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

Yuxiu Hua, Rongpeng Li, *Member, IEEE*, Zhifeng Zhao, Xianfu Chen, and Honggang Zhang

***Abstract*— Network slicing is a key technology in 5G communications system. Its purpose is to dynamically and efficiently allocate resources for diversified services with distinct requirements over a common underlying physical infrastructure. Therein, demand-aware resource allocation is of significant importance to network slicing. In this paper, we consider a scenario that contains several slices in a radio access network with base stations that share the same physical resources (e.g., bandwidth or slots). We leverage deep reinforcement learning (DRL) to solve this problem by considering the varying service demands as the environment *state* and the allocated resources as the environment *action*. In order to reduce the effects of the annoying randomness and noise embedded in the received service level agreement (SLA) satisfaction ratio (SSR) and spectrum efficiency (SE), we primarily propose generative adversarial network-powered deep distributional Q network (GAN-DDQN) to learn the action-value distribution driven by minimizing the discrepancy between the estimated action-value distribution and the target action-value distribution. We put forward a reward-clipping mechanism to stabilize GAN-DDQN training against the effects of widely-spanning utility values. Moreover, we further develop Dueling GAN-DDQN, which uses a specially designed dueling generator, to learn the action-value distribution by estimating the state-value distribution and the action advantage function. Finally, we verify the performance of the proposed GAN-DDQN and Dueling GAN-DDQN algorithms through extensive simulations.**

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

Manuscript received June 20, 2019; revised October 15, 2019; accepted November 6, 2019. Date of publication December 12, 2019; date of current version February 19, 2020. This work was supported in part by the National Key R&D Program of China under Grant 2017YFB1301003, in part by the National Natural Science Foundation of China under Grant 61701439 and Grant 61731002, in part by the Zhejiang Key Research and Development Plan under Grant 2019C01002 and Grant 2019C03131, in part by the Zhejiang Lab under Grant 2019LC0AB01, in part by the Zhejiang Provincial Natural Science Foundation of China under Grant LY20F010016, and in part by the Fundamental Research Funds for the Central Universities under Grant 2019QNA5010. This article was presented in part at the IEEE Globecom 2019 [1]. *(Corresponding author: Rongpeng Li.)*

Y. Hua, R. Li, and H. Zhang are with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China (e-mail: 21631087@zju.edu.cn; lirongpeng@zju.edu.cn; honggangzhang@zju.edu.cn).

Z. Zhao is with the Zhejiang Lab, Hangzhou 311121, China, and also with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China (e-mail: zhaozf@zhejianglab.com).

X. Chen is with the VTT Technical Research Centre of Finland, 90570 Oulu, Finland (e-mail: xianfu.chen@vtt.fi).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2019.2959185

I. Introduction

THE emerging fifth-generation (5G) mobile systems, armed with novel network architecture and emerging technologies, are expected to offer support for a plethora of network services with diverse performance requirements [2], [3]. 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 [4]. According to ITU-R recommendations, it is a consensus that the key technologies of 5G wireless systems will spawn three generic application scenarios: enhanced mobile broadband (eMBB), massive machine-type communications (mMTC), and ultra-reliable and low-latency communications (URLLC) [5]. Specifically, (a) eMBB supports data-driven use cases requiring high data rates across a wide coverage area; (b) mMTC supports a very large number of devices in a broad area, which may only send data sporadically, such as Internet of Things (IoT) use cases; (c) URLLC supports strict requirements on latency and reliability for mission-critical communications, such as remote surgery, autonomous vehicles or Tactile Internet. Serving a diverse set of use cases over the same network will increase complexity, which must be managed to ensure acceptable service levels. As the services related to different use cases can have very different characteristics, they will impose varying requirements on the network resources. For example, considering mMTC devices for utility metering and parking sensors, they hardly move and do not require mobility management, i.e., the need to track location. On the other hand, sensors related to freight management would commonly move even across countries' borders and would require mobility management, including roaming agreements.

However, legacy mobile networks are mostly designed to provide services for mobile broadband consumers and merely consist of a few adjustable parameters like priority and quality of service (QoS) for dedicated services. Thus it is difficult for mobile operators to extend their networks into these emerging vertical services because of the different service requirements for network design and development. The concept of network slicing has recently been proposed to address this challenging problem; the physical and computational resources of the network can be sliced to meet the diverse needs of a range

of 5G users [2], [6], [7]. In this way, heterogeneous requirements can be served cost-effectively by the same physical infrastructure, since different network slice (NS) instances can be orchestrated and configured according to the specific requirements of the slice tenants.

As a non-nascent concept, network slicing can be traced back to the Infrastructure as a Service (IaaS) cloud computing model [8], whereby different tenants share computing, networking and storage resources to create different isolated fully-functional virtual networks on a common infrastructure. In the context of 5G and beyond, network functions virtualization (NFV) and software defined networking (SDN) technologies serve as a basis for the core network slicing by allowing both physical and virtual resources to be used to provide certain services, thus enabling 5G networks to deliver different kinds of services to various customers [9], [10]. On the other hand, the Next Generation Mobile Networks (NGMN) alliance puts forward an evolved end-to-end network slicing idea while the Third-Generation Partnership Project (3GPP) also suggests that radio access network (RAN) should not be excluded to "design specific functionality to support multiple slices or even partition of resources for different network slices" [11], [12].

However, in order to provide better-performing and cost-efficient services, RAN slicing involves more challenging technical issues for the realtime resource management on existing slices, since (a) for RANs, spectrum is a scarce resource and it is essential to guarantee the spectrum efficiency (SE) [9]; (b) the service level agreements (SLAs) with slice tenants usually impose stringent requirements; and (c) the actual demand of each slice heavily depends on the request patterns of mobile users (MUs) [13]. Therefore, the classical dedicated resource allocation fails to address these problems simultaneously [11]. Instead, it is necessary to intelligently allocate the radio resources (e.g., bandwidth or slots) to slices according to the dynamics of service requests from mobile users coherently [14] with the goal of meeting SLA requirements in each slice, but at the cost of acceptable SE. In this regard, there have been extensive efforts [13], [15]–[20], [43]. [15] proposed an online genetic slicing strategy optimizer for inter-slice resource management. However, [15] did not consider the explicit relationship between the required resource and SLA on a slice, as one slice might require more resources given its more stringent SLA. Reference [16] considered the problem of different types of resources (bandwidth, caching, backhaul capacities) being allocated to NS tenants based on user demands. The authors proposed mathematical solutions, but the optimization problem would become intractable when the simulation parameters are scaled up (e.g., increasing the number of NSs or the shareable resources). Reference [17] mathematically analyzed the joint optimization problem of access control and bandwidth allocation in the multi-base station (BS) multi-NS scenario. However, the solutions therein are based on the assumption that different users have the same fixed demand rate, which is a condition unlikely to be found in practice. From the perspective of bandwidth usage-based pricing (UBP), [20] used game theory to analyze the relationship between Internet service providers (ISPs) and users, thereby improving the profit of ISPs and solving the peak-time congestion problem. However, the timeslot for bandwidth allocation in [20] was 1 hour, which is unrealistic in a situation where the number of demands changes drastically in a short period.

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 optimal actions in an environment from observing state transitions and obtaining feedback (rewards/costs). In RL, the action value, $Q(s, a)$, describes the expected return, or the discounted sum of rewards, when performing action a in state s. Usually, the Q value can be estimated by classic value-based methods such as SARSA [21] and Q-learning [22] based on the Bellman equation. Reference [23] used deep neural networks to approximate the Q function, namely deep Q network (DQN), which demonstrated human-like performance on simple computer games and inspired a research wave of deep reinforcement learning (DRL). Besides modeling $Q(s, a)$, [24] showed that we could learn the distribution of $Q(s, a)$ by a distributional analogue of Bellman equation; this approach improved the estimation of action values in an inherently randomness environment. Specifically, [24] proposed C51 algorithm to minimize the Kullback–Leibler (KL) divergence between the approximated Q distribution and the target Q distribution calculated by the distributional Bellman optimality operator. Inspired by the theory of quantile regression [25], [26] proposed the quantile regression DQN (QR-DQN) and thus successfully performed distributional RL over the Wasserstein metric, leading to the state-of-the-art performance. Reference [27] extended QR-DQN from learning a discrete set of quantiles to learning the full quantile function and put forward the implicit Q network (IQN). Given the success of replacing $Q(s, a)$ by its distribution in [24], [26], [27] as well as the reputation of generative adversarial network (GAN) for approximating distributions [28], it naturally raises a question of whether GAN is viable for approximating the action-value distribution and thus improving distributional RL.

In the field of communications and networking, DRL has triggered tremendous research attention to solving resource allocation issues in some specific fields like power control [29], green communications [30], cloud RANs [31], mobile edge computing and caching [32]. Given the challenging technical issues in resource management on existing NSs, the previous work in [13] leveraged DQN to find the optimal resource allocation policy and investigated its performance. However, the method proposed in [13] did not consider the effects of random noise on the calculation of SE and SLA satisfaction ratio (SSR). To mitigate the potential risk of incorrectly estimating the action value due to the randomness in SE and SSR, we intend to introduce the distributional RL to estimate the action-value distribution, thus avoiding the action-value overestimation or underestimation issue that plagues many traditional value-based RL algorithms (e.g., DQN). Meanwhile, the cutting-edge performance of Wasserstein generative adversarial network with gradient penalty (WGAN-GP) in the distribution approximation suggests to us that we might use it to learn the

action-value distribution. To this end, we propose a new approach, the GAN-powered deep distributional Q network (GAN-DDQN), based on distributional RL and WGAN-GP, to realize dynamic and efficient resource allocation per slice. The main contributions of this paper are as follows:

- To find the optimal resource allocation policy under the uncertainty of slice service demands, we design the GAN-DDQN algorithm, where the generator network outputs a fixed number of particles that try to match the action-value distribution for each action. Such a design in GAN-DDQN can mitigate the effects of learning from a nonstationary environment and is significantly different from the concurrent yet independent works [33], [34].[1]
- We demonstrate that the widely-spanning system utility values could destabilize GAN-DDQN's training process, and correspondingly design a reward-clipping mechanism to reduce this negative impact. Specifically, we clip the system utility values to some constant values according to a straightforward rule with several heuristics-guided adjustable thresholds, and then use these constants as the final rewards in RL.
- GAN-DDQN suffers from the challenge that only a small part of the generator output is included in the calculation of the loss function during the training. To compensate for this, we further propose Dueling GAN-DDQN, which is a special solution derived from Dueling DQN [35] and the discrete normalized advantage functions (DNAF) algorithm [36]. Dueling GAN-DDQN separates the state-value distribution from the action-value distribution and combines the action advantage function to obtain the action values. In addition, we elaborate on twofold loss functions that further take advantage of the temporal difference (TD) error information to achieve performance gains. The introduction of dueling networks to GAN-DDQN makes the work described in this paper significantly different from our previous work presented in IEEE Globecom 2019 [1].
- Finally, we perform extensive simulations to demonstrate the superior efficiency of the proposed solutions over the classical methods, such as DQN, and provide insightful numerical results for the implementation details.

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

II. Preliminaries and System Model

A. Preliminaries

Table I lists the important notations used in this paper. 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 is the reward, $P(\cdot|s,a)$ is the transition probability, and $\gamma \in (0,1]$ is a discount factor. A policy $\pi(\cdot|s)$ maps a state to a distribution over actions. The state-value function of a state s under a policy $\pi(\cdot|s)$, denoted $V^\pi(s)$, is the expected return when starting in s and following π thereafter. Similarly, we define the value of taking action a in state s under the policy π, denoted $Q^\pi(s,a)$, as the expected return starting from s, taking the action a, and thereafter following policy π. Mathematically, the state-value function is

$$V^\pi(s) = \mathbb{E}_{\pi,P}\left[\sum_{t=0}^{\infty}\gamma^t R_t | S_0 = s\right], \tag{1}$$

and the action-value function is

$$Q^\pi(s,a) = \mathbb{E}_{\pi,P}\left[\sum_{t=0}^{\infty}\gamma^t R_t | S_0 = s, A_0 = a\right], \tag{2}$$

where $\mathbb{E}$ denotes the expectation. The relationship between the value of a state and the values of its successor states is expressed by the Bellman equation for V^π

$$V^\pi(s) = \mathbb{E}_{\pi,P}\left[R + \gamma V^\pi(s')\right]. \tag{3}$$

Similarly, the Bellman equation for Q^π is

$$Q^\pi(s,a) = \mathbb{E}_{\pi,P}\left[R + \gamma Q^\pi(s',a')\right], \tag{4}$$

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

The goal of RL is to find the optimal policy which yields the maximum $Q(s,a)$ for all s and a. Let $\pi^* = \arg\max_\pi Q^\pi(s,a)$ be the optimal policy and let $Q^*(s,a)$ be the corresponding

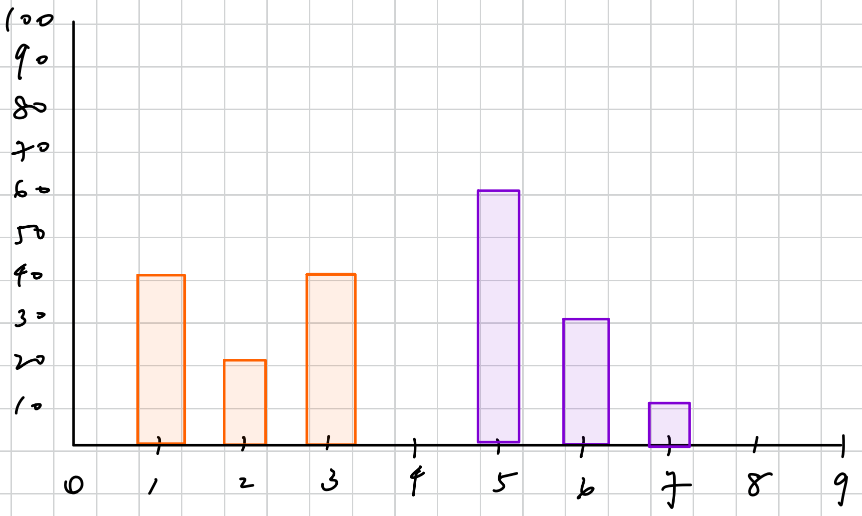
TABLE I
Notations Used in This Paper

Notation	Definition
$\mathcal{S}$	State space
$\mathcal{A}$	Action space
P	Transition probability
V	State-value function
Q	Action-value function
Z_v	Random variable to statistically model the state values
Z_q	Random variable to statistically model the action values
$\mathcal{T}^*$	Bellman optimality operator
s, s'	States
a	An action
r	A reward
S_t	State at time t
A_t	Action at time t
R_t	Reward at time t
γ	Discount factor
π	Policy
J	System utility
α	Weight of the SE
β	Weight of the SSR
τ	Quantile samples
λ	Gradient penalty coefficient
n_{critic}	The number of discriminator updates per training iteration

[1] [33] used a generator network that directly outputs action values and did not show any significant improvement of GAN Q-learning over conventional DRL methods. [34] used the policy iteration method [37] to loop through a two-step procedure for the value estimation and policy improvement, where GAN was only used to estimate the action-value distribution. Besides, [34] exploited a totally different framework without the target generator, which is a key component for GAN-DDQN.

1-Wasserstein Distance

Q



X: $P(X=1)=0.4$, $P(X=2)=0.2$, $P(X=3)=0.4$

Y: $P(Y=5)=0.6$, $P(Y=6)=0.3$, $P(Y=7)=0.1$

p-Wasserstein Distance $W_p(X,Y)=\left(\int_0^1 |F_X^{-1}(w)-F_Y^{-1}(w)|^p\,dw\right)^{1/p}$

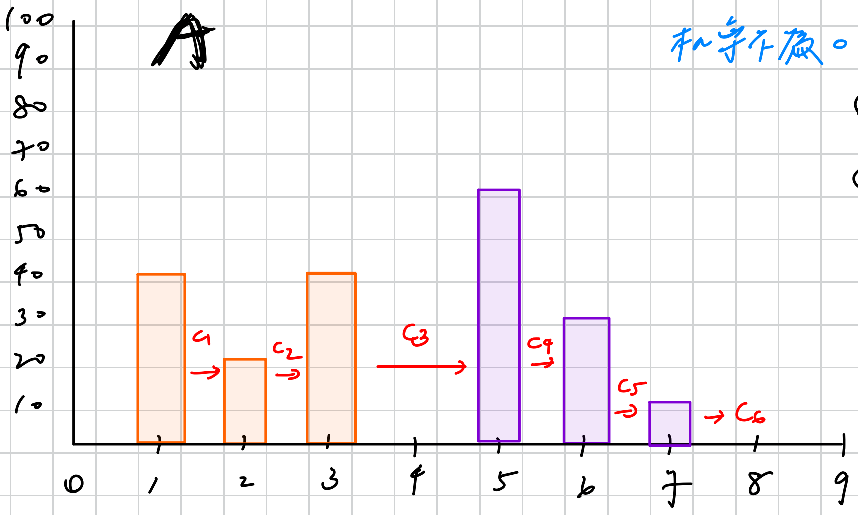
這裡 Demo 的 → 1-Wasserstein Distance by PMF (Discrete)

$$W_1(X,Y)=\sum_{i=1}^{n} |F_P(x_i)-F_Q(x_i)|\cdot \Delta x$$

質量　　搬運距離 $= x_{i+1}-x_i$

☆ 這裡用 CDF 而非 Inverse CDF，離散時它們理論上等價。

A



機率不為0的點 ↘

① Support $= \{1,2,3,5,6,7\}$ （x_1 … x_6）

② CDF of each x of X & Y:

$F_X(x)=P(X\le x)$

$F_Y(x)=P(Y\le x)$

x	$F_X(x)$	$F_Y(x)$
1	0.4	0
2	0.6 (0.2+0.4)	0
3	1 (0.2+0.4+0.4)	0
5	1	0.6
6	1	0.9
7	1	1

③

	x	$\|F_X(x)-F_Y(x)\|$	Δx
c_1	1	$0.4-0=0.4$	1
		X 能往右推的土	跟下一點的距離
c_2	2	$0.6-0=0.6$	1
		前面推來的0.4跟現有的0.2	
c_3	3	$1-0=1$	2
		在1,2,3中X剩1能往右推	下一個點的距離
c_4	5	$1-0.6=0.4$	1
		在5的位置用了0.6去堆Y了，剩0.4能往右	
c_5	6	$1-0.9=0.1$	1
c_6	7	$1-1=0$	1

Sum $=0.4+0.6+2+0.4+0.1=3.5$ #

Contraction Mapping

$\|f(x_1) - f(x_2)\| \leq \Gamma \cdot \|x_1 - x_2\|$

Def T^π 為 D.B.E. 之 Operator：

$T^\pi Z_q(s,a) \overset{D}{=} R + \Gamma \cdot Z_q(s',a')$

$Z_q^\pi(s,a) \overset{D}{=} R + \gamma Z_q^\pi(s',a')$,

(A)

Bellemare *et al.* proved that (the distributional Bellman equation is a contraction in p-Wasserstein distance,) but the (distributional Bellman optimality operator is not necessarily a contraction, which provides a guideline for metric selection)

$$\mathcal{T}^* Z_q(s,a) \overset{D}{=} R + \gamma Z_q\left(s', \arg\max_{a' \in \mathcal{A}} \mathbb{E}[Z_q(s',a')]\right)$$

① 即 $W_p(T^\pi Z_1, T^\pi Z_2) \leq \Gamma \cdot W_p(Z_1, Z_2)$，

代表 $T^\pi Z_1$ & $T^\pi Z_2$ 的時候 Wasserstein Distance 會比 Z_1 & Z_2 的時候 W.D. 還又 Γ 倍小。

② $W_p(T^* Z_1, T^* Z_2) \leq \Gamma \cdot W_p(Z_1, Z_2)$

若在 KL-Divergence 的condition下，D.B.E. 就不會收斂又。

故在設計 Loss function 時會用 p-Wasserstein Distance。

即 Model: $\hat{Z}_\theta(s,a)$

Target: $Z_{target} \overset{D}{=} r + \Gamma \cdot Z(s',a')$ ← 理論上的真實分布

Loss: $L(\theta) = W_p(\hat{Z}_\theta(s,a), T^\pi Z(s,a))$

但前面不是說要最小化 $\sup_{s,a} \text{dist}(T^* Z_q(s,a), Z_q(s,a))$，

但後面又說它無法收斂又。 ★後面又講到 DBOO (T^*)

是這樣的，雖然沒有理論上的收斂又保証，

但我們仍能設計一个合理的 Loss function (by p-Was).

畢竟未來就很難滿足理論 in R.L.

在 QR-DQN & p-Wass. 下會收斂又

Ⓑ

A. 前面那段說：

> The **distributional Bellman optimality operator** is **not necessarily a contraction**, which provides a guideline for metric selection.

✅ 這是 **Bellemare et al. 2017** 的原始分析（C51 論文）：

- 在某些距離空間（如 KL 或 TV 距離），DBOO 不是 contraction
- 甚至連在 **1-Wasserstein distance** 上也不是，除非你有額外假設

→ 放C51使用KL-Divergence

→ 文章收斂性 & 穩定性的問題

B. 現在你這段說：

> DBOO is a **strict contraction** in **p-Wasserstein distance**

✅ 這來自的是 **QR-DQN（Dabney et al., 2018）** 的分析，它針對的是：

> **分布是透過 quantile 表示（support 是固定位置）** 的特定 case，
> 在這種架構下，DBOO 的確可以證明為 strict contraction in W_p。

→ 放QR-DQN使用p-Wasserstein

🔍 **核心關鍵點：**

條件	是否是 contraction
一般隨機變數分布（無固定 support）	❌ DBOO **不是** contraction in W_1
分布以 quantile 表示（QR-DQN 設計）	✅ DBOO 是 strict contraction in W_p

➡️ 也就是說：**不是所有 case 都有 contraction，但某些結構設計下是可以證明 contraction 的。**

🧠 **更具體地說：**

- 若你令 return distribution 表示為：

$$Z(s,a) = \sum_{i=1}^{N} \delta_{z_i} \cdot \frac{1}{N}$$

  並固定 support 的位置（像 QR-DQN），就可以證明 DBOO 是 contraction in W_1

- 但若分布是任意形式，或是 support 不是固定的，則 contraction 保證會失效

結論：

原文敘說，DBOO不保証 Contraction.

但在 QR-DQN 這種結構中 Contraction.

2. 固定Quantile levels, 2. 有限支撐集

Strict Contraction $\|f(x_1) - f(x_2)\| < \|x_1 - x_2\|$

→ in p-Wasserstein $W_p(f(x_1), f(x_2)) < W_p(x_1, x_2)$

🧠 那直接最小化 Wasserstein 可以嗎？

❌ 在原始形式下不太可行，原因如下：

1. Wasserstein 距離需要排序或 inverse CDF，這些是不可微操作（梯度無法傳遞）
2. 你需要知道兩個分布的 完整形式或 samples，而不是單一點估計
3. 無法直接在神經網路參數上定義 gradient

🔁 世界上主要有兩種方法在逼近 Wasserstein Distance：

方法類型	代表技術	說明
✅ 間接法（可微）	Quantile Regression（QR-DQN）	針對 一維機率分布，利用 quantile loss 近似 W_1，可以直接反向傳播
✅ 對偶法（GAN 用）	Wasserstein GAN (WGAN)	使用 Kantorovich–Rubinstein 對偶定義，導入 critic 來估計 W_1，間接逼近

🛠️ WGAN 怎麼讓 f 是 1-Lipschitz？

常見的方法有：

方法	說明
Weight Clipping（傳統WGAN作法）	將每層神經網路的權重限制在一個區間（如 $[-0.01, 0.01]$）
Gradient Penalty (WGAN-GP)	加一項 loss：懲罰梯度 norm 偏離 1 的程度
Spectral Normalization	對每層權重做奇異值正規化，保證 Lipschitz 常數不超過 1

傳統WGAN的作法為使用 weight Clipping 作法來去D為 1-Lipschitz function

↳ 但這種作法太剛性，容易去訓練不穩定

WGAN-GP改用 Gradient Penalty 的方式來作改。

↳ 不再硬性 Clipping，而是以G.P.來軟性約束，以穩定訓練。

C51 → Q值切成51等分，minimize KL-Divergence

QR-DQN → 把Q值切成去分Quantile，minimize 1-Wasserstein by Quantile Regression Loss

action-value function. $Q^*(s,a)$ satisfies the following Bellman optimality equation

$$Q^*(s,a) = \mathbb{E}_{\pi^*,P}\left[R + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a')\right]. \quad (5)$$

Eq. (5) illustrates the temporal consistency of the action-value function, which allows for the design of learning algorithms. Define the Bellman optimality operator $\mathcal{T}^*$ as

$$\mathcal{T}^* Q(s,a) = \mathbb{E}_{\pi,P}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(s',a')\right]. \quad (6)$$

When $\gamma \in (0,1)$, starting from any $Q_t(s,a)$, iteratively applying the operator $Q_{t+1}(s,a) \leftarrow \mathcal{T}^* Q_t(s,a)$ leads to convergence $Q_t(s,a) \to Q^*(s,a)$ as $t \to \infty$ [37].

In high dimensional cases, it is critical to use function approximation as a compact representation of action values. Let $Q_\theta(s,a)$ denote a function with parameter θ that approximates a table of action values with entry (s,a). The optimization aim is to find θ such that $Q_\theta(s,a) \approx Q^*(s,a)$, and the optimal solution can be found by iteratively leveraging the Bellman optimality operator $\mathcal{T}^*$. In other words, the optimal parameter θ can be approached by minimizing the squared TD error

$$\zeta^2 = \left[r + \gamma \max_{a' \in \mathcal{A}} Q_\theta(s',a') - Q_\theta(s,a)\right]^2 \quad (7)$$

over samples (s,a,r,s'), which are randomly selected from a replay buffer [38] that stores transitions which record the interaction between an agent and the environment when following the policy driven by Q_θ. In cases where $Q_\theta(s,a)$ is linear, the iterative process to find $Q^*(s,a)$ can be shown to converge [39]. However, in cases where $Q_\theta(s,a)$ is nonlinear (e.g., a neural network), $Q_\theta(s,a)$ becomes more expressive at the cost of no convergence guarantee. A number of DRL algorithms are designed following the above formulation, such as DQN [23] and Dueling DQN [35].

1) Distributional Reinforcement Learning: The main idea of distributional RL [24] is to work directly with the distribution of returns rather than their expectation (i.e., Q^π), so as to increase robustness to hyperparameter variation and environment noise [40]. Let the random variable $Z_q^\pi(s,a)$ be the return obtained by following a policy π to perform action a from the state s. Notably, the value of $Z_q^\pi(s,a)$ varies due to the unexpected randomness in the environment. Then we have

$$Q^\pi(s,a) = \mathbb{E}\left[Z_q^\pi(s,a)\right], \quad (8)$$

and an analogous distributional Bellman equation, that is,

$$Z_q^\pi(s,a) \overset{D}{=} R + \gamma Z_q^\pi(s',a'), \quad (9)$$

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

$$\mathcal{T}^* Z_q(s,a) \overset{D}{=} R + \gamma Z_q\left(s', \arg\max_{a' \in \mathcal{A}} \mathbb{E}[Z_q(s',a')]\right). \quad (10)$$

In traditional RL algorithms, we seek the optimal Q function approximator by minimizing a scalar value ζ^2 in Eq. (7). In distributional RL, our objective is to minimize a statistical distance:

$$\sup_{s,a} \operatorname{dist}\left(\mathcal{T}^* Z_q(s,a), Z_q(s,a)\right), \quad (11)$$

where $\operatorname{dist}(A,B)$ denotes the distance between random variable A and B, which can be measured by many metrics, such as KL divergence [24], p-Wasserstein [26], etc. In [24], Bellemare *et al.* proved that the distributional Bellman equation is a contraction in p-Wasserstein distance, but the distributional Bellman optimality operator is not necessarily a contraction, which provides a guideline for metric selection. C51 algorithm [24] approximates the distribution over returns using a fixed set of equidistant points and optimizes Eq. (11) by minimizing KL divergence. Different from KL divergence based on the probability density function, p-Wasserstein metric builds on the cumulative distribution function. Assume that there are two real-valued random variables U and V with respective cumulative distribution functions F_U and F_V, the p-Wasserstein between them is given by[2]

$$W_p(U,V) = \left(\int_0^1 \left|F_U^{-1}(\omega) - F_V^{-1}(\omega)\right|^p d\omega\right)^{1/p}. \quad (12)$$

Theoretically, the distributional Bellman optimality operator is a strict contraction in p-Wasserstein distance; that is, minimizing Eq. (11) with p-Wasserstein distance can give the optimal action-value distribution. QR-DQN [26] used the values on some uniformly distributed quantiles to describe the action-value distribution, leveraging the loss of quantile regression to train the neural network, which is an effective approach to minimizing 1-Wasserstein distance. Therefore, QR-DQN obtains a better balance between theory and practice by working on a special case (i.e., 1-Wasserstein distance, the special case for p-Wasserstein distance with $p = 1$).

2) Generative Adversarial Network: GAN [28] is intended to learn the distribution of data from all domains, mostly image, music, text, etc., to generate convincing data. GAN consists of two neural networks, a generator network G and a discriminator network D, which are engaged in a zero-sum game against each other. The network G takes an input from a random distribution and maps it to the space of real data. The network D obtains input data from both real data and the output of G, and attempts 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 Jensen-Shannon (JS) divergence between the real data and generated data distributions. However, [41] shows that JS metric is not continuous and does not provide a usable gradient all the time. To overcome this shortcoming, [41] proposed WGAN, in which the JS metric is replaced by 1-Wasserstein distance that provides sufficient gradients almost everywhere. Given that the equation for 1-Wasserstein distance is highly intractable, WGAN uses Kantorovich-Rubinstein duality to simplify the calculation, but introducing an essential constraint that ensures the discriminator is an appropriate 1-Lipschitz function. WGAN satisfies the

[2]We further explain the advantage of the Wasserstein metric in the next part.

WGAN-GP Obj. function 解析

G：產生圖片，D：幫圖片打分數，越高越真

$$\min_G \max_{D\in\mathcal{D}} \underbrace{\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})]}_{①} - \underbrace{\mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[D(G(\mathbf{z}))]}_{②} + p(\lambda),$$

① $\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})]$　pdata：真實的圖片集之机率分佈

x ~ pdata → x，sample from pdata，遵從pdata這个机率分布

D(x) → x 的真實度分數

→ 代表 D 幫真實圖片打的真實度分數的期望值.

② $\mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[D(G(\mathbf{z}))]$　$p_z(z)$ → 雜訊分布

z ~ $p_z(z)$ → z，sample from $p_z(z)$，遵從 p_z 這个机率分布

G(z) → G 根據 z 產出的圖片

D(G(z)) → D 幫 G(z) 打的真實度分數

→ 代表 D 幫 G 產生的假圖所打的真實度分數的期望值

回到，該 loss function 結合 G & D 的目標成一个式子：

① D：$\max_D$ $\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[D(G(\mathbf{z}))]$

→ 想最大化真圖 & 假圖的差異

② G：$\min_G$ $\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[D(G(\mathbf{z}))]$

→ 想最小化真圖 & 假圖的差異

> 產生对抗訓練

学之當達到 Nash 平衡時，G(z) 會和 pdata 重合，代表 GAN 成功學習，即 G 成功學到真實的資料分佈。 → G不能再更真，D不能再分辨 (ex. D 都以各半 0.5)

☆ 实務上，訓練不會真的達到完美的 Nash，但只要 G 能產生出真的樣本即說 GAN 成功訓練。☆ D 的學習速度会"未消失"。

* Gradient Penalty (GP)

為了 D 為 1-Lipschitz 函數。

基本想法是將 D 的梯度限制在範數 ≈ 1

① $\hat{x} = \varepsilon \mathbf{x} + (1-\varepsilon) G(\mathbf{z}),\ \varepsilon \sim U(0,1).$

- $\hat{x}$：為用來檢查 1-Lipschitz 是否成立的檢查矣。
- $\mathbf{x}$：real
- $G(\mathbf{z})$：fake
- $\varepsilon \sim U(0,1)$：決定 real / fake 的混合比例

② $p(\lambda) = \frac{\lambda}{2} \left(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1 \right)^2$

- λ：hyperparameter，用來控制懲罰的力度
- $\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})$：衡量矣 $\hat{x}$ 在 D 附近的變化率（1-Lipschitz 限制是 ≤ 1）
- $\|\cdot\|_2$：L_2 norm，表長度 or 強度
- -1：矣工的偏差（1-Lipschitz 要是 1）

這種相較於剛性的 Weight Clipping 的柔性作法，能更穩定，較不會有梯度消失的問題矣。

* System Model

State : $[d_1, d_2, d_3]$

↓

各次使用 2nd N.S. 要傳送的 packets 數

Action Space : $[w_1, w_2, w_3]$ → 這代表，在一時間都有這麼多的頻寬資源能用。

↳ Total Bandwidth = 10 MHz

Bandwidth Allocation Resolution = 1 MHz (Granularity)

$w_1 + w_2 + w_3 = 10$

$w_n \geq 1$

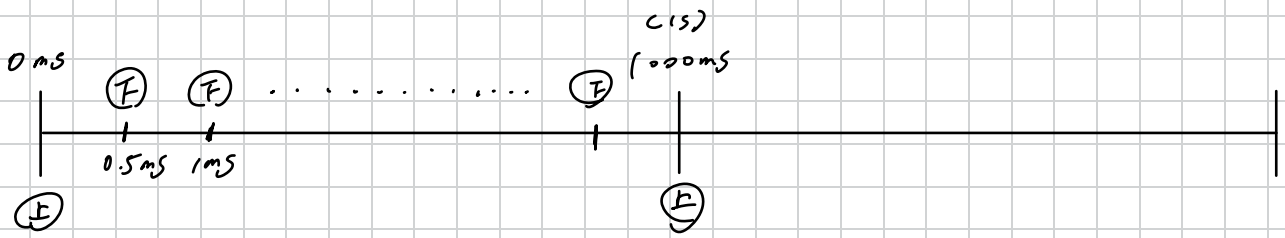
↳ $k_1 + k_2 + k_3 = 7$

$k_n \geq 0$

→ 所有的集合，共 $C_{7}^{7+3-1} = C_{2}^{9} = 36$

|Action Space| = 36

* time sequence



(上)：分 Bandwidth 給 N.S. by GAN-DDQN ：每 1s 執行一次

(下)：將 N.S. 分到的 Band. 分給 active UE, by RR：每 0.5 ms 執行一次

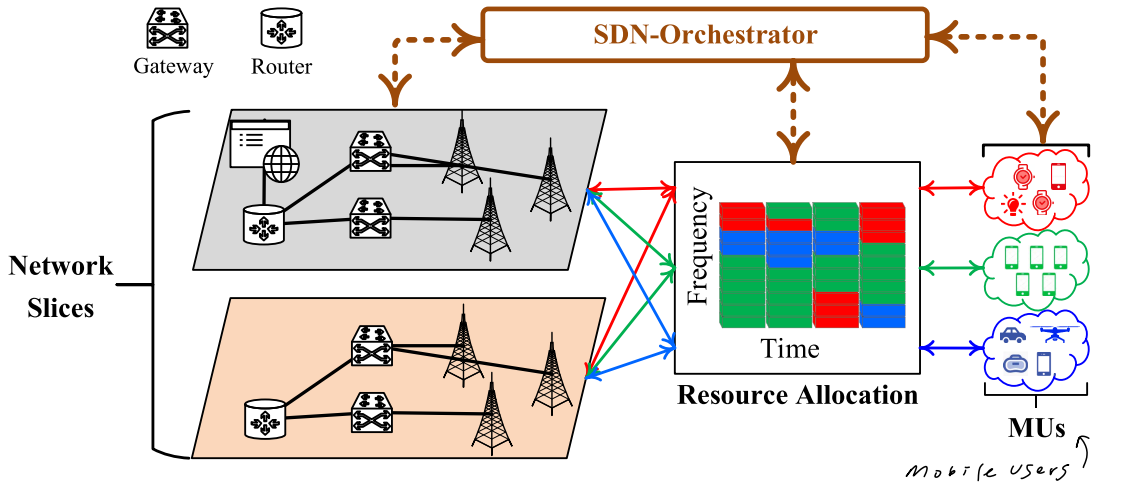


Fig. 1. The considered scenario showing uplink and downlink transmissions on different NSs.

constraint by clipping the weights of the discriminator to be within a certain range that is governed by a hyperparameter. Furthermore, [42] proposed WGAN-GP and adopted gradient penalty to enforce the 1-Lipschitz constraint instead of simply clipping weights. Its optimization objective is formulated as follow:

$$\min_G \max_{D\in\mathcal{D}} \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[D(G(\mathbf{z}))] + p(\lambda), \quad (13)$$

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 $p(\lambda) = \frac{\lambda}{2}\left(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1\right)^2$, $\hat{\mathbf{x}} = \varepsilon\mathbf{x} + (1-\varepsilon)G(\mathbf{z})$, $\varepsilon \sim U(0,1)$. Gradient penalty increases the computational complexity but it does make WGAN-GP perform much better than previous GANs.

B. System Model

Fig. 1 illustrates the SDN-based system model for dynamic allocation of wireless bandwidth in the RAN scenario with uplink and downlink transmissions. We consider the downlink case in this paper. In this respect, [43] built an SDN-based C-RAN testbed that realizes the dynamic allocation of radio resources (e.g., wireless bandwidth) by using frequency division duplex scheme, and the demonstration was presented in [44]. Under the framework of hierarchical network slicing, we consider a RAN scenario with a single BS, where a set $\mathcal{N}$ of NSs share the aggregated bandwidth W.[3] The bandwidth is allocated to each NS according to the number of demands for the corresponding type of service. For an NS, say NS n, it provides a single service for a set of users $\mathcal{U}_n$. We consider a timeslot model where the slicing decision is updated according to the demand of users periodically (e.g., 1 second). In one timeslot, the number of demands NS n receives is denoted as d_n, which partially determines the wireless bandwidth that the BS allocates to this NS, denoted as w_n.

[3]In fact, such a bandwidth allocation could be realized by physical multiplexing methods [45]. Meanwhile, the shared resources could be temporal slots as well. However, for simplicity of representation, we take the bandwidth allocation problem as an example.

The objective of our work is to find an optimal bandwidth-allocation solution that maximizes the system utility, denoted by J, which can be described by the weighted sum of SE and SSR. We now study the two sub-objectives, respectively. Let r_{u_n} be the downlink rate of user u_n served by NS n, which is, for simplicity, defined by Shannon theory as follows

$$r_{u_n} = w_{u_n}\log(1+\text{SNR}_{u_n}), \quad \forall u_n \in \mathcal{U}_n, \quad (14)$$

where SNR_{u_n} is the signal-to-noise-ratio between user u_n and the BS. SNR_{u_n} can be given as

$$\text{SNR}_{u_n} = \frac{g_{u_n}P_{u_n}}{N_0 w_n}, \quad (15)$$

where g_{u_n} is the average channel gain that captures path loss and shadowing from the BS to the user u_n, P_{u_n} is the transmission power, and N_0 is the single-side noise spectral density. Given the transmission rate, SE can be defined as follows

$$\text{SE} = \frac{\sum_{n\in\mathcal{N}}\sum_{u_n\in\mathcal{U}_n} r_{u_n}}{W}. \quad (16)$$

On the other hand, SSR of NS n is obtained by dividing the number of successfully transmitted packets by the total number of arrived packets on NS n. Before formulating this problem, we define $\mathcal{Q}_{u_n}$ as the set of packets sent from the BS to user u_n, determined by the actual traffic demand patterns, and define a binary variable $x_{q_{u_n}} \in 0,1$, where $x_{q_{u_n}} = 1$ indicates that the packet $q_{u_n} \in \mathcal{Q}_{u_n}$ is successfully received by user u_n, i.e., the downlink data rate r_{u_n} and the latency $l_{q_{u_n}}$ are simultaneously satisfied. Therefore, $x_{q_{u_n}} = 1$ if and only if $r_{u_n} \geq \overline{r}_n$ and $l_{q_{u_n}} \leq \overline{l}_n$, where $l_{q_{u_n}}$ denotes the latency that takes account of both queuing delay and transmission delay. $\overline{r}_n$ and $\overline{l}_n$ are the predetermined rate and latency values according to the SLA for service type n. We can formulate the SSR for NS n as:

$$\text{SSR}_n = \frac{\sum_{u_n\in\mathcal{U}_n}\sum_{q_{u_n}\in\mathcal{Q}_{u_n}} x_{q_{u_n}}}{\sum_{u_n\in\mathcal{U}_n}|\mathcal{Q}_{u_n}|}, \quad (17)$$

where $|\mathcal{Q}_{u_n}|$ denotes the number of packets sent from the BS to user u_n.

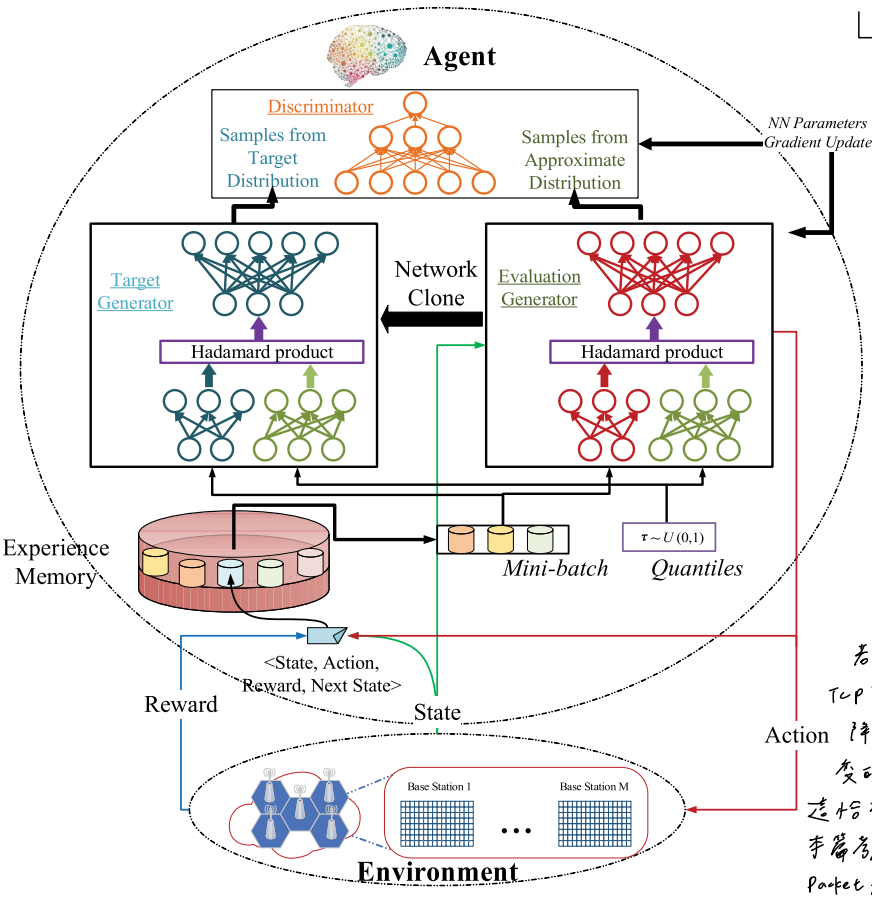


Fig. 2. An illustration of GAN-DDQN for resource management in network slicing.

The bandwidth allocation problem in the RAN network slicing is formulated as follows

$$\max_{w_n} \ \alpha \mathrm{SE} + \sum_{n\in\mathcal{N}} \beta_n \cdot \mathrm{SSR}_n = \max_{w_n} \ \alpha \frac{\sum_{n\in\mathcal{N}} \sum_{u_n\in\mathcal{U}_n} r_{u_n}}{W} + \sum_{n\in\mathcal{N}} \beta_n \cdot \frac{\sum_{u_n\in\mathcal{U}_n} \sum_{q_{u_n}\in\mathcal{Q}_{u_n}} x_{q_{u_n}}}{\sum_{u_n\in\mathcal{U}_n} |\mathcal{Q}_{u_n}|}, \tag{18}$$

$$\text{s.t.} \ \sum_{n\in\mathcal{N}} w_n = W, \tag{19}$$

$$\sum_{u_n\in\mathcal{U}_n} |\mathcal{Q}_{u_n}| = d_n, \tag{20}$$

$$x_{p_{u_n}} = \begin{cases} 1, & r_{u_n} \geq \overline{r}_n \ \ \& \ \ l_{p_{u_n}} \leq \overline{l}_n, \\ 0, & otherwise. \end{cases} \tag{21}$$

where α and $\boldsymbol{\beta} = [\beta_1, \beta_2, \cdots \beta_n]$ are the coefficients that adjust the importance of SE and SSR, and β_n refers to the importance weight of SSR_n. In this problem, the objective is to maximize two components: (a) the spectral efficiency (i.e., SE), and (b) the proportion of the packets satisfying the constraint of data rate and latency (i.e., SSR).

Notably, in the current timeslot, d_n depends on both the number of demands and the bandwidth-allocation solution in the previous timeslot, since the maximum transmission capacity of RAN 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. Therefore, the traffic demand varies without knowing a prior transition probability, making Eq. (18) difficult to yield a direct solution. However, RL promises to be applicable to tackle this kind of problem. Therefore, we refer to RL to find the optimal policy for network slicing. In particular, consistent with [13], we map the RAN scenario to the context of RL by taking the number of arrived packets in each slice within a specific time window as the state, and the bandwidth allocated to each slice as the action.

III. GAN-Powered Deep Distributional Q Network

In this section, we describe the proposed GAN-DDQN algorithm, shown in Fig. 2, that address the demand-aware resource allocation problem in network slicing. We then discuss the methods of improving the performance of the algorithm and analyze its convergence.

A. GAN-DDQN Algorithm

Our previous work [13] has discussed how to apply RL to the resource slicing problem. However, the DQN algorithm used in that work is based on the expectation of the action-value distribution, and thus does not take into account

Code中Shannon香農的單位推導

$$dB = 10 \cdot \log_{10}\left(\frac{P}{B}\right)$$

$$dBW = 10 \cdot \log_{10}\left(\frac{P}{1W}\right)$$

↳ 從 dBW (相對單位) 轉回 W (絕對單位)

$$A\,(dBW) = 10 \cdot \log_{10}\left(\frac{P}{1W}\right)$$

$$\frac{A}{10} = \log_{10}\left(\frac{P}{1W}\right)$$

$$10^{\frac{A}{10}} = \frac{P}{1W} \Rightarrow P = 10^{\frac{A}{10}}\ W$$

$BS\_tx\_power = 16\ dBW$

$$rx\_power_{dBW} = BS\_tx\_power_{dBW} - chan\_loss_{dB} + UE\_rx\_gain_{dB}$$

↓ 為算SNR, 轉回 W

$$rx\_power_{W} = 10^{\left(\frac{rx\_power}{10}\right)}$$

rx_power_UE W

$$rate\_UE_{bits/s} = band\_UE_{Hz} * \log_{10}\left(1 + \frac{P_{signal}}{P_{noise}}\right)$$

$bits/次$

$band\_UE$ Hz → 次/s

$$\left(noise\_PSD_{dBW/Hz} \xrightarrow{\text{轉成 W}} noise\_PSD_{W/Hz}\right) * band\_UE_{Hz}$$

W

* Hadamard Product 以外的交互方法

方法	問題/效果
concat	→ 只是把 state 跟 quantile 接起來，沒有做交互作用（平行資訊）
add	→ 必須向量有相同意義與單位，但 state 和 quantile 嵌入意義差很大
✅ Hadamard	→ **逐維交互作用**：每一維的特徵都會根據 quantile 調整

* 以 batch 角度, 各 Model 的 input / output shape

input_shape　　self.state_size　　output_shape

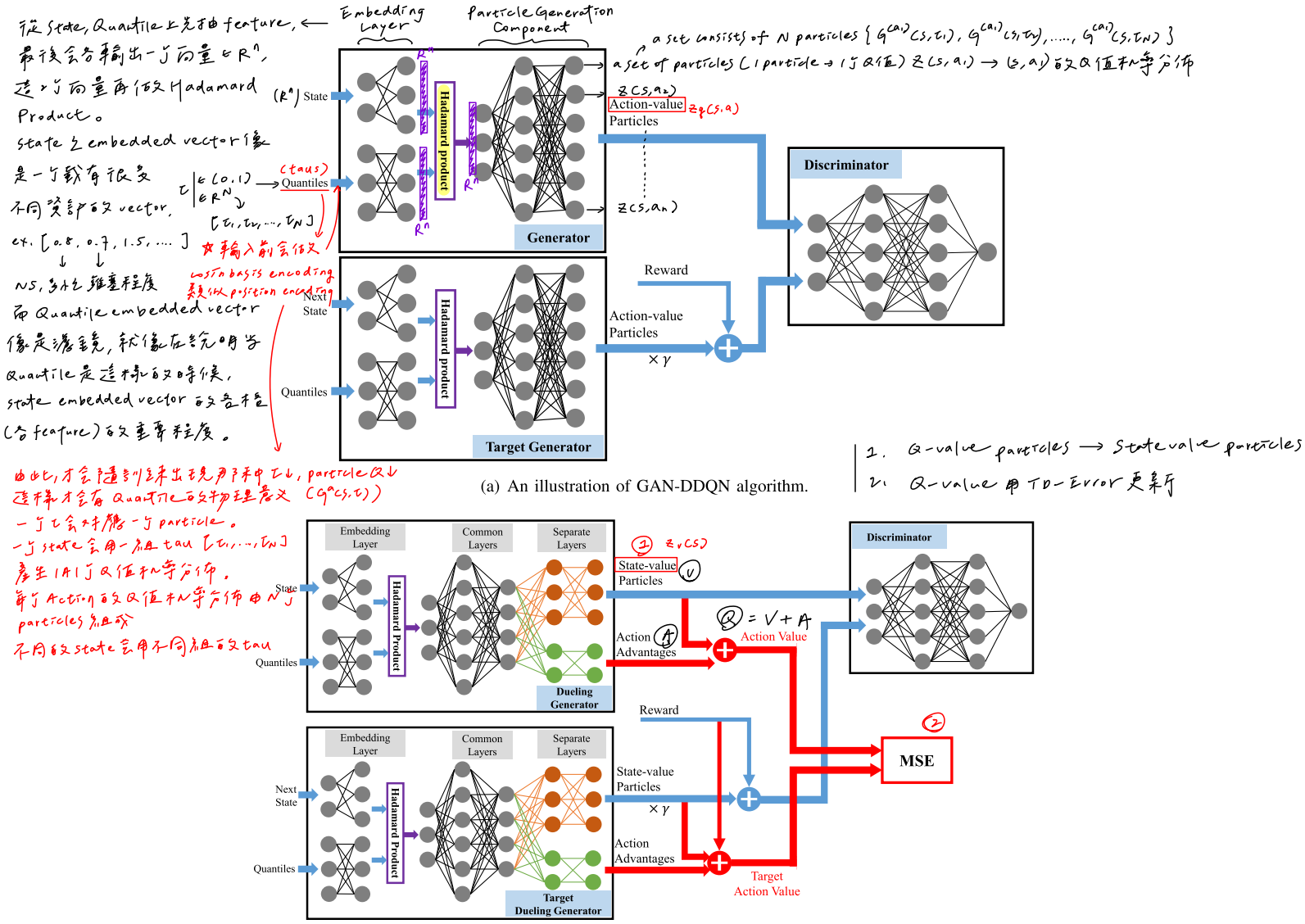
Ⓖ State : (self.batch_size * self.num_samples,) (self.batch_size, self.num_actions, self.num_samples)

Quantiles : (self.batch_size * self.num_samples,)

self.embedding_dim

Ⓓ (self.batch_size, self.num_actions, self.num_samples) (self.batch_size, self.num_actions, self.num_outputs)

1



(a) An illustration of GAN-DDQN algorithm.

(b) An illustration of Dueling GAN-DDQN algorithm.

Fig. 3. The comparison of GAN-DDQN and Dueling GAN-DDQN.

the adverse effects of random noise on the received values of SE and SSR. To overcome this problem, we resort to the combination of the distributional RL and GAN. In this regard, we introduce WGAN-GP to learn the optimal action-value distribution. Specifically, the generator network G outputs a fixed number of samples (we refer to them as *particles* for clarity) that characterize the estimated action-value distribution learned by network G. Similar to [23], we leverage a target generator network $\hat{G}$ to obtain the target action-value particles. The discriminator network D realizes the 1-Wasserstein criterion when it attempts to minimize the distance between the estimated action-value particles and the target action-value particles calculated by the Bellman optimality operator. GAN-DDQN is able to approximate the optimal action-value distribution by alternately updating networks G and D.

Before we introduce the details of GAN-DDQN algorithm, it is necessary to describe the structure of the networks G and D. Network G consists of three components, which are responsible for state embedding, sample embedding, and particles generation. The state-embedding and sample-embedding components are both built with two neural layers and process the input state and the quantile samples in parallel. Then, the output of these two components are combined through Hadamard product operation. This step is consistent with [27] to force interaction between the state embedding and sample embedding. Afterwards, the particles generation component, which contains multiple neural layers, takes the fused information as input, outputting several sets of particles where each set is treated as a representation of the corresponding action-value distribution. On the other hand, network D is a multilayer perceptron (MLP) with one neuron in the output layer. Fig. 3(a) further details the structure of GAN-DDQN.

The GAN-DDQN algorithm can be explained as follows, without loss of generality. At iteration t, the agent feeds the current state $S_t = s$ and the samples τ from a uniform distribution (e.g., $U(0,1)$) to network G; τ is the quantile values of the action-value distribution [27]. Network G outputs a set of estimated action-value particles, denoted as $G(s,\tau)$,

where the particles belonging to action a are denoted as $G^{(a)}(s,\tau)$; the number of $G^{(a)}(s,\tau)$ is N. Then, the agent calculates $Q(s,a) = \frac{1}{N}\sum G^{(a)}(s,\tau), \forall a \in \mathcal{A}$, and selects $a^* = \arg\max_a Q(s,a), \forall a \in \mathcal{A}$ to perform. As a result, the agent receives a reward r, and the environment moves to the next state $S_{t+1} = s'$. The tuple (s, a^*, r, s') is stored into the replay buffer $\mathcal{B}$. When $\mathcal{B}$ is full, the agent updates networks G and D using all the transition tuples in $\mathcal{B}$ every K iterations.

In the training and updating process, the agent first randomly selects m transitions from $\mathcal{B}$ as a minibatch for training GAN-DDQN. Then, the agent executes the Bellman optimality operator on each transition of the selected minibatch and obtains the target action-value particles. For example, the target action-value particles for the transition i is $y_i = r_i + \gamma \hat{G}^{(a_i^*)}(s_i', \tau_i)$ where a_i^* is the action with the maximum expectation of action-value particles, i.e., $a_i^* = \arg\max_a \frac{1}{N}\sum \hat{G}^{(a)}(s_i', \tau_i)$. Finally, the agent uses the following loss functions to train networks D and G, respectively:

$$\mathcal{L}_D = \mathop{\mathbb{E}}_{\substack{\tau\sim U(0,1)\\(s,a)\sim\mathcal{B}}}[D(G^{(a)}(s,\tau))] - \mathop{\mathbb{E}}_{(s,a,r,s')\sim\mathcal{B}}[D(y)] + p(\lambda), \quad (22)$$

$$\mathcal{L}_G = -\mathop{\mathbb{E}}_{\substack{\tau\sim U(0,1)\\(s,a)\sim\mathcal{B}}}[D(G^{(a)}(s,\tau))] \quad (23)$$

where $p(\lambda)$ is as mentioned in Eq. (13). The training goal for network D is to increase its accuracy in distinguishing the target action-value particles from the action-value particles produced by network G. The goal of training network G, on the other hand, is to improve its ability to generate the action-value particles that "fool" network D as much as possible. Note that in order to further stabilize the training process, we update the target network $\hat{G}$ every C iterations.

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

Algorithm 1 GAN-DDQN

1: Initialize a generator G and a discriminator D with random weights $\boldsymbol{\theta}_G$ and $\boldsymbol{\theta}_D$ respectively, the number of particles N, gradient penalty coefficient λ, batch size m, discount factor γ.
2: Initialize a target generator $\hat{G}$ with weight $\boldsymbol{\theta}_{\hat{G}} \leftarrow \boldsymbol{\theta}_G$, a replay buffer $\mathcal{B} \leftarrow \varnothing$, the iteration index $t = 0$.
3: **repeat**
4: The agent observes $S_t = s$.
5: The agent samples $\tau \sim U(0,1)$.
6: The agent calculates $Q(s,a) = \frac{1}{N}\sum G^{(a)}(s,\tau), \forall a \in \mathcal{A}$.
7: The agent performs $a^* \leftarrow \arg\max_a Q(s,a), \forall a \in \mathcal{A}$.
8: The agent receives the system utility J and observes $S_{t+1} = s'$.
9: The agent performs the reward-clipping with respect to J and gets the reward r.
10: The agent stores transition (s, a^*, r, s') in $\mathcal{B}$.
11: If $\mathcal{B}$ is full, the agent updates the weights of network G and network D every K iterations.
12: *# Train GAN*
13: **repeat**
14: The agent samples a minibatch $\{s, a, r, s'\}_{i=1}^m$ from $\mathcal{B}$ without replacement.
15: The agent samples a minibatch $\{\tau\}_{i=1}^m \sim U(0,1)$.
16: The agent gets the target action-value particles $y_i = r_i + \gamma \hat{G}^{(a_i^*)}(s_i', \tau_i)$, where the optimal action is $a_i^* = \arg\max_a \frac{1}{N}\sum \hat{G}^{(a)}(s_i', \tau_i), \forall a \in \mathcal{A}$.
17: The agent samples a minibatch $\{\varepsilon\}_{i=1}^m \sim U(0,1)$, and sets $\hat{x}_i = \varepsilon_i y_i + (1-\varepsilon_i) G^{(a_i)}(s_i, \tau_i)$.
18: 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 $\mathcal{L}_i = D(G^{(a_i)}(s_i,\tau_i)) - D(y_i) + \lambda(\|\nabla_{\hat{x}_i} D(\hat{x}_i)\|_2 - 1)^2$.
19: The agent updates the weights $\boldsymbol{\theta}_G$ by leveraging gradient descent algorithm to $-\frac{1}{m}\sum_{i=1}^m D(G^{(a_i)}(s_i,\tau_i))$
20: **until** All the transitions in $\mathcal{B}$ are used for training.
21: The agent clones network G to the target network $\hat{G}$ every C iterations by resetting $\boldsymbol{\theta}_{\hat{G}} = \boldsymbol{\theta}_G$.
22: The iteration index is updated by $t \leftarrow t + 1$.
23: **until** A predefined stopping condition (e.g., the $\frac{1}{m}\sum_{i=1}^m \mathcal{L}_i$, the preset number of iterations, etc.) is satisfied.

B. Convergence Analysis

It has been proven in [24] that the distributional RL can converge when the metric for diverging distributions is p-Wasserstein distance. On the other hand, the fundamental guidance for distinguishing the target and estimated distributions in WGAN-GP is 1-Wasserstein distance. Therefore, the convergence of GAN-DDQN can be analyzed from the perspective of WGAN-GP's convergence on the data sampled from the dynamic RL interaction process. However, as explored in [46], in many currently popular GAN architectures, converging to the target distribution is not guaranteed and oscillatory behavior can be observed. This is a twofold challenge for GAN-DDQN, as we must ensure both the ① stationarity of the target distribution and the ② convergence of the WGAN-GP to this target distribution.

In an idealized WGAN-GP, the generator should be able to learn from the target distribution, and the discriminator should be able to learn any 1-Lipschitz function to produce the exact Wasserstein distance. However, the target distribution will not be stationary as the target network $\hat{G}$ regularly updates its weights; thus an idealized WGAN-GP might not be successful in practice. Fortunately, a slight change in the target distribution has little effect on the convergence of WGAN-GP. For example, suppose the real distribution that is the ultimate learning goal of WGAN-GP is a Gaussian distribution with a mean of 100 and a standard deviation of 1, and suppose that the target distribution that the WGAN-GP is expected to approximate at each stage is a Gaussian distribution with a standard deviation of 1 and a mean that starts at 0, increasing periodically by $\Delta\mu$. WGAN-GP will need more updates to learn the target distribution if $\Delta\mu$ is large; the number of

updates is difficult to determine. However, if $\Delta\mu$ is small, a few times of updates is sufficient for WGAN-GP to learn the changed target distribution. Hence, the small $\Delta\mu$ is more potential to enable WGAN-GP to learn the real distribution smoothly.

To analyze the convergence characteristic of WGAN-GP while avoiding directly dealing with sophisticated data and WGAN-GP model, [46] introduces a simple but illustrative model, namely Dirac-WGAN-GP. Specifically, Dirac-WGAN-GP consists of a linear discriminator $D_\psi(x) = \psi \cdot x$ and a generator with parameter θ that indicates the position of the Dirac distribution (i.e., δ_θ) output by the generator. Whilst the real data distribution p_d is given by a Dirac-distribution concentrated at ξ (i.e., δ_ξ). It is worthy to further investigate the training characteristic of Dirac-WGAN-GP when the real data distribution (i.e., δ_ξ) is varying during the training process, like the typical situation in RL. Consistent with [46], we carry out analysis based on Dirac-WGAN-GP, and we have the following Theorem 1

Theorem 1: When trained with gradient descent with a fixed number of the generator and the discriminator updates and a fixed learning rate $h > 0$, if the value of ξ varies dramatically, Dirac-WGAN-GP needs more learning steps to converge from the old optimal boundary to the new one after the variation of ξ.

We leave the proof of Theorem 1 in Appendix. As for WGAN-GP, we further have the following Corollary 1

Corollary 1: WGAN-GP could converge to the optimal boundary more rapidly if the real data change by a small amount.

Corollary 1 reveals that estimating the optimal action-value distribution requires a large amount of training if we directly use the system utility as the reward in RL. Therefore, we put forward a new reward-clipping mechanism to prevent the target action-value distribution from greatly changing. Specifically, (assuming that there are T thresholds that partition the system utility, we set $T+1$ constants whose values are much smaller than the system utility.) Then the system utility can be clipped to these $T+1$ constants that are taken as the rewards in RL. For example, if $T = 2$ and the clipping constants are $-\eta$, 0, and η ($\eta > 0$), then the clipping strategy can be formulated by Eq. (24), where c_1 and c_2 ($c_1 > c_2$) are the manually set thresholds:

$$r = \begin{cases} \eta, & J(\mathbf{w}, \mathbf{d}) \geq c_1, \\ 0, & c_2 < J(\mathbf{w}, \mathbf{d}) < c_1, \\ -\eta, & J(\mathbf{w}, \mathbf{d}) \leq c_2. \end{cases} \tag{24}$$

However, as T becomes larger, the number of the manually set parameters in the reward-clipping mechanism increases, which makes the parameter setting process more sophisticated. Therefore, we adopt the reward-clipping mechanism defined in Eq. (24) as an experiment. Note that introducing the reward-clipping mechanism to GAN-DDQN algorithm is effortless, and we only need to apply the reward-clipping mechanism to the system utility before storing the transition tuple in the replay buffer, which is described in line 9 of Algorithm 1.

C. Dueling GAN-DDQN

The training of GAN-DDQN is not a trivial task since it uses the data yielded from a dynamic environment, and only a tiny portion of the output of the generator is useful for gradient calculation. One intuitive indicating to alleviate the training problem is to carefully adjust the values of GAN-DDQN's hyper-parameters, such as the discount factor γ, the gradient penalty coefficient λ, etc. Nevertheless, we plan to make systemic and architectural changes to the generator and the loss function. Particularly, inspired by [36], which uses a specialized dueling Q network to separate the action value into a state-value stream and an advantage stream, we divide the approximation of the action-value distribution into the approximation of the state-value distribution and the approximation of the advantage function for each action. This dueling architecture ignores the trivial variations of the environment and focuses on some crucial states to enhance the stability of DRL algorithms [35]. In addition, in our improved model, namely Dueling GAN-DDQN, the loss function of the discriminator turns to work on the estimated and target state-value distributions. Moreover, the squared TD error is added to the generator's loss as the criterion that measures the distance of the estimated and target action values.

The detailed structure of Dueling GAN-DDQN is presented in Fig. 3(b), and we remarkably highlight the key differences from GAN-DDQN. It can be observed that the significant difference compared with GAN-DDQN is the generator or the dueling generator for the sake of distinguishing. In the dueling generator, after Hadamard product operation, we continue to handle the output using multiple neural layers (the *common layers*). Then, the refined information is separated into two paths, one flowing to a neural network to approximate the state-value distribution, and the other flowing to another neural network to estimate the action advantage function. Accordingly, the dueling generator outputs not only particles from the approximated state-value distribution but also the estimated action advantage values for each action. Note that the discriminator of Dueling GAN-DDQN has the same structure as GAN-DDQN.

Similarly to our analysis of the random variable Z_v^π, we analyze the random variable $Z_v^\pi(s)$, which denotes the return obtained by following a policy π from state s. Then we have

$$V^\pi(s) = \mathbb{E}\left[Z_v^\pi(s)\right], \tag{25}$$

and an analogous distributional Bellman equation for Z_v

$$Z_v^\pi(s) \overset{D}{=} \mathop{\mathbb{E}}_{\substack{a\sim\mathcal{A} \\ s'\sim\mathcal{S}}}[R + \gamma Z_v^\pi(s')]. \tag{26}$$

It is difficult to find the distributional Bellman optimality operator for Z_v^π. Even worse, Eq. (26) indicates that the iterative calculation of Z_v^π requires a reward from every state-action pair, which is a noticeable time-consuming operation. Therefore, we introduce a degraded but simplified method to estimate Z_v^π, which is to minimize the difference between the estimated Z_v^π and $\mathcal{T}Z_v^\pi$ calculated by

$$\mathcal{T}Z_v^\pi \overset{D}{=} r + \gamma Z_v^\pi(s'), \tag{27}$$

where r and s' are from the transition (s, a, r, s') sampled from the replay buffer. This degraded approximation may fail to find the optimal state-value distribution, yet it can significantly reduces computation time. In addition, only considering the 1-Wasserstein loss for the state-value distribution results in the network G weights related to the action advantage function not being trained. Therefore, we leverage the TD error to measure the difference of the estimated and the target action values, where the action value is calculated by adding the corresponding action advantage value to the mean of the state-value particles. As a consequence, the ultimate loss function for training Dueling GAN-DDQN is composed of the 1-Wasserstein distance and the squared TD error, which can be formulated as follows

$$\mathcal{L}_D = \mathop{\mathbb{E}}_{\substack{\tau\sim U(0,1)\\(s,a)\sim\mathcal{B}}}[D(G_v(s,\tau))] - \mathop{\mathbb{E}}_{\substack{\tau\sim U(0,1)\\(r,s')\sim\mathcal{B}}}[D(r+\gamma\hat{G}_v(s',\tau))] + p(\lambda), \tag{28}$$

$$\mathcal{L}_G = -\mathop{\mathbb{E}}_{\substack{\tau\sim U(0,1)\\(s,a)\sim\mathcal{B}}}[D(G_v(s,\tau))] + \frac{1}{2}\zeta^2, \tag{29}$$

where G_v denotes the state-value particles output by the dueling generator, and ζ^2 is the squared TD error as defined in Eq (7). Algorithm 2 and Fig. 3(b) provide the details of Dueling GAN-DDQN.

IV. Simulation Results and Numerical Analysis

A. Simulation Environment Settings

In this part, we verify the performance of GAN-DDQN and Dueling GAN-DDQN in a RAN scenario where there are three types of services (i.e., VoLTE, video, and URLLC) and three corresponding slices in one serving BS, as in [13]. There exist 100 registered subscribers randomly located within a 40-meter-radius circle surrounding the BS. These subscribers generate standard service traffics as summarized in Table II based on 3GPP TR 36.814 [47] and TS 22.261 [48]. The total bandwidth is 10 MHz, and the bandwidth allocation resolution is (1 MHz or 200 KHz.) We will show the simulation results for both cases. On the other hand, the packet size of URLLC service has a strong influence on the system utility. For example, it is difficult to meet the latency requirement of URLLC service when the packet size is large, if there is insufficient bandwidth guaranteed for transmission. As a result, SSR degrades, and the system utility is reduced. Therefore, we simulate the network slicing scenario with suitably-sized URLLC packets.

With the mapping shown in Table III, RL algorithms can be used to optimize the system utility (i.e., the weighted sum of SE and SSR). Specifically, we perform round-robin scheduling within each slice at 0.5 ms granularity; that is, we sequentially allocate the bandwidth of each slice to the active users within each slice every 0.5 ms. Besides, we adjust the bandwidth allocation to each slice per second. Therefore, the agent updates its neural networks every second. (Considering the update interval of the bandwidth-allocation process is much larger than the service arrival interval, the number of arrived packets (i.e., the state) is rarely zero when updating the agents. Therefore, it is reasonable to ignore the situation of zero bandwidth for any NS.) Moreover, this filter setting narrows the range for the action exploration, as well as enhancing the

Algorithm 2 Dueling GAN-DDQN

1: Initialize a dueling generator G and a discriminator D with random weights $\boldsymbol{\theta}_G$ and $\boldsymbol{\theta}_D$ respectively, the number of particles N, gradient penalty coefficient λ, batch size m, discount factor γ, $n_{critic} = 5$.
2: Initialize a target dueling generator $\hat{G}$ with weight $\boldsymbol{\theta}_{\hat{G}} \leftarrow \boldsymbol{\theta}_G$, a replay buffer $\mathcal{B} \leftarrow \varnothing$, the iteration index $t = 0$.
3: **repeat**
4: The agent observes $S_t = s$.
5: The agent samples $\tau \sim U(0,1)$.
6: The agent feeds s and τ to network G, getting the state-value particles $G_v(s,\tau)$ and each action advantage value $G_{ad}^{(a)}(s,\tau), \forall a \in \mathcal{A}$.
7: The agent calculates $V(s) = \frac{1}{N}\sum G_v(s,\tau)$.
8: The agent calculates $Q(s,a) = V(s) + G_{ad}^{(a)}(s,\tau), \forall a \in \mathcal{A}$.
9: The agent performs $a^* \leftarrow \arg\max_a Q(s_t, a)$.
10: The agent receives the system utility J and observes a new state $S_{t+1} = s'$.
11: The agent performs the reward-clipping with respect to J and gets the reward r.
12: The agent stores transition (s, a^*, r, s') in $\mathcal{B}$.
13: *# Train network D*
14: **for** $n = 1$ to n_{critic} **do**
15: The agent randomly samples $\{s, a, r, s'\}_{i=1}^m$ from $\mathcal{B}$.
16: The agent samples $\{\tau\}_{i=1}^m$ and $\{\varepsilon\}_{i=1}^m$ from $U(0,1)$.
17: The agent gets $y_i = G_v(s_i, \tau_i)$, and $\hat{y}_i = r_i + \gamma\hat{G}_v(s_i', \tau_i)$.
18: The agent sets $\hat{x}_i = \varepsilon_i\hat{y}_i + (1-\varepsilon_i)y_i$.
19: 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 $\mathcal{L}_i = D(y_i) - D(\hat{y}_i) + \lambda\left(\|\nabla_{\hat{x}_i} D(\hat{x}_i)\|_2 - 1\right)^2$.
20: **end for**
21: *# Train network G*
22: The agent randomly samples $\{s, a, r, s'\}_{i=1}^m$ from $\mathcal{B}$.
23: The agent samples $\{\tau\}_{i=1}^m$ from $U(0,1)$.
24: The agent calculates the estimated action value $Q_i = \frac{1}{N}\sum G_v(s_i,\tau) + G_{ad}^{(a_i)}(s_i,\tau)$, and the target action value $\hat{Q}_i = r_i + \gamma\frac{1}{N}\sum G_v(s_i',\tau) + \gamma\max_a G_{ad}^{(a)}(s_i',\tau), \forall a \in \mathcal{A}$.
25: The agent updates the weights $\boldsymbol{\theta}_G$ by leveraging gradient descent algorithm to $\frac{1}{m}\sum_{i=1}^m[-D(G_v(s_i)) + \frac{1}{2}(\hat{Q}_i - Q_i)^2]$.
26: The agent clones network G to the target network $\hat{G}$ every C iterations by resetting $\boldsymbol{\theta}_{\hat{G}} = \boldsymbol{\theta}_G$.
27: The iteration index is updated by $t \leftarrow t+1$.
28: **until** A predefined stopping condition (e.g., the $\frac{1}{m}\sum_{i=1}^m \mathcal{L}_i$, the preset number of iterations, etc.) is satisfied.

eMBB 採 Pareto 分佈 (base=1.2)

Pareto 分布是一種 **heavy-tailed** 分布，用來模擬：

- 大量小事件、少數大事件的情況
- 比如：大多數人很少上網，但有一小部分人流量爆炸高（比如一直在看 4K 影片）

Pareto(α) 的特性是：

- 越小的 α（例如 α = 1.2），尾巴越粗（也就是高機率出現極大值）
- 它適合用來模擬 video streaming、web traffic 等 bursty 行為

📘 **為何乘上 `6ms`？**

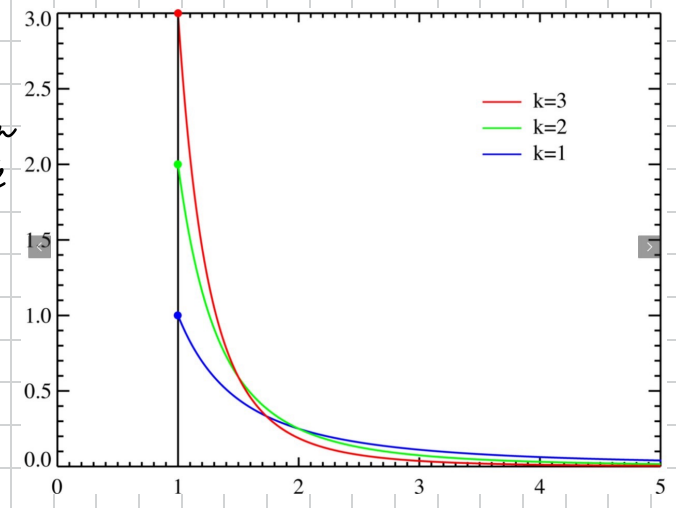
```
python                                   Copy  Edit

tmp_readtime = pareto(1.2) x 6 ms
```

代表：一個 burst 平均持續時間大約是幾毫秒（這裡以 6ms 為單位），但偶爾會出現極大的時間間隔。

🎯 **總結這行的物理意義：**

> 為每個 eMBB UE 建立一個隨機「封包產生時間間隔」，模擬影片串流中偶爾產生的 bursty traffic（連續出現資料）、或 silent interval（較長時間沒封包）。

和等



Packet 大小

URLLC 採 Exponential 分佈

✅ **指數分布的特性：**

- 記憶無關（memoryless）：未來的事件發生機率與過去無關
- 常用來模擬 **隨機突發事件發生的間隔**
- PDF 為：`f(x) = λ e^{-λx}`，平均為 `1/λ`

🚀 **為什麼適合 URLLC？**

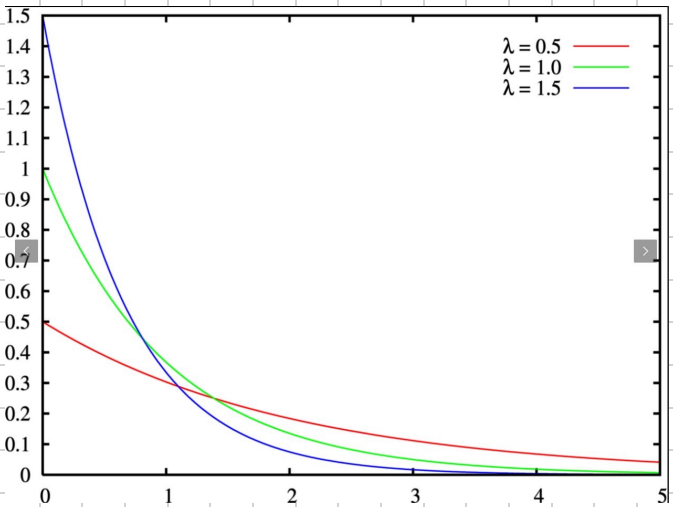
URLLC（Ultra-Reliable Low-Latency Communications）通常是：

- **突發的事件觸發**（例如：感測器發送緊急資訊、車輛撞擊偵測）
- 並不是週期性的傳輸
- 所以封包間隔時間應該是**隨機且稀疏的**

使用 **指數分布**可以很自然地模擬：

> URLLC 封包是在一段平均為 180ms 的時間內突發發生。

和等



Packet 大小

TABLE II

A BRIEF SUMMARY OF KEY SETTINGS FOR TRAFFIC GENERATION PER SLICE

	VoLTE	Video (eMBB)	URLLC
Bandwidth	20 MHz		
Scheduling	Round robin per slot (0.5 ms)		
Slice Band Adjustment (Q-Value Update)	1 second (2000 scheduling slots)		
Channel	Rayleigh fading → only consider large-scale fading		
User No. (100 in all)	46	46	8
Distribution of Inter-Arrival Time per User	Uniform [Min = 0, Max = 160ms]	Truncated stationary distribution [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]	Variable constant: {6.4, 12.8, 19.2, 25.6, 32} KByte or {0.3, 0.4, 0.5, 0.6, 0.7} MByte
SLA: Rate	51 Kbps	100 Mbps	10 Mbps
SLA: Latency	10 ms	10 ms	1 ms

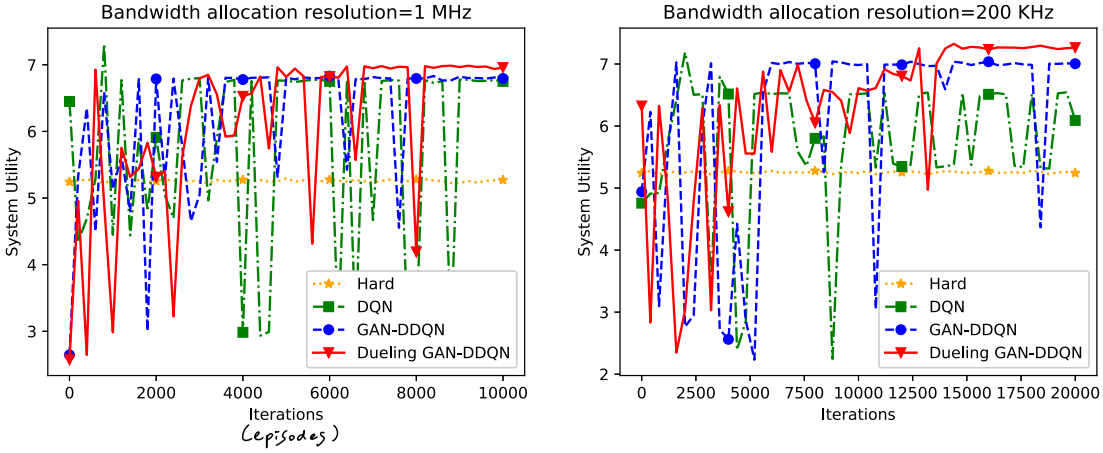


Fig. 4. An illustration of performance comparison between different slicing schemes (hard slicing, DQN, GAN-DDQN, Dueling GAN-DDQN).

TABLE III

THE MAPPING FROM RESOURCE MANAGEMENT FOR NETWORK SLICING TO RL ENVIRONMENT

RL Environment	Radio Resource Slicing
State	The number of arrived packets in each slice within a specific time window
Action	bandwidth allocation to each slice
Reward	Clipped weighted sum of SE and SSR in 3 sliced bands

stability of the training process. Meanwhile, this filter setting does affect our main results.

B. Simulation Results

In this part, we show the simulation results of the proposed GAN-DDQN and Dueling GAN-DDQN algorithms, in comparison with the hard slicing method and the standard DQN-based scheme. Hard slicing means that each service is always allocated with $\frac{1}{3}$ of the whole bandwidth (because there are three types of services in total); round-robin scheduling is conducted within each slice. The DQN-based bandwidth allocation scheme was first proposed in [13], which directly applied the original DQN algorithm [23] to the network slicing scenario. Notably, our previous works in [13] have demonstrated the classical DQN-driven method is superior to other machine learning methods (e.g., long short-term memory (LSTM)-based prediction-before-allocation method). Therefore, due to the space limitation, we put more emphasis on the performance comparison with the classical DQN in [13].

1) Small Packets for URLLC Service: We first observe the performance of the proposed algorithms within the scenario that the packet size of URLLC service is small. The traffic parameters are shown in Table II. We consider two cases: the bandwidth allocation resolution is either 1 MHz or 200 KHz. The importance weights in the optimization objective (i.e., Eq (18)) are set to $\alpha = 0.01, \boldsymbol{\beta} = [1, 1, 1]$. The values of the clipping parameters c_1 and c_2 are determined heuristically.[4] In both cases, we set $c_1 = 6.5, c_2 = 4.5$ to clip the system utility according to Eq. (24), where η is fixed at 1. The experimental evaluation of the reward-clipping setting is investigated hereinafter. Fig. 4 depicts the variations of the

[4]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 suitable values that guarantee both performance and stability.

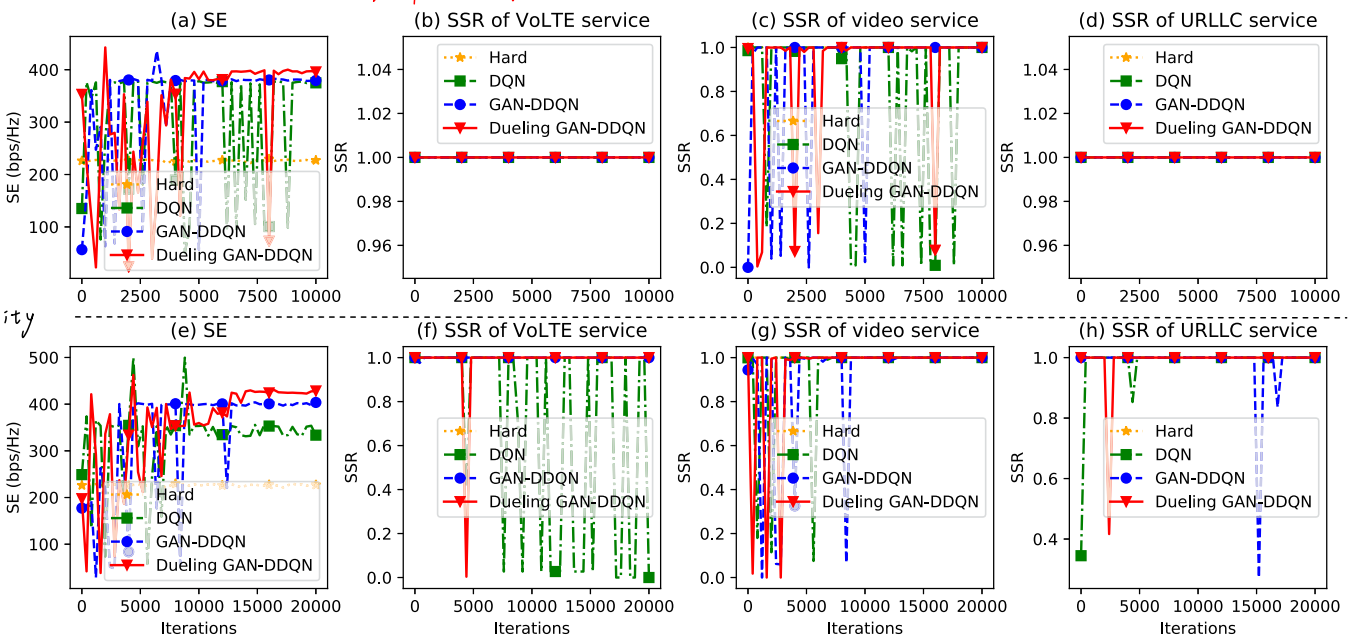


Fig. 5. An illustration of SE and SSR in the different cases where the bandwidth allocation resolution is 1 MHz (shown in the top sub-figures) and 200 KHz (shown in the bottom sub-figures).

system utility with respect to the iteration index. The left part of Fig. 4 shows that when the bandwidth allocation resolution is 1 MHz, the three RL-based algorithms perform similarly, but Dueling GAN-DDQN is slightly better; DQN is the most erratic in training. The right part of Fig. 4 illustrates that when the bandwidth allocation resolution becomes 200 KHz, GAN-DDQN and Dueling GAN-DDQN expand the gap to DQN, which demonstrates the performance improvement coming from distributional RL by the characterization of the action-value or state-value distributions. It's worth noting that Dueling GAN-DDQN improves visibly over GAN-DDQN in both performance and stability, consistent with our previous discussion. Furthermore, it can be observed in Fig. 4 that the system utility obtained by the three RL-based algorithms is significantly greater than that for the hard slicing scheme. The reason for this lies in that the RL-based algorithms can dynamically and reasonably manage bandwidth resources, thereby avoiding wasted resources and improving resources utilization. Moreover, the utilization of the bandwidth resource further gets improved when we slice the bandwidth more finely. Fig. 4 shows that the system utility obtained by the two GAN-based algorithms, especially Dueling GAN-DDQN, becomes significantly larger when the bandwidth allocation resolution changes to 200 KHz from 1 MHz, but that the performance of DQN is slightly degraded.

Fig. 5 presents the variations of SE and SSR with respect to the iteration index for both bandwidth allocation resolution settings (i.e., 1 MHz and 200 KHz). It can be observed from Fig. 5 that SE curves are basically consistent with the system utility curves. However, the SSR curves of the three algorithms for the NSs show different patterns. When the bandwidth allocation resolution is 1 MHz, the SSRs for both VoLTE and URLLC services reach 100% with iterative training. Nevertheless, for the SSR of video service, GAN-DDQN and Dueling GAN-DDQN basically converge to 100% after 5000 iterations, while DQN shows no obvious signs of convergence. When the bandwidth allocation resolution is 200 KHz, it can be observed that GAN-DDQN and Dueling GAN-DDQN, by and large, realize 100% of SSR for all three services by the end of training, but DQN shows extreme instability for VoLTE service. Note that the unusual sudden performance drop late in training is caused by the tiny nonzero exploration rate in the ϵ-greedy exploration strategy.

In Fig. 6, we illustrate the policy learned by the three algorithms when the bandwidth allocation resolution is 200 KHz. It can be observed that all three algorithms converge after 15000 iterations. However, there are some differences between the learned bandwidth allocation policies. The DQN agent allocates the least bandwidth to VoLTE service and keeps it unchanged; as a result, SSR of VoLTE service does not always reach 100%. Between the GAN-DDQN agent and the Dueling GAN-DDQN agent, the latter behaves more intelligently, which is prominently manifested in the fact that it maximizes the bandwidth allocated to video service and reduces the bandwidth allocated to the other two services while meeting the SLA. The Dueling GAN-DDQN agent provides as much bandwidth as possible to satisfy the SLA of video service which is requested frequently and bandwidth-consuming, thus improving the SE. Besides, Dueling GAN-DDQN agent provides a policy to better balance the demands of VoLTE and URLLC services that are relatively rarely requested.

We next investigate the impact of the reward-clipping mechanism. Fig. 7 shows the differences in system utility during the iterative learning of GAN-DDQN with and without the reward clipping when the bandwidth allocation resolution is 1 MHz. When there is no reward clipping, GAN-DDQN directly takes

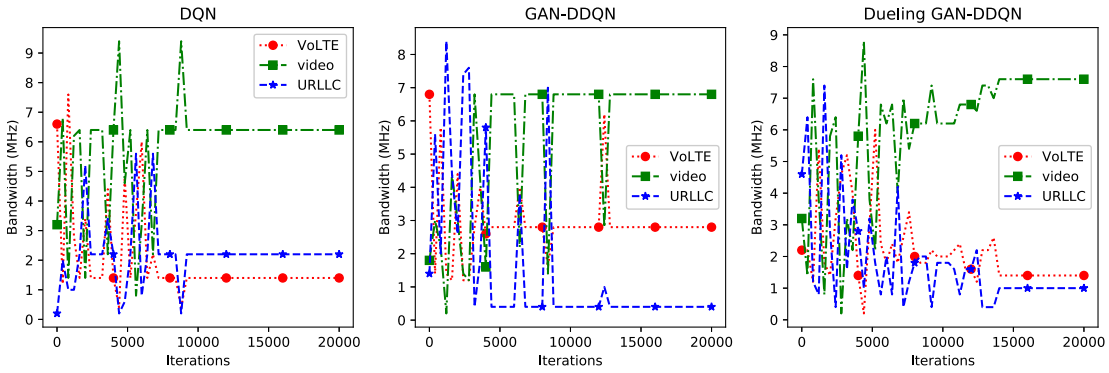


Fig. 6. An illustration of bandwidth allocation schemes in the case where the bandwidth allocation resolution is 200 KHz.

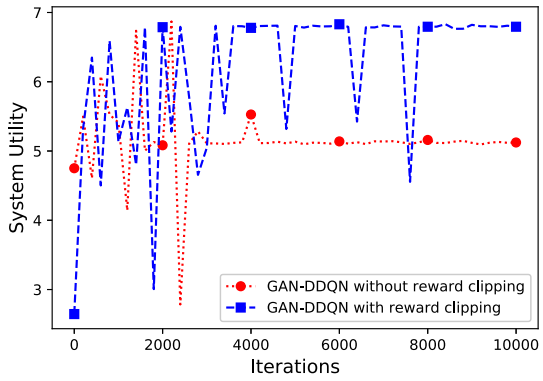


Fig. 7. Comparison of GAN-DDQN with and without reward clipping.

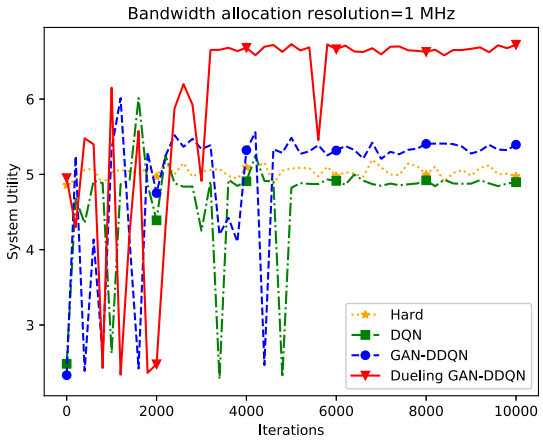


Fig. 8. An illustration of performance comparison between different slicing schemes in the case where the packets of URLLC service are large and the bandwidth allocation resolution is 1 MHz.

the system utility as the reward. Note that the values of system utility, although they fluctuate, are much larger than the clipping constants set manually in the reward-clipping mechanism. As a result, if the system utility is directly used as the reward, the target action-value distribution might vary significantly, making it difficult for GAN-DDQN to converge. Therefore, GAN-DDQN without reward clipping requires more training steps to converge from one equilibrium to a new one. It can be observed from Fig. 7 that the GAN-DDQN performs significantly better with reward clipping than without. The simulation results verify the effectiveness of GAN-DDQN together with reward clipping.

2) Large Packets for URLLC Service: In this part, we consider the case where the packet size of URLLC service is evenly sampled from $\{0.3, 0.4, 0.5, 0.6, 0.7\}$ MByte, which gives a considerably larger packet size than we just analyzed and requires more bandwidth to guarantee meeting the SLA of URLLC service. In the case where bandwidth allocation resolution is 1 MHz, we set $c_1 = 5.7, c_2 = 3$ to clip the system utility according to Eq. (24), where η is fixed to 1. Fig. 8 shows the performance of each slice algorithm, from which it can be observed that Dueling GAN-DDQN is way ahead of the others in terms of system utility. However, quite unexpectedly, DQN performs poorly, worse even than hard slicing scheme. Fig. 9 reveals the details of SE and SSR, from which we can find that Dueling GAN-DDQN agent learned a policy that maximizes system utility by sacrificing the SSR of URLLC service in exchange for higher SE. The reason for this is that when SSR is equally important to all three services (i.e., $\boldsymbol{\beta} = [1, 1, 1]$), it is challenging for URLLC service to satisfy its SLA given the large transmission volume and the strictly low latency requirement. Therefore, we further investigate the situation in

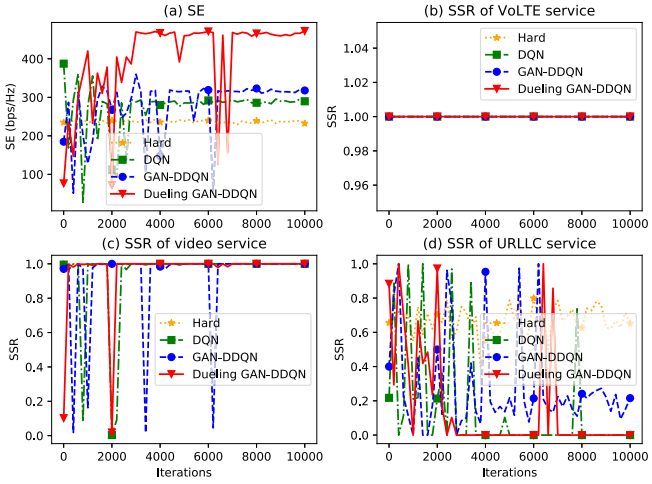


Fig. 9. An illustration of SE and SSR in the case where the bandwidth allocation resolution is 1 MHz.

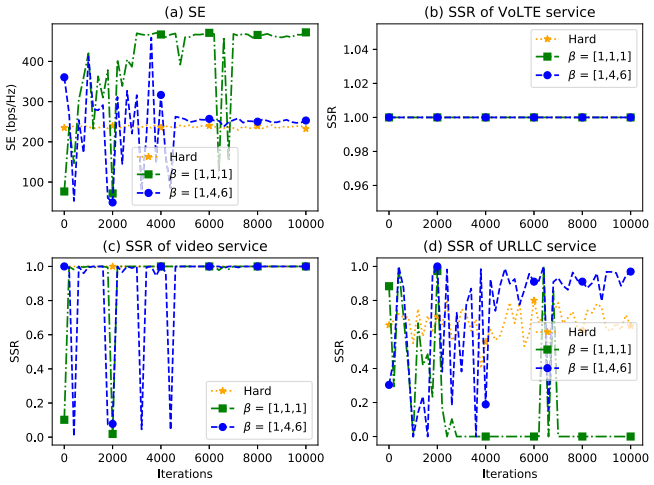


Fig. 10. An illustration of SE and SSR achieved by Dueling GAN-DDQN when the bandwidth allocation resolution is 1 MHz and the importance weights of SSR are $[1, 1, 1]$ and $[1, 4, 6]$, respectively.

which URLLC service is more concerned while keeping video service dominating VoLTE service, which is reflected in the value of $\boldsymbol{\beta}$ changing to $[1, 4, 6]$ from $[1, 1, 1]$. Fig. 10 presents the results for these different $\boldsymbol{\beta}$ settings and demonstrates that the adjusted importance weight makes the SLA of URLLC service well guaranteed; meanwhile, the SLA of the other two services can be 100% satisfied as well. However, the requests for URLLC service are scarce in a timeslot—in our simulation, it is the interval between two updates of the agent, defined as 1 second—while the agent has to allocate more bandwidth to URLLC slice to guarantee conformity to SLA until the next update, which wastes the bandwidth to some extent and thus leads to the decrease of SE. There are two lessons that we learn from these simulation results: (a) it is non-trivial to optimize multiple conflicting objectives, even when using cutting-edge RL algorithms; (b) shortening the interval between successive bandwidth allocations may improve performance but it also increases computational costs and raises issues of stability due to more drastic changes in demand.

V. Conclusion

In this paper, we have investigated the combination of deep distributional RL and GAN and proposed GAN-DDQN to learn the optimum solution for demand-aware resource management in network slicing. In particular, we have applied GAN to approximate the action-value distribution, so as to avoid the negative impact of randomness and noise on the reward and grasp much more details therein than the conventional DQN. We have also designed a new update procedure that combines the advantages offered by distributional RL with the training algorithm of WGAN-GP. Furthermore, we have adopted the reward-clipping scheme to enhance the training stability of GAN-DDQN. Besides, we have introduced the dueling structure to the generator (i.e., Dueling GAN-DDQN), so as to separate the state-value distribution and the action advantage function from the action-value distribution and thus avoid the inherent training problem of GAN-DDQN.

Extensive simulations have demonstrated the effectiveness of GAN-DDQN and Dueling GAN-DDQN with superior performance over the classical DQN algorithm. In the future, we will try to further improve the GAN-DDQN mechanism under various scenarios with multiple-metric constraints as well as non-stationary traffic demands.

Appendix
The Proof of Theorem 1

Before the proof of Theorem 1, we give the following lemmas:

Lemma 1: Because $v(\theta, \psi) = 0$ *if and only if* $(\theta, \psi) = (\xi, 0)$*, the unique Nash-equilibrium point of the training objective in Eq.* (30) *is given by* $\theta = \xi, \psi = 0$.

Lemma 2: The distance between the optimal boundaries of Dirac-WGAN-GP on $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *is* δ.

Proof: Dirac-WGAN-GP consists of a generator with parameter θ and a linear discriminator $D_\psi(x) = \psi \cdot x$, where the generator outputs a Dirac distribution centralized θ (i.e., δ_θ). Whilst the real data distribution is given by a Dirac distribution concentrated at ξ (i.e., δ_ξ). Therefore, the training objective of Dirac-WGAN-GP is given by

$$L(\theta, \psi) = \psi\theta - \xi\psi \tag{30}$$

and the gradient penalty proposed in [42] is given by

$$\begin{aligned} p(\psi) &= \frac{\lambda}{2}\mathbb{E}_{\hat{x}}\left(\|\nabla_{\hat{x}} D_\psi(\hat{x})\| - 1\right)^2 \\ &= \frac{\lambda}{2}\left(|\psi| - 1\right)^2 \end{aligned} \tag{31}$$

Inspired by [46], we use gradient vector field to analyze convergence, which is defined as follow

$$v(\theta, \psi) := \begin{pmatrix} -\nabla_\theta L(\theta, \psi) \\ \nabla_\psi L(\theta, \psi) \end{pmatrix} \tag{32}$$

For Dirac-WGAN-GP, the corresponding gradient vector field is given by

$$v(\theta, \psi) = \begin{pmatrix} -\psi \\ \theta - \xi + \text{sign}(\psi)\lambda\left(|\psi| - 1\right) \end{pmatrix} \tag{33}$$

where $\text{sign}(\cdot)$ denotes the signum function and we have Lemma 1.

Assume that the iteration (θ_k, ψ_k) converges towards the equilibrium point $(\xi, 0)$ but $(\theta_k, \psi_k) \neq (\xi, 0)$ for all $k \in \mathbb{N}$, which implies that $v(\theta_k, \psi_k) \approx 0$ and thus we have

$$-\psi_k \approx \theta_k - \xi + \text{sign}(\psi_k)\lambda\left(|\psi_k| - 1\right) \tag{34}$$

in other words,

$$\theta_k \approx -\psi_k + \xi - \text{sign}(\psi_k)\lambda(|\psi_k| - 1) \tag{35}$$

Then, we can get the update amount of parameter θ after the $(k+1)$th training as follow

$$\begin{aligned} &|\theta_{k+1} - \theta_k| \\ &\quad\approx h\left|-\psi_k + \xi - \text{sign}\left(\psi_k\right)\lambda(|\psi_k| - 1) - \theta_k\right| \\ &\quad\approx h\left|-(\lambda+1)\psi_k + (\xi - \theta_k) + \text{sign}\left(\psi_k\right)\lambda\right| \end{aligned} \tag{36}$$

Therefore, we have $\lim_{k\to\infty}|\theta_{k+1} - \theta_k| = h\lambda$, which shows that Dirac-WGAN-GP cannot converge to the equilibrium

point, and the value of generator's parameter will finally oscillate between $\xi - \frac{h\lambda}{2}$ and $\xi + \frac{h\lambda}{2}$.

Assume that $\mathcal{D}_1$ and $\mathcal{D}_2$ are two different real data, which are the Dirac distributions concentrated at ξ_1 and ξ_2, respectively. Let $\delta = |\xi_1 - \xi_2|$, which indicates the statistic distance between $\mathcal{D}_1$ and $\mathcal{D}_2$. Note that usually $h\lambda$ is two or three orders of magnitude smaller than δ, which implies that the optimal boundaries is rarely overlapped, thus further training is required when the real data varies. With the constant learning rate, it is easy to deduce from Lemma 2 that the larger the δ is, the more training steps are required for the Dirac-WGAN-GP to reach the new optimal boundary. Finally, based on Lemma 1 and 2, we obtain the proof of Theorem 1. ■

REFERENCES

[1] Y. Hua, R. Li, Z. Zhao, H. Zhang, and X. Chen, "GAN-based deep distributional reinforcement learning for resource management in network slicing," in *Proc. Globecom*, Waikoloa, HI, USA, Dec. 2019.

[2] 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, Aug. 2017.

[3] R. Li *et al.*, "Intelligent 5G: When cellular networks meet artificial intelligence," *IEEE Wireless Commun.*, vol. 24, no. 5, pp. 175–183, Oct. 2017.

[4] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 94–100, May 2017.

[5] *Minimum Requirement Related to Technical Performance for IMT-2020 Radio Interface (s)*, document ITU-R M.2410-0, Nov. 2017.

[6] X. Zhou, R. Li, T. Chen, and H. Zhang, "Network slicing as a service: Enabling enterprises' own software-defined cellular networks," *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 146–153, Jul. 2016.

[7] X. Li *et al.*, "Network slicing for 5G: Challenges and opportunities," *IEEE Internet Comput.*, vol. 21, no. 5, pp. 20–27, Sep. 2017.

[8] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2429–2453, 3rd Quart., 2018.

[9] H. Zhang, N. Liu, X. Chu, K. Long, A. Aghvami, and V. C. M. Leung, "Network slicing based 5G and future mobile networks: Mobility, resource management, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 138–145, Aug. 2017.

[10] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Network slicing for 5G with SDN/NFV: Concepts, architectures, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 80–87, May 2017.

[11] I. da Silva *et al.*, "Impact of network slicing on 5G radio access networks," in *Proc. EuCNC*, Athens, Greece, Jun. 2016, pp. 153–157.

[12] *Study on New Services and Markets Technology Enables, Release 14*, document 3GPP TR 22.981, Mar. 2016.

[13] R. Li *et al.*, "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74429–74441, 2018.

[14] S. Vassilaras *et al.*, "The algorithmic aspects of network slicing," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 112–119, Aug. 2017.

[15] B. Han, J. Lianghai, and H. D. Schotten, "Slice as an evolutionary service: Genetic optimization for inter-slice resource management in 5G networks," *IEEE Access*, vol. 6, pp. 33137–33147, 2018.

[16] P. L. Vo, M. N. H. Nguyen, T. A. Le, and N. H. Tran, "Slicing the edge: Resource allocation for RAN network slicing," *IEEE Wireless Commun. Lett.*, vol. 7, no. 6, pp. 970–973, Dec. 2018.

[17] Y. Sun, G. Feng, L. Zhang, M. Yan, S. Qin, and M. A. Imran, "User access control and bandwidth allocation for slice-based 5G-and-beyond radio access networks," in *Proc. ICC*, Shanghai, China, May 2019, pp. 1–6.

[18] M. Jiang, M. Condoluci, and T. Mahmoodi, "Network slicing management & prioritization in 5G mobile systems," in *Proc. Eur. Wireless Conf.*, Oulu, Finland, May 2016, pp. 1–6.

[19] S. D'Oro, F. Restuccia, T. Melodia, and S. Palazzo, "Low-complexity distributed radio access network slicing: Algorithms and experimental results," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2815–2828, Dec. 2018.

[20] Z. Zhou, L. Tan, B. Gu, Y. Zhang, and J. Wu, "Bandwidth slicing in software-defined 5G: A Stackelberg game approach," *IEEE Veh. Technol. Mag.*, vol. 13, no. 2, pp. 102–109, Jun. 2018.

[21] G. A. Rummery and M. Niranjan, *On-Line Q-Learning Using Connectionist Systems*, vol. 37. Cambridge, U.K.: Univ. of Cambridge, 1994.

[22] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn*, vol. 9, nos. 3–4, pp. 279–292, May 1992.

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

[24] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proc. ICML*, Sydney, NSW, Australia, Aug. 2017, pp. 449–458.

[25] R. Koenker and K. Hallock, "Quantile regression," *J. Econ. Perspect.*, vol. 15, no. 4, pp. 143–156, 2001.

[26] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proc. AAAI*, New Orleans, LA, USA, Feb. 2018, pp. 2892–2901.

[27] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," Jun. 2018, *arXiv:1806.06923*. [Online]. Available: https://arxiv.org/pdf/1806.06923.pdf

[28] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. NeurIPS*, Montreal, QC, Canada, Dec. 2014, pp. 2672–2680.

[29] X. Li, J. Fang, W. Cheng, H. Duan, Z. Chen, and H. Li, "Intelligent power control for spectrum sharing in cognitive radios: A deep reinforcement learning approach," *IEEE Access*, vol. 6, pp. 25463–25473, 2018.

[30] 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. ICC*, Paris, France, May 2017, pp. 1–6.

[31] N. Liu *et al.*, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *Proc. ICDCS*, Atlanta, GA, USA, vol. Jun. 2017, pp. 372–382.

[32] Y. He, F. R. Yu, N. Zhao, V. C. M. Leung, and H. Yin, "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach," *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 31–37, Dec. 2017.

[33] T. Doan, B. Mazoure, and C. Lyle, "GAN Q-learning," May 2018, *arXiv:1805.04874*. [Online]. Available: https://arxiv.org/pdf/1805.04874.pdf

[34] D. Freirich, T. Shimkin, R. Meir, and A. Tamar, "Distributional multivariate policy evaluation and exploration with the Bellman GAN," *Proc. ICML*, Long Beach, CA, USA, Jun. 2019.

[35] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," Nov. 2015. *arXiv:1511.06581*. [Online]. Available: https://arxiv.org/pdf/1511.06581.pdf

[36] C. Qi, Y. Hua, R. Li, Z. Zhao, and H. Zhang, "Deep reinforcement learning with discrete normalized advantage functions for resource management in network slicing," *IEEE Commun. Lett.*, vol. 23, no. 8, pp. 1337–1341, Aug. 2019.

[37] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[38] V. Mnih *et al.*, "Playing Atari with deep reinforcement learning," Dec. 2013, *arXiv:1312.5602*. [Online]. Available: https://arxiv.org/pdf/1312.5602.pdf

[39] J. N. Tsitsiklis and B. Van Roy, "Feature-based methods for large scale dynamic programming," *Mach. Learn.*, vol. 22, no. 1, pp. 59–94, 1996.

[40] G. Barth-Maron *et al.*, "Distributed distributional deterministic policy gradients," Apr. 2018, *arXiv:1804.08617*. [Online]. Available: https://arxiv.org/pdf/1804.08617.pdf

[41] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," Jan. 2017, *arXiv:1701.07875*. [Online]. Available: https://arxiv.org/pdf/1701.07875.pdf

[42] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of Wasserstein GANs," in *Proc. NeurIPS*, Long Beach, CA, USA, Dec. 2017, pp. 5767–5777.

[43] S. Costanzo, I. Fajjari, N. Aitsaadi, and R. Langar, "A network slicing prototype for a flexible cloud radio access network," in *Proc. CCNC*, Las Vegas, NV, USA, Jan. 2018, pp. 1–4.

[44] S. Costanzo, I. Fajjari, N. Aitsaadi, and R. Langar, "DEMO: SDN-based network slicing in C-RAN," in *Proc. CCNC*, Las Vegas, NV, USA, Jan. 2018, pp. 1–2.

[45] *Network Slicing Architecture*. Accessed: Oct. 13, 2019. [Online]. Available: https://tools.ietf.org/id/draft-geng-netslices-architecture-02.html#rfc.references.2

[46] L. Mescheder, A. Geiger, and S. Nowozin, "Which training methods for GANs do actually converge?" Jan. 2018, *arXiv:1801.04406*. [Online]. Available: https://arxiv.org/pdf/1801.04406.pdf
[47] *Evolved Universal Terrestrial Radio Access (E-UTRA); Further Advancements for E-UTRA Physical Layer Aspects, Release 9*, document 3GPP TR 36.814, Mar. 2010.
[48] *Service Requirements for Next Generation New Services Markets, Release 15*, document 3GPP TS 22.261, Mar. 2017.

Yuxiu Hua received the B.S. degree in electronic information engineering from Hangzhou Dianzi University, Hangzhou, China, in June 2016. He is currently pursuing the Ph.D. degree with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou. His research interest includes deep learning, reinforcement learning, and network slicing.

Rongpeng Li (S'12–M'17) received the B.E. degree (Hons.) from Xidian University, Xi'an, China, in June 2010, and the Ph.D. degree (Hons.) from Zhejiang University, Hangzhou, China, in June 2015. He was a Research Engineer with the Wireless Communication Laboratory, Huawei Technologies Company, Ltd., Shanghai, China, from August 2015 to September 2016. In November 2016, he returned to academia as a Post-Doctoral Researcher with the College of Computer Science and Technologies, Zhejiang University, which is sponsored by the National Post-Doctoral Program for Innovative Talents. He is currently an Assistant Professor with the College of Information Science and Electronic Engineering, Zhejiang University. His research interests currently focus on reinforcement learning, data mining, and all broad-sense network problems (e.g., resource management and security). He has authored or coauthored several articles in the related fields. He serves as an Editor of *China Communications*.

Zhifeng Zhao received the bachelor's degree in computer science, the master's degree in communication and information system, and the Ph.D. degree in communication and information system from the PLA University of Science and Technology, Nanjing, China, in 1996, 1999, and 2002, respectively. From 2002 to 2004, he acted as a Post-Doctoral Researcher with the Zhejiang University, Hangzhou, China, where his researches were focused on multimedia next-generation networks (NGN) and soft-switch technology for energy efficiency. From 2005 to 2006, he also acted as a Senior Researcher with the PLA University of Science and Technology, Nanjing, where he performed research and development on advanced energy-efficient wireless router, ad hoc network simulator, and cognitive mesh networking test-bed. From 2006 to 2019, he was an Associate Professor with the College of Information Science and Electronic Engineering, Zhejiang University. He is currently with Zhejiang Lab, Hangzhou, China. His research area includes cognitive radio, wireless multi-hop networks (ad hoc, Mesh, and WSN), wireless multimedia networks, and green communications. He is the Symposium Co-Chair of the ChinaCom 2009 and 2010. He is also the Technical Program Committee (TPC) Co-Chair of the IEEE ISCIT 2010 (10th IEEE International Symposium on Communication and Information Technology).

Xianfu Chen received the Ph.D. degree (Hons.) in signal and information processing from the Department of Information Science and Electronic Engineering (ISEE), Zhejiang University, Hangzhou, China, in March 2012. Since April 2012, he has been with the VTT Technical Research Centre of Finland, Oulu, Finland, where he is currently a Senior Scientist. His research interests cover various aspects of wireless communications and networking, with emphasis on human-level and artificial intelligence for resource awareness in next-generation communication networks. He is serving and served as the Track Co-Chair and a TPC member for a number of IEEE ComSoc flagship conferences. He is also a Vice Chair of the IEEE Special Interest Group on Big Data with Computational Intelligence, a member of which come from more than 15 countries worldwide.

Honggang Zhang was an Honorary Visiting Professor with the University of York, U.K., and an International Chair Professor of Excellence for the Université Européenne de Bretagne and Supèlec, France. He is currently a Full Professor with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China. He has coauthored and edited two books: *Cognitive Communications: Distributed Artificial Intelligence (DAI), Regulatory Policy and Economics, Implementation* (John Wiley & Sons) and *Green Communications: Theoretical Fundamentals, Algorithms and Applications* (CRC Press), respectively. He is also active in the research on cognitive green communications. He was the leading Guest Editor of the *IEEE Communications Magazine* special issues on Green Communications. He served as the Chair of the Technical Committee on Cognitive Networks of the IEEE Communications Society from 2011 to 2012 and the Series Editor for the *IEEE Communications Magazine* for its Green Communications and Computing Networks Series from 2015 to 2018. He is an Associate Editor-in-Chief of *China Communications*.