution calculated by distributional Bellman optimality operator. [10] proposed an alternative algorithm, quantile regression DQN (QR-DQN), which is inspired by the theory of quantile regression [11], and as a result successfully performed distributional reinforcement learning over the Wasserstein metric, leading to state-of-the-art performance. Given the reputation of generative adversarial

This work was supported in part by National Natural Science Foundation of China (No. 61701439, 61731002), Zhejiang Key Research and Development Plan (No. 2019C01002), the Fundamental Research Funds for the Central Universities (2019QNA5010).

978-1-7281-0962-6/19/$31.00 ©2019 IEEE

network (GAN) to approximate one distribution [20], it naturally raises a question whether GAN can be an alternative scheme to approximate the distribution of the action values and improve distributional RL.

On the other hand, DRL has triggered tremendous research attention in the communications and networking area to solve resource allocation issues in some specific fields like power control [14], green communications [15], cloud radio access networks [16], mobile edge computing and caching [17]. In this work, we aim to design a new framework based on distributional RL and WGAN-GP to realize dynamic and efficient spectrum allocation per slice, reaping better SE and QoE than the scheme utilizing the conventional DQN. The main contributions of this work are as follows:

- We design the GAN-based deep distributional Q network (GAN-DDQN), where the generator network outputs a set of samples for each action that describe the distribution of state-action values, differing from [13] where the generator network directly outputs state-action values.
- Inspired by IQN [12], we adopt a method to learn the full quantile function, so as to reduce the impact of the number of quantiles on model performance. In particular, we regard the quantiles sampled from a base distribution (e.g. $U[0,1]$) as a part of the generator network's input and embed the quantiles to estimate the expectation of state-action values.
- In order to protect the stability of GAN-DDQN's training process from the widely-spanning utility values, we design a reward-clipping mechanism. Specifically, we clip the weighted sum of SE and QoE to three small values (e.g., -1, 0 and 1) by setting two adjustable thresholds, and then take these values as the rewards used in RL procedure.

The remainder of the paper is organized as follows: Section II talks about some necessary mathematical background and formulates the system model. Section III gives the details of the GAN-DDQN, while Section IV presents the related simulation results. Finally, Section V summarizes the paper and gives future prospects.

II. Mathematical Background and System Model

A. Mathematical Background

An agent tries to find the optimal behavior in a given setting through interaction with the environment, which can be treated as solving an RL problem. This interactive process can be modeled as a Markov Decision Process $(\mathcal{S}, \mathcal{A}, R, P, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, $R(s,a)$ is the reward function, $P(\cdot|s,a)$ is the transition probability, and γ is a discount factor. A policy $\pi(\cdot|s)$ maps a state to a distribution over actions. In order to intuitively compare actions in a given state, state-action value function is defined, which is the expected sum of discounted reward in state s and action a, formulated as $Q^\pi(s,a) = \mathbb{E}_{\pi,P}\left[\sum_{t=0}^{\infty}\gamma^t R\left(s_t,a_t\right)|s_0=s,a_0=a\right]$, and can be characterized by the Bellman equation

$$Q^\pi(s,a) = \mathbb{E}_{\pi,P}\left[R(s,a) + \gamma Q^\pi\left(s',a'\right)\right] \tag{1}$$

where s' and a' can be derived from $P(\cdot|s,a)$ and $\pi(\cdot|s')$, respectively.

The goal of RL is to find the optimal policy π^* which yields the maximum $Q(s,a)$ for all s, a. In Q-learning [7], the state-action value function can be approximated by a parameterized function Q_θ (e.g., a neural network), and trained by minimizing the squared temporal difference (TD) error

$$\delta^2 = \left[r + \gamma \max_{a'\in\mathcal{A}} Q_\theta\left(s',a'\right) - Q_\theta\left(s,a\right)\right]^2 \tag{2}$$

over samples (s,a,r,s') randomly selected from replay buffer [18] that stores transitions while an agent interacts with an environment following a policy based on Q_θ. Table I lists the important notations used in this paper.

1) Distributional Reinforcement Learning: The main idea of distributional RL [9] is to work directly with the distribution of returns rather than with theirs expectation Q^π, so as to improve sample complexity and increase robustness to hyperparameter variation [19]. Let the random variable $Z^\pi(s,a)$ be the return obtained by starting from state s, performing action a and following the current policy π, then we arrive to $Q^\pi(s,a) = \mathbb{E}\left[Z^\pi(s,a)\right]$ and an analogous distributional Bellman equation:

$$Z^\pi(s,a) \overset{D}{=} R(s,a) + \gamma Z^\pi\left(s',a'\right) \tag{3}$$

where $A \overset{D}{=} B$ denotes that random variable A has the same probability law as B. Therefore, a distributional Bellman optimality operator $\mathcal{T}$ can be defined by

$$\mathcal{T}Z(x,a) \overset{D}{=} R(s,a) + \gamma Z\left(s', \arg\max_{a'\in\mathcal{A}} \mathbb{E}Z\left(s',a'\right)\right) \tag{4}$$

TABLE I
Notations Used in This Paper

Notation	Definition
$\mathcal{S}$	State space
$\mathcal{A}$	Action space
s or s' or s_t	State
a or a' or a_t	Action
R	Reward function
r or r_t	Reward
P	Transition probability
γ	Discount factor
π	Policy
Q	State-action value function
Z	Random variable to statistically model state-action value
$\mathcal{T}$	Bellman optimality operator
α	Weight of the SE
β	Weight of the QoE
τ	Quantile samples
λ	Gradient penalty coefficient

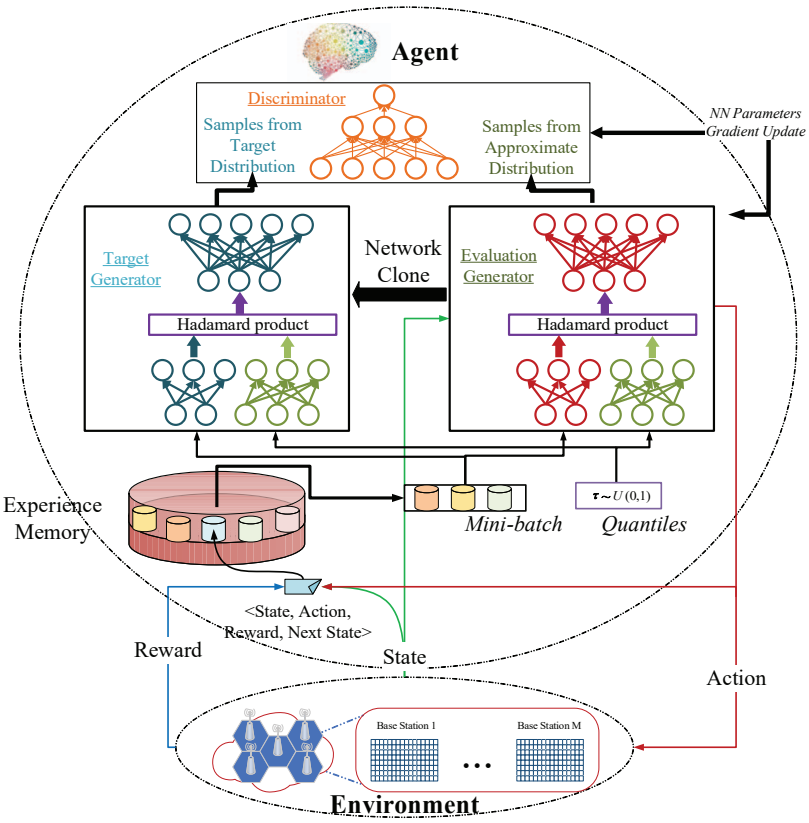

Fig. 1. An illustration of GAN-DDQN for resource management in network slicing.

In traditional RL algorithms, we seek the optimal Q function approximator by minimizing δ^2 in Eq. (2), which is an operation on scale values. In distributional RL, our objective is to minimize a distributional error:

$$\sup_{s,a} \text{dist}\left(\mathcal{T}Z(s,a), Z(s,a)\right) \tag{5}$$

where $\text{dist}(A,B)$ denotes the distance between random variable A and B, which can be measured by many metrics, such as p-Wasserstein [10], KL divergence [9], etc. In [9], Bellemare *et al.* proved the distributional Bellman equation is a contraction in the p-Wasserstein distance, while the Bellman optimality operator $\mathcal{T}$ is not necessarily a contraction, which provides a guideline for metric selection.

2) Generative Adversarial Network: The main focus of GAN [20] is to learn the distribution of data from all domains, mostly image, music, text, etc., so as to generate convincing data. GAN composes of two neural networks, a generator network and a discriminator network, which play a zero-sum game against each other. The generator network G takes an input from a random distribution, and maps it to the space of real data. The discriminator network D gets input data from real data and the generator's output, then tries to distinguish the real data from the generated data. The two networks are trained by gradient descent algorithms in alternating steps.

The classical GAN minimizes the Jensen-Shannon (JS) divergence between the real and approximate distributions. However, [21] shows that JS metric is not continuous and does not provide a usable gradient all the time. To overcome the defect, [21] proposed WGAN which replaces the JS metric by 1-Wasserstein distance that provides sufficient gradients almost everywhere. However, it is vital for WGAN to find an 1-Lipschitz function that is approximated by the discriminator network. [22] proposed WGAN with gradient penalty (WGAN-GP) algorithm and adopted gradient penalty to enforce the 1-Lipschitz constraint. Its optimization objective is formulated as follow:

$$\min_{G} \max_{D\in\mathcal{D}} \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{z}}(\mathbf{z})}[D(G(\mathbf{z}))] + p(\lambda) \tag{6}$$

where $\mathcal{D}$ denotes the set of 1-Lipschitz functions, $\mathbf{x}$ denotes the samples from real data, $\boldsymbol{z}$ denotes the samples from a random distribution, and $p(\lambda) = \lambda\left(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1\right)^2$, $\hat{\mathbf{x}} = \varepsilon\mathbf{x} + (1-\varepsilon)G(\mathbf{z})$, $\varepsilon \sim U(0,1)$. Gradient penalty increases the computational complexity but it does make WGAN-GP perform much better than previous GANs.

B. System Model

We consider a radio access network (RAN) scenario with several base stations (BS) , where there exists a list of existing slices $1,\cdots,M$ sharing the aggregated bandwidth W and having fluctuating traffic demands $\mathbf{d} = (d_1,\cdots,d_M)$. The objective of our work is to find an optimal bandwidth-sharing solution $\mathbf{w} = (w_1,\cdots,w_M)$ which maximizes the expectation of the whole system utility, $\mathbb{E}[J(\mathbf{w},\mathbf{d})]$, where $J(\mathbf{w},\mathbf{d})$ is

defined as the weighted sum of SE and QoE satisfaction ratio. Mathematically,

$$J(\mathbf{w}, \mathbf{d}) = \alpha \cdot \mathrm{SE}(\mathbf{w}, \mathbf{d}) + \beta \cdot \mathrm{QoE}(\mathbf{w}, \mathbf{d}) \tag{7}$$

where α and β are the coefficients that adjust the importance of SE and QoE. Therefore, this radio resources slicing problem is given below:

$$\begin{aligned} & \arg\max_{\mathbf{w}} \mathbb{E}[J(\mathbf{w}, \mathbf{d})] \\ = & \arg\max_{\mathbf{w}} \mathbb{E}[\alpha \cdot \mathrm{SE}(\mathbf{w}, \mathbf{d}) + \beta \cdot \mathrm{QoE}(\mathbf{w}, \mathbf{d})] \\ s.t. \quad & \mathbf{w} = (w_1, \cdots, w_M) \\ & w_1 + \cdots + w_M = W \\ & \mathbf{d} = (d_1, \cdots, d_M) \\ & d_i \sim \text{ Certain Traffic Model, } \forall i \in [1, \cdots, M] \end{aligned} \tag{8}$$

Notably, the provisioning capabilities are tangled with the transmission capacity. For example, the TCP sending window size is influenced by the estimated channel throughput. Therefore, the traffic demand varies without knowing a prior transition probability, making Eq. (8) difficult to yield a direct solution. However, RL promises to be applicable to tackle with this problem. Therefore, we refer to RL to find the optimal policy for network slicing and build the system model as shown in Fig. 1.

III. GAN-based Deep Distributional Reinforcement Learning

Our previous work [4] has discussed how to apply RL to find a network slicing policy. However, during the practical application, the way to estimate the expectation of state-action values in RL induces some negative impact, especially for scenarios to learn from a non-stationary policy. In order to solve this problem, we leverage distributional RL, which learns an approximate distribution of the state-action values, so as to obtain more stable and superior learning results. Consistent with [4], we map the RAN scenario to the environment in RL by taking the number of arrived packets in each slice within a specific time window as the state, and the allocated bandwidth to each slice as the action. However, considering the stability of GAN-DDQN's training process might be affected by the widely-spanning utility values, we clip the system utility to several fixed values, which is taken as the reward in RL. The clipping strategy we apply is shown in Eq. (9), where c_1 and c_2 ($c_1 > c_2$) are manually carefully set thresholds.

$$r = \begin{cases} 1, & J(\mathbf{w}, \mathbf{d}) \geq c_1 \\ 0, & c_2 < J(\mathbf{w}, \mathbf{d}) < c_1 \\ -1, & J(\mathbf{w}, \mathbf{d}) \leq c_2 \end{cases} \tag{9}$$

Then, inspired by [13], we leverage GAN to approximate the distribution. Specifically, the generator network G is responsible for producing the estimated samples of $Z(s, a)$, and the discriminator network D aims to distinguish the samples of $\mathcal{T}Z(s, a)$ from the samples of $Z(s, a)$.

Overally, at each episode, the agent feeds the current state s_t and the samples from a uniform distribution (e.g $U(0, 1)$), namely τ, to G network, where τ represents the respective quantile values of the target distribution [12]. As a result, the agent gets the samples $G^{(a)}(s_t, \tau)$ for every action a from the current approximate $Z(s_t, a)$. Then the agent calculates the expectation of $Z(s_t, a)$, i.e. $Q(s_t, a)$, and leveraging ϵ-greedy strategy to decide whether random action or the action $a_t \leftarrow \arg\max_a Q(s_t, a)$ to perform, receiving a reward r_t and moving to the next state s_{t+1}. The tuple (s_t, a_t, r_t, s_{t+1}) is stored into a replay buffer $\mathcal{B}$ as done in [18]. When the buffer $\mathcal{B}$ is full, the agent updates D and G networks using

Algorithm 1 GAN-DDQN

Initialize the generator network G and the discriminator network D with random weights $\boldsymbol{\theta}_G$ and $\boldsymbol{\theta}_D$ respectively, a target generator network $\hat{G}$ with weight $\boldsymbol{\theta}_{\hat{G}} \leftarrow \boldsymbol{\theta}_G$, the replay buffer $\mathcal{B} \leftarrow \varnothing$, the number of samples N, gradient penalty coefficient λ, batch size m, discount factor γ.
Initialize an episode index $t = 0$.
repeat
  At episode t, the agent observes the state s_t.
  The agent samples $\tau \sim U(0, 1)$.
  The agent calculates action values $Q(s_t, a) = \frac{1}{N} \sum G^{(a)}(s_t, \tau)$.
  The agent performs the action $a_t \leftarrow \arg\max_a Q(s_t, a)$ following ϵ-greedy strategy, receiving the system utility J and observes a new state s_{t+1}.
  The agent performs the reward-clipping with respect to J and gets the reward r_t.
  The agent stores transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{B}$.
  If $\mathcal{B}$ is full, the agent updates the weights of G network and D network every K episodes.
  # Train GAN
  repeat
    The agent samples a minibatch $\{s, a, r, s'\}_{i=1}^m$ from $\mathcal{B}$ without replacement.
    The agent samples a minibatch $\{\tau\}_{i=1}^m \sim U(0, 1)$.
    The agent gets the next action $a_i^* = \arg\max_a \frac{1}{N} \sum \hat{G}^{(a)}(s_i', \tau_i)$, and sets $y_i = r_i + \gamma \hat{G}^{(a_i^*)}(s_i', \tau_i)$.
    The agent samples a minibatch $\{\varepsilon\}_{i=1}^m \sim U(0, 1)$, and sets $\hat{x}_i = \varepsilon_i y_i + (1 - \varepsilon_i) G^{(a_i)}(s_i, \tau_i)$
    The agent updates the weights $\boldsymbol{\theta}_D$ by leveraging gradient descent algorithm to $\frac{1}{m} \sum_{i=1}^m \mathcal{L}_i$, where $\mathcal{L}_i = D(G^{(a_i)}(s_i, \tau_i)) - D(y_i) + \lambda \left(\|\nabla_{\hat{x}_i} D(\hat{x}_i)\|_2 - 1 \right)^2$.
    The agent updates the weights $\boldsymbol{\theta}_G$ by leveraging gradient descent algorithm to $-\frac{1}{m} \sum_{i=1}^m D(G^{(a_i)}(s_i, \tau_i))$
  until All the transitions in $\mathcal{B}$ are used for training.
  The agent clones the G network to the target network $\hat{G}$ every C episodes by resetting $\boldsymbol{\theta}_{\hat{G}} = \boldsymbol{\theta}_G$
  The episode index is updated by $t \leftarrow t + 1$
until A predefined stopping condition (e.g., the $\frac{1}{m} \sum_{i=1}^m \mathcal{L}_i$, the episode length, etc.) is satisfied.

all the transition tuples in $\mathcal{B}$ every K episodes.

As for the updating and training procedues, the agent first executes the Bellman optimality operator $\mathcal{T}$ and obtains a set of samples that describe the real distribution, i.e., $y_i = r_i + \gamma \hat{G}^{(a_i^*)}(s_i', \tau_i)$ where $a_i^* = \arg\max_a \frac{1}{N} \sum \hat{G}^{(a)}(s_i', \tau_i)$ and $\hat{G}$ is the target generator network [18]. Then we use the following loss functions to train G and D networks, respectively:

$$\mathcal{L}_D = \mathop{\mathbb{E}}_{\substack{\tau \sim U(0,1) \\ (s,a) \sim \mathcal{B}}} [D(G^{(a)}(s,\tau))] - \mathop{\mathbb{E}}_{(s,a,r,s') \sim \mathcal{B}} [D(y)] + p(\lambda) \tag{10}$$

$$\mathcal{L}_G = - \mathop{\mathbb{E}}_{\substack{\tau \sim U(0,1) \\ (s,a) \sim \mathcal{B}}} [D(G^{(a)}(s,\tau))] \tag{11}$$

where $p(\lambda)$ is as mentioned in Eq. (6). The training goal, on the one hand, is to make the D network more accurate in distinguishing the real data from the "fake" data produced by the G network. On the other hand, training the G network is to improve the generator's ability to produce "fake" data that fools the D network as much as possible. Note that in order to further stabilize the training process, we update the target network $\hat{G}$ every C episodes.

Step by step, we incorporate the aforementioned methods and design the GAN-DDQN as in Algorithm 1.

IV. Simulation Results and Numerical Analysis

In this part, we compare the performance of GAN-DDQN, QR-DQN [10], and DQN [8] in a RAN scenario where there are three types of services (i.e., VoLTE, video, and URLLC) and three corresponding slices in one single BS as in [4]. The specific parameters setting is the same as those in [4], i.e., the total bandwidth to these three slices is 10 MHz, and the minimal bandwidth allocation resolution is 1 MHz, which means the number of valid actions is 36. We primarily consider the downlink case and adopt system SE and QoE satisfaction ratio as the evaluation metrics. In particular, the system SE is computed as the number of bits transmitted per second per unit bandwidth, in which the rate from the BS to mobile users is derived based on Shannon capacity formula. Therefore, if part of the available bandwidth has been allocated to one slice but the slice has no service activities at that slot, such part of bandwidth would have been wasted, thus degrading the system SE. QoE satisfaction ratio is obtained by dividing the number of completely transmitted packets satisfying rate and latency requirement by the total number of arrived packets.

Fig. 2 depicts the variations of the system utility with respect to the episode index. We fix β to 1, and consider the situations of $\alpha = 0.1$ and $\alpha = 0.01$. The setting of the clipping parameters c_1 and c_2 is heuristic [1]. For the two cases, we respectively set $c_1 = 33, c_2 = 28$ and $c_1 = 4, c_2 = 2$ to clip the system utility according to Eq. (9). It can be observed from

[1]We first directly regard the system utility as the reward in order to find the range of the system utility, and then try different combinations of the two parameters (i.e., c_1 and c_2) to find the values that guarantee both performance and stability.

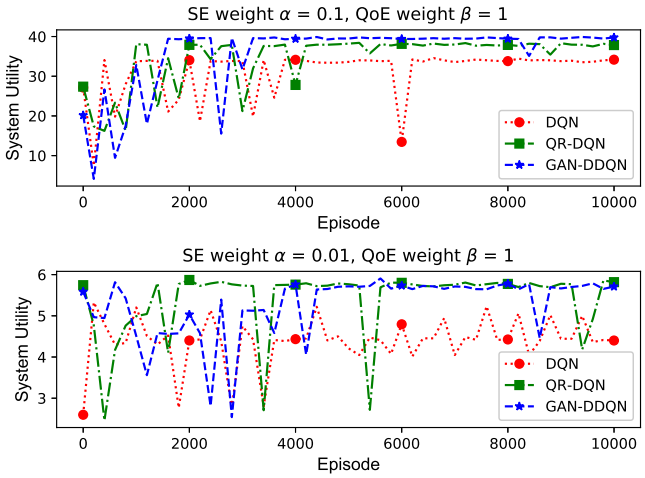

Fig. 2. An illustration of the performance comparison between different models.

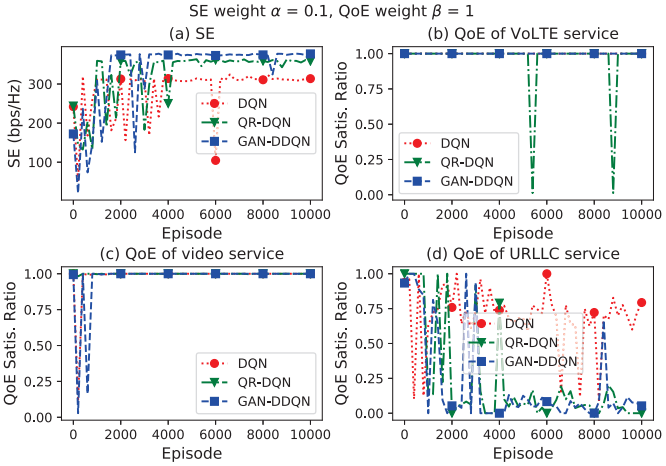

Fig. 3. An illustration of the SE and QoE satisfaction ratio in the case of $\alpha = 0.1, \beta = 1$.

Fig. 2 that our proposed GAN-DDQN performs much better than DQN regardless of the α value, and a little bit better than QR-DQN in the case of $\alpha = 0.1$.

Fig. 3 and Fig. 4 present the variations of the SE and QoE satisfaction ratio with respect to the episode index in the case of $\alpha = 0.1$ and $\alpha = 0.01$, respectively. It can be observed that with respect to the SE metric, GAN-DDQN works best, followed by QR-DQN and finally DQN. However, as for the QoE satisfaction ratio, neither GAN-DDQN nor QR-DQN provides satisfactory result for URLLC service as shown in Fig. 3(d) and Fig. 4(d), while DQN produces the policy that achieves better QoE satisfaction ratio. The reason why the three models perform either poorly or unstably under the QoE satisfaction ratio for the URLLC service might be that the agent prefers to increase the SE and sacrifice the QoE of the URLLC service considering the trivial number of requests for the URLLC service.

Fig. 5 shows the variations of the system utility during the

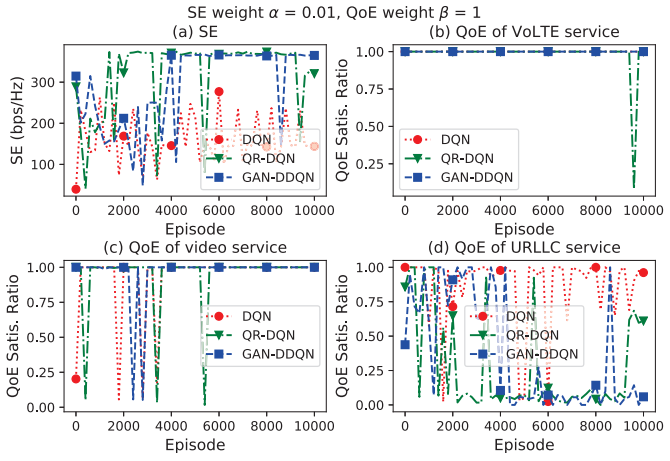

Fig. 4. An illustration of the SE and QoE satisfaction ratio in the case of $\alpha = 0.01, \beta = 1$.

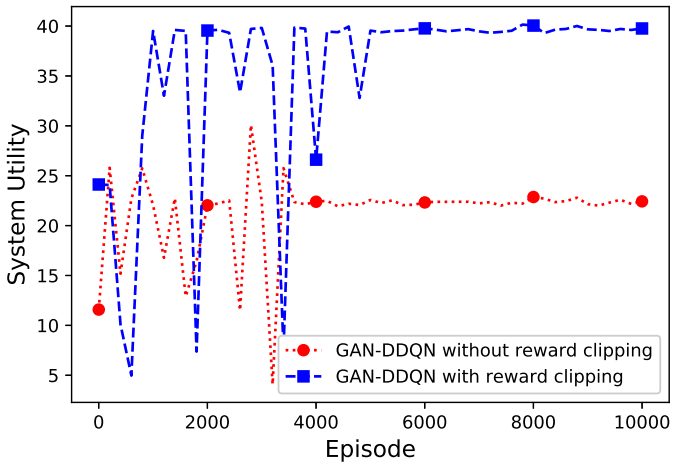

Fig. 5. Comparison of GAN-DDQN with and without reward clipping.

iteration of GAN-DDQN with and without the reward clipping. For the case of no clipping, GAN-DDQN directly takes the system utility as the reward, converging to a bad value as shown in Fig. 5. From Algorithm 1, we can see that the discriminator network takes the reward as a fraction of input. However, the large rewards make the discriminator difficult to converge, resulting in the GAN-DDQN's training process becoming very unstable and ultimately failing to find the optimal policy. The simulation results verify that the reward clipping does contribute to getting superior performance.

V. Conclusion

In this paper, we have investigated the combination of deep distributional RL and GAN, and proposed GAN-DDQN to learn the solution for demand-aware resource management in network slicing. In particular, we have applied GAN to approximate the distribution of state-action values, so as to grasp further details therein than the conventional DQN. We have also designed a new update procedure to combine the advantages of both the training algorithm in WGAN-GP and the settings of DQN. Furthermore, we have adopted the reward-clipping procedure to enhance the training stability of GAN-DDQN. Numerical results have demonstrated the effectiveness of GAN-DDQN with superior performance over DQN and QR-DQN. In the future, we will try to further improve the GAN-DDQN under various scenarios with multiple-metric constraints as well as non-stationary traffic demands.

References

[1] K. Katsalis, N. Nikaein, *et al.* Network slices toward 5G communications: Slicing the LTE network. *IEEE Communications Magazine*, 55(8):146–154, 2017.

[2] R. Li, Z. Zhao, *et al.* Intelligent 5G: When cellular networks meet artificial intelligence. *IEEE Wireless Communications*, 24(5):175–183, 2017.

[3] X. Foukas, G. Patounas, *et al.* Network slicing in 5G: Survey and challenges. *IEEE Communications Magazine*, 55(5):94–100, 2017.

[4] R. Li, Z. Zhao, *et al.* Deep reinforcement learning for resource management in network slicing. *IEEE Access*, 6:74429–74441, 2018.

[5] S. Vassilaras, L. Gkatzikis, *et al.* The algorithmic aspects of network slicing. *IEEE Communications Magazine*, 55(8):112–119, Aug 2017.

[6] G. A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

[7] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[8] V. Mnih, K. Kavukcuoglu, *et al.* Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[9] M. G. Bellemare, W. Dabney, *et al.* A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR. org, 2017.

[10] W. Dabney, M. Rowland, *et al.* Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[11] R. Koenker and K. F. Hallock. Quantile regression. *Journal of economic perspectives*, 15(4):143–156, 2001.

[12] W. Dabney, G. Ostrovski, *et al.* Implicit quantile networks for distributional reinforcement learning. *arXiv preprint arXiv:1806.06923*, 2018.

[13] T. Doan, B. Mazoure, *et al.* GAN Q-learning. *arXiv preprint arXiv:1805.04874*, 2018.

[14] X. Li, J. Fang, *et al.* Intelligent power control for spectrum sharing in cognitive radios: A deep reinforcement learning approach. *IEEE Access*, 6:25463–25473, 2018.

[15] Z. Xu, Y. Wang, *et al.* A deep reinforcement learning based framework for power-efficient resource allocation in cloud rans. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2017.

[16] N. Liu, Z. Li, *et al.* A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 372–382, June 2017.

[17] Y. He, F. R. Yu, *et al.* Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach. *IEEE Communications Magazine*, 55(12):31–37, Dec 2017.

[18] V. Mnih, K. Kavukcuoglu, *et al.* Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[19] G. Barth-Maron, M. W. Hoffman, *et al.* Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

[20] I. Goodfellow, J. Pouget-Abadie, *et al.* Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[21] M. Arjovsky, S. Chintala, *et al.* Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

[22] I. Gulrajani, F. Ahmed, *et al.* Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.