IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 17, 2019, accepted March 9, 2019, date of publication March 18, 2019, date of current version April 8, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2905589

A Reinforcement Learning Based Smart Cache Strategy for Cache-Aided Ultra-Dense Network

WEI LI, JUN WANG, (Member, IEEE), GUOYONG ZHANG, LI LI, ZE DANG, AND SHAOQIAN LI, (Fellow, IEEE)

National Key Laboratory of Science and Technology, University of Electronic Science and Technology of China, Chengdu 610054, China

Corresponding author: Jun Wang (junwang@uestc.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFC0807101, in part by the National Defense Pre-Research Foundation of China under Grant 6141B062901, in part by the National Research Program of China under Grant 9020302, in part by the Foundation of National Key Laboratory of Science and Technology on Communications, in part by the Innovation Fund of NCL (IFN) under Grant IFN2018214, and in part by the National Natural Science Foundation of China (NSFC) under Grant 61471099.

ABSTRACT The integration of caching and ultra-dense network (UDN) can not only improve the efficiency of content retrieval by reducing duplicate content transmissions but also improve the network throughput and system energy efficiency (EE) of the UDN. In this paper, we focus on energy consumption aspects of cache-aided UDN (CUDN) and develop a novel caching strategy to improve the system EE. Different from the existing researches, we consider a more realistic scenario where the popularity of the cache content is dynamic and unknown. The proposed caching strategy is a deep reinforcement learning (DRL)-based approach that uses a deep Q-network to approximate the Q action-value function. We optimize the structure and corresponding parameters of the deep Q-network according to the latest findings in the field of DRL and deep learning (DL) to improve the performance of this caching strategy. The simulation results show that the performance in terms of EE of the CUDN can be significantly improved by using our proposed caching strategy.

INDEX TERMS Ultra-dense network, cache, energy efficiency, deep reinforcement learning.

I. INTRODUCTION

The ultra-dense network (UDN) [1] is considered as one of the key techniques of the fifth generation (5G) mobile communication network to achieve high system throughput and energy efficiency (EE). However, for the ever-increasing applications which need not only high data rate but also high caching and computing capabilities, the existing UDN cannot work well [2]. A promising approach to address this issue is to integrate UDN and cache into one system [3]. By incorporating cache functionality into the UDN, the resulting system, i.e., the cache-aided UDN (CUDN), can provide native support for highly scalable and efficient content retrieval, and meanwhile, the duplicate content transmissions within the network can be significantly reduced. As a consequence, the performance of the CUDN, such as mobility, flexibility, throughput and system EE, can be considerably improved.

The associate editor coordinating the review of this manuscript and approving it for publication was Richard Yu.

For the CUDN, in order to improve the accessibility of data content to users, the most popular content should be stored in the local caches to maximize the content hit probability. Hence, the content placement strategy, i.e., which files are chosen to store in the local caches, is very crucial. To address the issue of content placement strategy, some schemes have been proposed based on the assumption that the popularity of the content is fixed and perfectly known [4]–[7]. In such circumstances, the cache content placement strategy can be formulated as a deterministic optimization problem, and the optimal content placement strategy can be obtained based on some advanced optimization methods. However, as mentioned in [9] and [10], such assumptions cannot be reasonably justified. Recently, a more practical case of cache content placement is investigated without prior information of the content popularity. Bharath *et al.* [9] assume that the popularity of the content is fixed and unknown. They estimate the popularity of the content according to the instantaneous demands of users within a specified time interval and obtain an estimation of the cost function. Then, a transfer

learning-based method is proposed to improve the estimation. In [10], by using a multi-armed bandit algorithm, the small base station (SBS) learns the fixed popularity profile online by refreshing its cache content and observing the instantaneous demands for the cached files over time. In this way, the cache content placement strategy for the SBS is optimized over time, and the users can be served by the SBS with maximal data rate.

However, for all the aforementioned approaches, neither the potential diversity nor the time-sensitive characteristics of different content popularity has been considered. In fact, Tatar *et al.* [11] show that the popularity of cache content is highly non-stationary over time. Therefore, the above mentioned estimation methods of the cache popularity will inevitability introduce deviation, and the corresponding cache placement strategy cannot achieve optimal performance in this case.

To cope with the dynamic of the content popularity, deep reinforcement learning (DRL) based approaches [12]–[14] have been proposed recently. He *et al.* [12] consider to obtain the optimal interference alignment (IA) user selection policy in cache-enabled opportunistic IA wireless networks. Somuyiwa *et al.* [13] consider the issue that a mobile user accesses contents in a dynamic environment, and models the proactive caching problem as a Markov decision process aiming at minimizing the long term average energy cost. Somuyiwa *et al.* [13] use reinforcement learning (RL) algorithms to find near optimal cache placement parameters. In [14], the local and global Markov process models of user request and RL framework are proposed for cache-aided heterogeneous networks. Sadeghi *et al.* [14] propose a novel RL based caching by utilizing the Q-learning algorithm to implement the optimal cache placement policy.

This paper focuses on the energy efficiency (EE) aspects of the CUDN and considers a realistic scenario, where the popularity of the contents is dynamic and unknown. We propose a novel online learning algorithm for content placement based on DRL by using a deep Q neural network to approximate the Q action-value function under our considered CUDN scenarios. The main differences between our work and the existing DRL-based approaches [12]–[14] are as follows.

- First, we consider a new scenario, i.e., CUDN, and put our effort on maximizing the EE.
- Second, we optimize the parameters and structure of the adopted deep Q neural network according to the latest findings of Google DeepMind [15]–[18], such as prioritized experience replay, dueling and recurrent neural network (RNN), and obtain a novel and near-optimal cache strategy for our considered CUDN scenarios.
- Third, we use Google TensorFlow [20] to implement our proposed cache strategy.

Simulation results show that the performance in terms of EE of the CUDN can be significantly improved by using our proposed caching strategy.

The rest of this paper is organized as follows. The system model and the problem formulation are described in Section II. In Section III, the DRL based content cache algorithm is proposed for our considered CUDN scenario, and the parameters and structure of the adopted neural network are also optimized. Simulation results are provided in Section IV, followed by concluding remarks in Section V.

II. SYSTEM MODEL AND PROBLEM FORMULATION

A. NETWORK MODEL

The general architecture of CUDN is shown in Fig.1. We consider the downlink of a CUDN, where the small access points (SAP) with caching capability are overlaid with macro base stations (MBS). We assume that universal frequency reuse (UFR) is adopted, where the BSs in different tiers share the whole spectrum of the system [21], to fully utilize the network spectrum. The SAPs, MBSs and users are distributed according to homogeneous spatial Poisson point process (PPP) with density λ_s, λ_m and λ_u, respectively, where $\lambda_s > \lambda_m$. The location distribution of SAPs, MBSs and users is denoted as $\Phi_s = \{r_i^s\}_{i\in\mathbb{N}}$, $\Phi_m = \{r_j^m\}_{j\in\mathbb{N}}$ and $\Phi_u = \{r_i^u\}_{i\in\mathbb{N}}$, respectively, where $r_i^s, r_i^m, r_i^u \in \mathbb{R}^2$, and $\mathbb{R}^2$ represents the feasible two-dimensional space. We use the notations $\mathcal{U}, \mathcal{S}$ and $\mathcal{M}$ to denote the set of users, SAPs and MBSs, respectively. We consider the single input single output (SISO) system, for which both large-scale fading and small-scale fading are considered. Specifically, due to large-scale fading, the transmitted signal is attenuated by $g(d) = bd^{-\hat{\alpha}}$, where d is the distance, b is the path loss coefficient, and $\hat{\alpha} > 2$ is the path loss exponent. For small-scale fading, Rayleigh fading is assumed, i.e., channel fading coefficient h satisfies $h \sim \mathcal{CN}(0, 1)$. For convenience, the important notations used in this paper are summarized in Table 1.

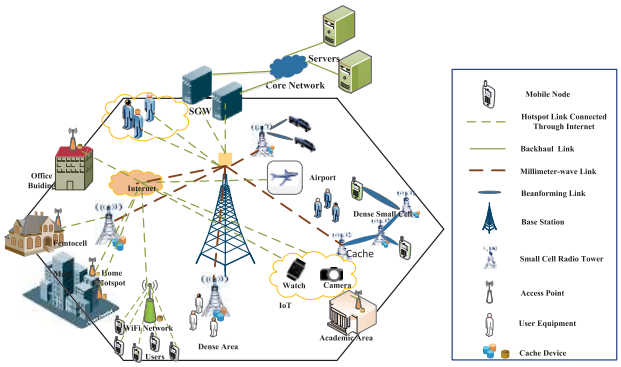


FIGURE 1. General architecture of cache-aided ultra-dense network [4].

At a certain service timeslot, each user can be served by any of the nearest SAP or the nearest MBS as long as the corresponding service condition is satisfied. Therefore, the signal strength received by a typical user i can be expressed as

$$p(r_i^u) = I_i^{\text{idx}} \left|h_{j,i}^s\right|^2 P_{j,i}^s b |r_i^u - r_j^s|^{-\hat{\alpha}} + (1 - I_i^{\text{idx}}) \left|h_{l,i}^m\right|^2 P_{l,i}^m b |r_i^u - r_l^m|^{-\hat{\alpha}}, \quad (1)$$

TABLE 1. Table of notations.

Symbols	Definitions
$\lambda_s, \lambda_m, \lambda_u$	density of SAPs, MBSs and users
Φ_s, Φ_m, Φ_u	distribution of SAPs, MBSs and users
$\mathcal{S}, \mathcal{M}, \mathcal{U}$	set of SAPs, MBSs and users
d	distance between a user and the service AP
b	path loss coefficient
$\hat{\alpha}$	path loss exponent
h	rayleigh fading coefficient
I_i^{idx}	association index of a typical user
SINR	signal to interference plus noise ratio
σ^2	power of Gaussian noise
P^s, P^m	transmit power of SAPs and MBSs
$\mathcal{F}$	set of content file library
N_s, N_m	cache size of SAPs and MBSs
D_s, D_m	number of combinations for SAPs and MBSs
P_o^s, P_o^m	operational charge power for SAPs and MBSs
P_{lr}	data retrieval power of local cache
P_{br}	data retrieval power of backhaul

where $I_i^{\text{idx}} = \{0, 1\}$ is the association index of the typical user. If $I^{\text{idx}_i} = 1$, the typical user is served by a SAP, otherwise, $I_i^{\text{idx}} = 0$ represents that the typical user is served by a MBS. $h_{j,i}^s$ and $h_{l,i}^m$ denote the Rayleigh fading between the typical user i and the service SAP j or service MBS l, respectively. $P_{j,i}^s$ and $P_{l,i}^m$ stand for the transmit power of the service SAP j or MBS l for the typical user i, respectively. As a result, the received signal to interference and noise ratio (SINR) of this typical user is given by

$$\text{SINR}(r_i^u) = \frac{p(r_i^u)}{\sigma^2 + \mathcal{I}_i^u}, \tag{2}$$

where σ^2 is the power of Gaussian noise, and

$$\begin{aligned}\mathcal{I}_i^u = I_i^{\text{idx}} \Bigg(& \sum_{r_{\hat{j}}^s \in \Phi_s \setminus r_j^s} \left|h_{\hat{j},i}^s\right|^2 P_{\hat{j}}^s b |r_i^u - r_{\hat{j}}^s|^{-\hat{\alpha}} \\ & + \sum_{r_{\hat{l}}^m \in \Phi_m} \left|h_{\hat{l},i}^m\right|^2 P_{\hat{l}}^m b |r_i^u - r_{\hat{l}}^m|^{-\hat{\alpha}} \Bigg) \\ & + (1 - I_i^{\text{idx}}) \Bigg(\sum_{r_{\hat{j}}^s \in \Phi_s} \left|h_{\hat{j},i}^s\right|^2 P_{\hat{j}}^s b |r_i^u - r_{\hat{j}}^s|^{-\hat{\alpha}} \\ & + \sum_{r_{\hat{l}}^m \in \Phi_m \setminus r_l^m} \left|h_{\hat{l},i}^m\right|^2 P_{\hat{l}}^m b |r_i^u - r_{\hat{l}}^m|^{-\hat{\alpha}} \Bigg), \end{aligned} \tag{3}$$

is the cumulative interference experienced from all the SAPs and MBSs except the service base station.

B. CACHE MODEL

We assume that each user independently requests data files from a library $\mathcal{F} = \{1, \cdots, F\}$ containing F files with equal size of L bits. We denote the content request density as λ_r per time period. The popularity of the contents is time-sensitive and unknown. Each SAP and MBS is equipped with a cache size of N_s and N_m, respectively. Here, $N_s < F, N_m < F$. Therefore, the cached files form a combination of all the files in the library. There are $D_s \triangleq \binom{F}{N_s}$ different combinations for SAPs, and $D_m \triangleq \binom{F}{N_m}$ different combinations for MBSs in total. Let all the content combinations in SAPs form a set $\mathcal{D}_s$, and all the content combinations in MBSs form another set $\mathcal{D}_m$. Once the typical user's request arrives, the system first retrieves the content in the local cache device of this user's service base station. If the system finds the required content, transmission between the typical user and the service base station happens. Otherwise, the required content will be searched in the core network and additional energy consumption will be introduced. Each base station fetches the contents using its backhaul link so that the combination of the contents in local cache device can be refreshed after a specific period.

C. ENERGY CONSUMPTION MODEL

In the context where caching and wireless network are integrated, an extensive power model that takes into account various detailed factors is widely adopted [22]. In our considered CUDN scenario, the following three components with respect to the total power consumption at an operating BS are considered:

i) Transmit power: As defined before, the transmit power of the i-th SAP and the i-th MBS is denoted as P_i^s and P_i^m, respectively. The accumulative transmit power of the service base station is proportional to the number of its associated users.

ii) Operational charge power: The operational power at a SAP and a MBS is P_o^s and P_o^m, respectively.

iii) Data retrieval power: If the data is retrieved from the local cache device, the consumed data retrieving power is denoted as P_{lr}. Otherwise, if the requested content is retrieved via the backhaul, the consumed data retrieving power is represented as P_{br}.[1]

D. PROBLEM FORMULATION

In this paper, we focus on maximizing the EE of the system. According to the power model described above, the total average power consumed by the system can be represented as follows,

$$\begin{aligned} P_{\text{sys}} = \sum_{i \in \mathcal{U}} \big(& I_i^{\text{idx}} P_{j,i}^s + (1 - I_i^{\text{idx}}) P_{l,i}^m + (P_{lr,i} \times (1 - \mathbb{P}_{\text{out}}) \\ & + P_{br,i} \times \mathbb{P}_{\text{out}}) \big) + \sum_{q \in \mathcal{S}} P_{o,q}^s + \sum_{k \in \mathcal{M}} P_{o,k}^m, \end{aligned} \tag{4}$$

[1]Note that the data retrieval power may be varying in practical systems. Especially for the case that the required data have to be retrieved via the backhaul links, so that the retrieval power is related to the link between the user and the server. Then, the dynamic network topology has to be considered. In this paper, as we focus on the basic methodology for cache placement strategy, we consider the data retrieval power P_{lr} and P_{br} to be constant to simplify the analysis. In future works, we will extend our work to cover the dynamic network topology. Even though, we also provide the performance comparison for different retrieval power to give some insights in this paper. Please refer to Section IV.

where $\mathbb{P}_{\text{out}}$ denotes the probability that the requested content is not within the local cache combination. Obviously, $\mathbb{P}_{\text{out}}$ is highly related to the chosen cache combination from the content library. We model the content requests arrival of the users as a poisson process. During each period, each user submits content requests to the service SAP or MBS with a density of λ_r. We assume that the users' demand for the content is greedy and the requests density λ_r is large enough, so that the system throughput is limited by spectrum efficiency [7]. Thus, the system EE can be formulated as:

$$\text{EE} = \frac{\sum_{r_i^u \in \Phi_u} \log(1 + \text{SINR}(r_i^u))}{P_{\text{sys}}}. \tag{5}$$

Mathematically, the system EE optimization problem can be formulated as

$$\begin{aligned} &\arg\max_{a_{I_s}, a_{I_m}} \ \text{EE} \\ &s.t.\ \text{SINR}(r_i^u) \geq \gamma \\ &\quad I_s \in \mathcal{D}_s, \quad I_m \in \mathcal{D}_m \\ &\quad a_{I_s}, a_{I_m} \in \{0, 1\}, \end{aligned} \tag{6}$$

where the constraint γ is the minimal SINR threshold satisfying the user's communication quality-of-service (QoS). We use the optimization variables a_{I_s}, a_{I_m} to represent the cache strategy.[2] If the I_s-th and the I_m-th cache combination are selected from their corresponding cache libraries, $a_{I_s} = 1$ and $a_{I_m} = 1$, otherwise $a_{I_s} = 0$ and $a_{I_m} = 0$. Therefore, the optimization Eq.(6) can be regarded as a l_0-norm problem. Due to the unknown parameter $\mathbb{P}_{\text{out}}$, the nonconvex variables and the discontinuous l_0-norm in the objective function, it is challenging to solve the optimization problem Eq.(6). To the best of our knowledge, the existing works coping with these challenges mainly lie in the following two aspects.

For one aspects, earlier works suppose that the popularity distribution of the content is perfectly known, and use a smooth function to approach the l_0-norm problem [5]–[7], [23], [25]. For example, to solve the l_0-norm problem, Tao *et al.* [23] have adopted the smoothed l_0-norm approximation and compared the performance of three smooth functions, including logarithmic function, exponential function, and arctangent function. The resulting approximated problem is then transformed into the difference of convex programming problems. As the distribution of content popularity is perfectly known, the relationship between the parameter $\mathbb{P}_{\text{out}}$, the library size F and the capacity of the local cache device N_s, N_m can be derived analytically. By the approximation of l_0-norm problem, the objective function of the optimization problem can be converted into a smooth and convex form. Therefore, the primal problem can be solved to get the optimal cache strategy. However, this two relaxations described above are too ideal in real system. To avoid these unpractical assumptions, we focus on a more realistic scenario where the popularity of the cached content is dynamic and unknown in this paper.

For the other aspect, the latest works focus on a more realistic case where the content popularity is dynamic and unknown [9], [10]. The relationship between the parameter $\mathbb{P}_{\text{out}}$, the library size F and the capacity of the local cache device N_s, N_m is derived based on the estimated content popularity. Then, the optimization problem can be solved analytically, and the cache content placement strategy at the local cache device can be optimized. However, the estimation of the cache popularity will inevitability introduce deviation so that the corresponding cache placement strategy can not achieve optimal performance. What's more, the estimation based method cannot cope with the dynamic content popularity distribution and the resulting performance is very poor in the case of dynamic content.

[2]Obviously, both the power consumption P_{sys} and the SINR in the objective function Eq.(5) are related to the cache strategy. Unfortunately, it is hard to explicitly expresss these two items as the function of a_{I_s} and a_{I_m}.

III. DEEP REINFORCEMENT LEARNING BASED CACHE STRATEGY

Based on the discussion in section II, we propose an online cache strategy learning algorithm based on DRL in the scenario where the popularity of the cached content is dynamic and unknown in this section.

A. BRIEF OVERVIEW OF DEEP REINFORCEMENT LEARNING

The principle of reinforcement learning (RL) can usually be described as a Markov Decision Process (MDP) [24]. Let $\mathcal{X} = \{x_1, \cdots, x_n\}$ be the state space and $\mathcal{A} = \{a_1, \cdots, a_m\}$ be the action set. Based on the current state $x(t) \in \mathcal{X}$, the agent applies an action $a(t) \in \mathcal{A}$ into the environment and then the system transfers to a new state $x(t+1) \in \mathcal{X}$ according to the transition probability $\mathbb{P}_{x(t)x(t+1)}(a)$. This process gives the agent an immediate reward $R\big(x(t), a(t)\big)$ and a long-term cumulative discounted reward $V(x)$. The long-term cumulative discounted reward from state x can be expressed by the following state-value function:

$$V(x) = \mathbb{E}\left[\sum_{t=0}^{\infty} \epsilon^t R\big(x(t), a(t)\big) | x(0) = x\right], \tag{7}$$

where $\mathbb{E}[\cdot]$ denotes the expectation operation, and $0 < \epsilon < 1$ is the discount factor. In RL, the goal of the agent is to find an optimal policy $a^* = \pi^*(x) \in \mathcal{A}$ for each state to maximize the cumulative reward over a long time. Due to the Markov property, the state-value function can be rewritten as

$$V^{\pi}(x) = R(x, \pi(x)) + \epsilon \sum_{x' \in \mathcal{X}} \mathbb{P}_{xx'}\big(\pi(x)\big) V^{\pi}(x'), \tag{8}$$

where x' represents the state at the next time instant. The optimal policy π^* follows Bellman's criterion, i.e.,

$$V^{\pi^*} = \max_{a \in \mathcal{A}} \left[R(x, a) + \epsilon \sum_{x' \in \mathcal{X}} \mathbb{P}_{xx'}(a) V^{\pi^*}(x')\right]. \tag{9}$$

Given the reward R and the transition probability $\mathbb{P}_{xx'}$, the optimal policy can then be obtained. When R and $\mathbb{P}_{xx'}$

are unknown, Q-learning is one of the most widely-used strategies to determine the best policy π^*. The Q-function is defined as

$$Q^{\pi}(x,a) = R(x,a) + \epsilon \sum_{x' \in \mathcal{X}} \mathbb{P}_{xx'}(a) V^{\pi}(x'). \tag{10}$$

Usually, the Q-function is obtained via a recursive manner by using the available information tuple (x, a, R, x'). The update of the Q-function is

$$Q_{t+1}(x,a) = Q_t(x,a) + \eta\big(R_{t+1} + \epsilon\big[\max_{a'} Q_t(x',a')\big] - Q_t(x,a)\big), \tag{11}$$

where η is the learning rate. According to [24], $Q_t(x,a)$ will definitely converge to $Q^*(x,a)$ while $t \to \infty$.

Unfortunately, we can not find a precise Q-function of the real environment for most of the time. Therefore, the deep neural network $Q(x,a;\boldsymbol{\theta})$ has been used to approximate the Q-function $Q(x,a)$, where $\boldsymbol{\theta}$ is the weight vector of the neural network. The deep neural network can be trained by minimizing the following loss function:

$$L_{\text{loss}}(\boldsymbol{\theta}) = \mathbb{E}\big[(y_t - Q(x,a;\boldsymbol{\theta}_t))^2\big], \tag{12}$$

where $y_t = \mathbb{E}\big[R_t + \epsilon \max_{a'} Q(x',a';\boldsymbol{\theta}_{t-1}) | x, a\big]$ is the target for interaction t.

B. DEEP REINFORCEMENT LEARNING BASED CACHE STRATEGY

In order to apply DRL into the cache placement of a CUDN, we first define some important notations under the framework of DRL.

1) SYSTEM STATES

At the beginning of a specific period, each base station fetches the contents via its backhaul link to get its content combination. In our scenario, there are $|\Phi_s| + |\Phi_m|$ base stations, where $|\cdot|$ represents the cardinality of a set. The current system state $x(t)$ can be defined as

$$x(t) = \{c_1^s(t), \cdots, c_{|\Phi_s|}^s(t), c_1^m(t), \cdots, c_{|\Phi_m|}^m(t)\}, \tag{13}$$

where $c_i^s(t) \in \mathcal{D}_s, i \in \{1, \cdots, |\Phi_s|\}$ is the cached content combination at the ith SAP, and $c_j^m(t) \in \mathcal{D}_m, j \in \{1, \cdots, |\Phi_m|\}$ is the cached content combination at the j-th MBS.

2) SYSTEM ACTIONS

During each period, the SAPs and MBSs should decide which content combination to be cached so that the system performance in terms of EE can be improved. The actions of the SAPs and MBSs are the chosen content combinations. Therefore, the current action $a(t)$ is denoted by

$$a(t) = \{a_1^s(t), \cdots, a_{|\Phi_s|}^s(t), a_1^m(t), \cdots, a_{|\Phi_m|}^m(t)\}, \tag{14}$$

where $a_i^s \in \mathcal{D}_s$ and $a_j^m \in \mathcal{D}_m$.

3) REWARD FUNCTION

The system reward represents the optimization objective. In this paper, the objective is to maximize the system EE, and the reward function is defined as Eq. (15), shown at the bottom of the next page, where $\mathcal{I}_i^u$ is shown in Eq.(3).

It follows Eq.(15) that variables I_i^{idx} and $\mathbb{P}_{\text{out}}$ prevent the system from getting the immediate reward. Fortunately, for our DRL approach, these two variables will degenerate into known quantities. The underlying reasons are as follows. First, I_i^{idx} can be determined when the distribution of the nodes, including the users, the SAPs and the MBSs, and the associated schedule of each user are given. Second, in our scenario, when the system state $x(t)$ and system action $a(t)$ are given, the content combination in the local cache device of the base station can be determined. When the content requests of a typical user arrive, the system can get the knowledge whether the local cache combination satisfies the requests by retrieving the local content combination. Therefore, $\mathbb{P}_{\text{out}}$ degenerates into a deterministic parameter that is given as follows,

$$\mathbb{P}_{\text{out}} = \begin{cases} 0, & \text{the requests are satisfied,} \\ 1, & \text{otherwise.} \end{cases} \tag{16}$$

As described in Section III-A, the optimal caching policy π^* can be determined by optimizing the Q-function, i.e., $\arg\max Q^{\pi}(x(t), a(t))$. For our considered CUDN, it is obvious that the number of possible system states can be very large. Due to the curse of high-dimensionality, it is very difficult for traditional approaches to handle our problem. Fortunately, as deep Q neural network is capable of directly learning from high-dimensional inputs, it is adopted in our system.

Two techniques are applied to modify the regular Q-learning into deep Q neural network. The first one is experience replay. At each time instant t, the system stores its interaction experience tuple $e(t) = (x(t), a(t), R(t), x(t+1))$ in a replay memory and forms a experience pool $\mathcal{D}_{\text{memory}}(t) = \{e(1), \cdots, e(t)\}$. Then, the networks are trained by sampling from the experience pool. The other modification is that the system uses two neural networks, which are net_target and net_evaluate, to approach the Q-function, i.e.,

$$Q_{t+1}(x,a;\boldsymbol{\theta}) = \overbrace{Q_t(x,a;\boldsymbol{\theta})}^{\text{net\_evaluate}} + \eta\Big(R_{t+1} + \underbrace{\epsilon\big[\max_{a'} Q_t(x',a';\boldsymbol{\theta}^-)\big]}_{\text{net\_target}} - \underbrace{Q_t(x,a;\boldsymbol{\theta})}_{\text{net\_evaluate}}\Big). \tag{17}$$

For net_target, the parameter $\boldsymbol{\theta}$ is updated per S time steps, i.e., $\boldsymbol{\theta}_i^- = \boldsymbol{\theta}_{i-S}$. In contrast, the parameter $\boldsymbol{\theta}$ in net_evaluate is updated for each time step. It can be proved that this modification can reduce the correlation between these two neural networks and make the learning process more stable [15]. Note that, Eq.(17) is updated based on the ε-greedy policy with $\varepsilon \in [0, 1]$ to balance the exploration and exploitation [24].

In the standard DRL algorithm, we use the regular multi-tier deep neural networks (DNN) to realize the deep Q neural network. The DNN can be trained by minimizing the loss function Eq.(12) via stochastic gradient descent (SGD) algorithm, and the weights are updated based on the loss gradient by back-propagation [24]. In this case, we iteratively compute a forward pass y_t and a backward pass $\partial L/\partial \boldsymbol{\theta}$. There are several optimizers provided in the TensorFlow [20], among which we choose the *AdamOptimizer* for our DNN training. The proposed DRL based cache (DRLC) algorithm is summarized in Algorithm 1.

Algorithm 1 DRL Based Cache Strategy

Initialization

Initialize the replay memory $\mathcal{D}_{\text{memory}}$ with capacity C.
Initialize net_evaluate with random weights $\boldsymbol{\theta}$.
Initialize net_target with weights $\boldsymbol{\theta}^- = \boldsymbol{\theta}$.

For episode $= 1, \cdots, K$ **do**

Initialize the initial state sequence $x(0)$ with random content combinations.

For $t = 1, \cdots, T$ **do**

Select action $a(t)$ based on ε-greedy policy:
a random action $a(t)$ with probability ε,
or $a(t) = \arg\max_a Q_{\text{net\_evaluate}}(x, a; \boldsymbol{\theta})$ with probability $1-\varepsilon$.

Execute action $a(t)$, observe the reward of Eq.(15).

Store tuple $(x(t), a(t), R(t), x(t+1))$ in $\mathcal{D}_{\text{memory}}$.

Randomly sample tuple $(x(j), a(j), R(j), x(j+1))$

Set $y_j = \begin{cases} R(j), \text{ if episode terminates at step } j+1 \\ R(j) + \epsilon \max Q_{\text{net\_target}}(x, a; \boldsymbol{\theta}^-), \text{otherwise} \end{cases}$.

Train the DNN by minimizing the loss function $(y_j - Q_{\text{net\_evaluate}}(x, a; \boldsymbol{\theta}))^2$.

Update $Q_{\text{net\_target}} \leftarrow Q_{\text{net\_evaluate}}$ every S steps.

End For

End For

The DRL-based caching algorithm can achieve the optimal cache strategy in an interactive manner. As presented in Algorithm 1, the procedure generates a caching placement strategy based on the network parameters obtained from the history experience tuples, and observes the reward from the environment. And then, the procedure updates the history experience tuples and trains the neural networks. This try-and-observe operation avoids the prior knowledge of the environment, i.e., the popularity of the contents. By enough trials, the well trained neural networks will nearly perfectly characterize the environment. Therefore, the cache placement strategy given by Algorithm 1 can adapt the real environment.

C. ADVANCED DEEP REINFORCEMENT LEARNING BASED CACHE STRATEGY

For the past few years, some exciting finds have been reported in the fields of both RL and DL, e.g., prioritized experience replay for history action tuple selection [17], dueling architecture for Q-functon [18], and deep recurrent neural network (RNN) [19] for deep Q-network. In what follows, we further utilize them to improve our DRL based cache strategy.

1) PRIORITIZED EXPERIENCE REPLAY FOR HISTORY EXPERIENCE TUPLE SELECTION

In DRLC, the experience tuples are randomly sampled from the experience pool regardless of their significance. However, the RL agent may learn more effectively from some tuples than the others. Meanwhile, some tuples may not be immediately useful for the agent, but might become useful as the episode increases. It has been proved that selecting action $a(t)$ with prioritized experience replay can improve the performance and speed up the algorithm [17]. For prioritized experience replay, the sampling probability of experience tuple i can be defined according to the priority as follows,

$$Pr(i) = \frac{\hat{p}_i^{\hat{\gamma}}}{\sum_k \hat{p}_k^{\hat{\gamma}}}, \tag{18}$$

where $\hat{p}_i > 0$ is the priority of tuple i. $\hat{p}$ can be computed as $\hat{p}_i = |\delta_i| + \hat{\epsilon}$, where $\hat{\epsilon}$ is a small positive constant, and δ_i is the temporal-difference (TD) error. δ_i can be computed as follows,

$$\delta_i = R_i + \epsilon Q_{\text{net\_target}}(x_i, \arg\max_a Q_{\text{net\_evaluate}}(x_i, a)) - Q_{\text{net\_evaluate}}(x_{i-1}, a_{i-1}). \tag{19}$$

The factor $\hat{\gamma}$ controls the effect of priority. Specifically, $\hat{\gamma} = 0$ corresponds to the uniform sampling.

2) DUELING ARCHITECTURE FOR Q-FUNCTION

In contrast to the standard deep Q neural network RL algorithm, the dueling networks architecture denotes the Q-function as two separate estimators. A estimator is used for the state value function, and another one is used for the state-dependent action advantage function, i.e.,

$$Q(x, a; \boldsymbol{\theta}) = V(x; \boldsymbol{\theta}) + A(x, a; \boldsymbol{\theta}), \tag{20}$$

where $V(x; \boldsymbol{\theta})$ is the value of state, and $A(x, a; \boldsymbol{\theta})$ is the advantage function expressed as $A^\pi(x, a; \boldsymbol{\theta}) = Q^\pi(x, a; \boldsymbol{\theta}) - V^\pi(x; \boldsymbol{\theta})$. This decomposition can generalize the learning across actions without imposing any change to underlying RL algorithm. Therefore, this architecture can lead to better policy evaluation in the presence of many similar-valued actions [18].

$$R(t) = \frac{\sum_{r_i^u \in \Phi_u} \log\left(1 + \frac{I_i^{\text{idx}} |h_{j,i}^s|^2 P_{j,i}^s b |r_i^u - r_j^s|^{-\alpha} + (1 - I_i^{\text{idx}}) |h_{l,i}^m|^2 P_{l,i}^m b |r_i^u - r_l^m|^{-\alpha}}{\sigma^2 + \mathcal{I}_i^u}\right)}{\sum_{i \in \mathcal{U}} \left(I_i^{\text{idx}} P_{j,i}^s + (1 - I_i^{\text{idx}}) P_{l,i}^m + (P_{lr,i} \times (1 - \mathbb{P}_{\text{out}}) + P_{br,i} \times \mathbb{P}_{\text{out}})\right) + \sum_{q \in \mathcal{S}} P_{o,q}^s + \sum_{k \in \mathcal{M}} P_{o,k}^m} \tag{15}$$

3) DEEP RECURRENT NEURAL NETWORK (RNN) FOR DEEP Q NEURAL NETWORK

In the above proposed DRLC procedure, the deep Q-network is usually implemented by a multi-tier DNN. In our considered CUDN, the popularity of the content is highly non-stationary and relevant. From a high-level point of view, Tatar *et al.* [11] show that the popularity evolution of cache content over time can be represented by power-law or exponential distribution, and the content requests of the users are dependent. As the RNN is good at extracting useful information from a correlated sequence [19], we then choose a RNN to replace the DNN used in Algorithm 1 and propose an advantaged DRL based cache placement strategy (ADRLC). The details of the ADRLC are summarized in Algorithm 2.

Algorithm 2 Advanced DRL Based Cache Strategy

Initialization: Initialize the memory $\mathcal{D}_{\text{memory}}$ with capacity C, the net_evaluate with random weights $\boldsymbol{\theta}$, the net_target weight $\boldsymbol{\theta}^- = \boldsymbol{\theta}$, $\Delta = 0$, $\hat{p}_0 = 1$, $w_0 = 1$, update steps S, batch size k, learn rate η, exponents α and β.
For episode $= 1, \cdots, K$ **do**
  Initialize initial state sequence $x(0)$ with random content combinations.
  For $t = 1, \cdots, T$ **do**
    Select action $a(t)$ based on ε-greedy policy: a random action $a(t)$ with probability ε, or $a(t) = \arg\max_a Q_{\text{net\_evaluate}}(x, a; \boldsymbol{\theta})$ with probability $1 - \varepsilon$.
    Execute action $a(t)$, observe the reward of Eq.(15).
    $t = t + 1$, store tuple $(x(t), a(t), R(t), x(t+1))$ in $\mathcal{D}_{\text{memory}}$ with maximal priority $\hat{p}_t = \max_{i<t} \hat{p}_i$.
    If $t \bmod C = 0$ **then**
      For $j = 1, \cdots, k$ **do**
        Sample $(x(j), a(j), R(j), x(j+1))$ with probability $Pr(j) = \frac{\hat{p}_j^{\alpha}}{\sum_i \hat{p}_i^{\alpha}}$
        Compute importance weight $w_j = \frac{(C \cdot Pr(j))^{-\beta}}{\max_i w_i}$
        Compute TD-error δ_j according to Eq.(19).
        Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot (\nabla_{\boldsymbol{\theta}} V(x; \boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} A(x, a; \boldsymbol{\theta}))$.
      End for
      Train the RNN by updating weights $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \cdot \Delta$, and reset $\Delta = 0$.
    End if
    Update $Q_{\text{net\_target}} \leftarrow Q_{\text{net\_evaluate}}$ every S steps.
  End For
End For

D. REMARKS

It this subsection, we discuss the complexity and scalability of our proposed DRL-based cache algorithm.

1) COMPLEXITY

To the best of our knowledge, the latest works [26]–[29] have analyzed the complexity of DRL based on experimental results. From the existing results, it can be concluded that the complexity of our proposed DRL based cache algorithm is dominated by the complexity of training the adopted neural networks and the convergence rate. The complexity of training the adopted neural networks depends on the network depth and the number of neurons [30]–[33]. The convergence rate of the DRL based cache algorithm is dominated by many factors, including the size of content library $\mathcal{F}$, the size of memory pool $\mathcal{D}_{\text{memory}}$, the structure of the adopted neural network $\pi(\cdot|\boldsymbol{\theta})$, the random action selection factor ε of the ε-greedy policy, the learning rate η, and the neural network parameter update step interval S, etc.. It is difficult to analyze the complexity of the DRL based caching algorithm analytically. We have to evaluate the complexity of our proposed algorithms in terms of the convergence steps and the consumed computation time through simulations. Moreover, by elaborately choosing the parameter tuple, i.e., $\{\varepsilon, \eta, S\}$, we can improve the tradeoff between the convergence rate and the global performance. We have performed extensive simulations to find the optimal parameter tuple that can make a good tradeoff between the global performance and the convergence rate.

2) SCALABILITY

Obviously, all the system parameters, including: the cache content library size F, the local cache capacity N_s, N_m, and the access point density λ_m, λ_s, have effect on the scalability of our proposed algorithms. Intuitively, the larger the cache content library size F and the local cache capacity N_s, N_m, the larger the number of cache combinations and the number of neurons in the output layer of the adopted neural network. Therefore, the complexity of training the adopted neural networks becomes higher and more interaction steps are needed for algorithm convergence. Similarly, the larger the access point density, the larger the number of the system sate and action. Then, the complexity and convergence rate of the algorithms will inevitably increase. As it is hard to analyze the effect of these factors on the scalability of our proposed algorithms analytically, we have to evaluate it through simulations. Based on the simulation results, we have found that the proposed learning based algorithms can adapt to complicated scenarios, and our proposed algorithm can always obtain the best performance comparing with the existing methods.

IV. SIMULATION RESULTS

In this section, we provide simulation results to validate our proposed cache placement strategies. The simulation setup is as follows. We consider a two-tier CUDN consisting of MBSs and SAPs. The network covers an area with radius $R_a = 0.5$ km. The MBSs, SAPs and users are distributed in the area according to PPP with density $\lambda_m = 8$, $\lambda_s = 20$ and $\lambda_u = 50$ per kilometer square, respectively. The transmit power of MBS and SAP are $P^m = 66$mW and $P^s = 20$mW, respectively. The path loss exponent $\alpha = -3.5$ and the noise power $\sigma^2 = -127$dBm. The data retrieving power are $P_{lr} = 20$mW in local cache device and

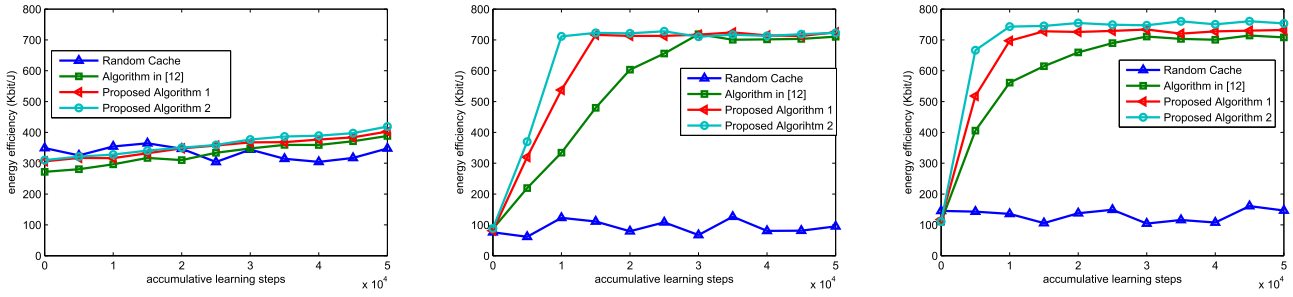


FIGURE 2. Comparison of the four cache algorithms under three types of content popularity distribution: (a) uniform content popularity distribution, (b) zipf content distribution with $\beta = 2$, (c) MovieLens dataset based real-world content popularity distribution. $F = 100, N_s = 12, N_m = 17, P_{lr} = 20mW, P_{br} = 500mW, \lambda_{\text{req}} = 8$.

TABLE 2. Parameters of the adopted Q neural networks. (a) Parameters of adopted DNN. (b) Parameters of adopted CNN. (c) Parameters of adopted RNN.

(a)

Layer	DNN 1	DNN 2	DNN 3	DNN 4
Input size	22	37	35	39
Activation	ReLU	ReLU	ReLu	ReLu
Output size	37	35	39	\

(b)

Layer	Conv 1	Conv 2	Conv 3	FC
Input size	22x22	20x20x10	10x10x15	1280
Filter size	3x3	3x3	4x4	\
Stride	1	2	1	\
Padding	0	1	2	\
No. filter.	10	15	20	39
Activation	ReLU	ReLU	ReLU	ReLU
Output size	20x20x10	10x10x15	8x8x20	\

(c)

Input size	Output size	No. cells	Batch size	Optimizer
22	22	128	32	Adam

$P_{br} = 500$mW in backhaul, respectively. The operation power is $P_o^s = 1500$mW for SAPs and $P_o^m = 2500$mW for MBs, respectively. We set the content size $L = 2$Mbits, the cache content request density $\lambda_r = 8$ per time period. The capacity of the experience memory $\mathcal{D}_{\text{memory}}$ $C = 100$, the deep Q neural network parameter update step interval $S = 20$, and the action selection probability $\varepsilon = 0.15$.

In our simulations, we use Google TensorFlow to implement our advanced deep Q neural network and choose the *AdamOptimizer* [20] to optimize the loss function. The simulations have been performed by our computer with Intel i7-6500 CPU, 16GB RAM, Win10 64-bit system, Pycharm 3.6.3 and tensorflow 1.8.0 environment. We set the learning rate $\eta = 0.001$. For the content popularity, both stationary and dynamic distributions are considered [11]. For stationary distribution, we consider both uniform distribution and Zipf distribution $f_{\hat{\eta}} = (\frac{1}{\hat{\eta}^{\varrho}})/\sum_{i=1}^{F}\frac{1}{i^{\varrho}}$, where $\hat{\eta}$ is the file index, ϱ is the file request coefficient controlling the popularity distribution of contents. For dynamic distribution, we use the MovieLens Dataset [34] to reflect the content request behavior of users. MovieLens is a website that recommends movies for its users operated by GroupLens research group at the University of Minnesota to gather research data on personalized recommendations. This dataset contains 1000209 ratings for 3952 movies provided by 6040 users from the year 2000 to 2003. Each entry of the dataset consists of an anonymous user identity (ID), a movie ID, a rating and a timestamp. We treat each movie rating from a user as a content request of the user (see [35] for a similar approach). As shown in [34], the request of a content is highly related to the timestamp. Therefore, the MovieLens dataset based content requests can be considered as a time-sensitive dynamic distribution. Note that the content popularity is unknown for all cache algorithms in the simulations. We compare our advanced cache strategy with other three cache strategies including: 1) random caching strategy, 2) standard DRL based caching strategy, and 3) the DRL with CNN based caching strategy presented in [12]. The parameters of the adopted Q neural networks are shown in Table 2.

Fig. 2 shows the performance comparison of these four cache algorithms under different popularity distributions. The size of content library is $F = 100$, and the capacity of the cache device in SAPs and MBSs are $N_s = 12$ and $N_m = 17$, respectively. In Fig. 2(a), uniform content popularity distribution is considered. It can be seen that these four cache strategies have nearly the similar performance. The underlying reason is that different cache combinations in local cache device have the same effect on the users' requests when the cache contents follow uniform popularity distribution. Therefore, the cache strategies make no significant different influences on the performance.

Zipf content popularity distribution and MovieLens dataset based real-world content popularity distribution are considered in Fig. 2(b) and Fig. 2(c), respectively. From these figures, we can find that our proposed advanced cache strategy (Algorithm 2) needs the least number of learning steps to get the maximum reward, i.e., energy efficiency. Meanwhile, it can be seen that our proposed advanced cache strategy can adapt to both stationary case, i.e., zipf distribution, and dynamic case, i.e., MovieLens dataset based real-world content popularity distribution. In contrast, from Fig. 2 (c), we can see that the proposed DRL based cache algorithm (Algorithm 1) and the algorithm proposed in [12] suffer

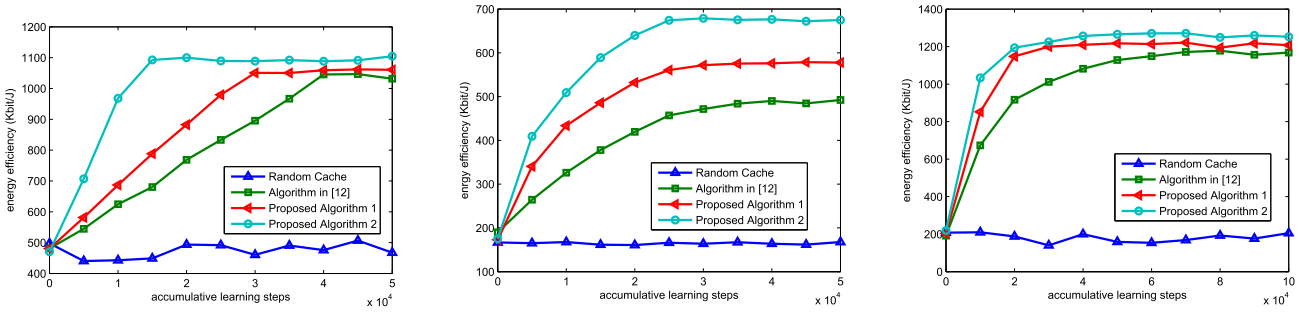


FIGURE 3. Comparison of the four cache algorithms under MovieLens dataset based real-world content popularity distribution with different library size and cache device capacity size: (a) $F = 500$, $N_S = 32$, $N_M = 39$, (b) $F = 1000$, $N_S = 32$, $N_M = 39$, (c) $F = 1000$, $N_S = 120$, $N_M = 150$, with $P_{lr} = 20mW$, $P_{br} = 500mW$, $\lambda_{req} = 8$.

significantly performance degradation, and need more steps to reach the maximal reward for the dynamic case. Moreover, the random cache strategy can only achieve very poor performance in these two cases. The advantage of our proposed advanced DRL based cache algorithm (algorithm 2) is due to the following reasons. First, the prioritized experience replay schedule makes full use of the history experience and speeds up the training of our algorithm. Second, dueling architecture for Q-function leads to better policy evaluation in the presence of many similar-valued actions. Finally, the RNN can extract the correlated information in the users' requests.

To investigate the complexity and scalability of our proposed algorithms, we have performed simulations under different scenarios. Fig. 3 shows the performance comparison of different cache algorithms with MovieLens dataset based real-world popularity distribution under different local cache device capacity (LCDC) tuples. It can be seen that our proposed algorithms can always obtain better performance under different LCDC tuples. Especially, our proposed Algorithm 2 can achieve the best performance. Meanwhile, from Fig. 3, we can find that the parameters F, N_s and N_m have significant effect on the system energy efficiency. The larger the libary size F, the lower the energy efficiency for given cache size N_s and N_m. Moreover, the larger the cache size N_s and N_m is, the higher the energy efficiency for given library size F will be. The underlying reason is that, a larger library size F will lower the hit probability of the cached content and then decrease the system energy efficiency. On the other hand, a larger cache capacity N_s and N_m can increase the hit probability of the cached content and then improve the system energy efficiency. Furthermore, with the increase of content library size and the number of cache nodes, the number of cache combination and the neurons of the output layer in the adopted neural network will become larger. Therefore, from Fig. 3, we can find that the needed interaction steps to reach the maximal reward become larger for all the three learning based algorithms with the increase of the content library size. As a consequence, the complexity of network training becomes higher. Even though, our proposed algorithm 2 always has the fastest converge speed.

When the learning based cache algorithms converge, Fig. 4 shows their achievable energy efficiency under five different access point (including SBS and MBS) densities, including: case 1 with $\{\lambda_m = 4, \lambda_s = 10\}$, case 2 with $\{\lambda_m = 8, \lambda_s = 20\}$, case 3 with $\{\lambda_m = 12, \lambda_s = 30\}$, case 4 with $\{\lambda_m = 6, \lambda_s = 40\}$, and case 5 with $\{\lambda_m = 20, \lambda_s = 50\}$. It can be seen that, with the increase of access point densities, the achievable energy efficiency of all these algorithms decreases. This result is due to the fact that higher access point densities will result in higher system operational charge power, i.e., the sum of P_o^s and P_o^m for all the access points. Therefore, the system energy efficiency decreases. Even though, as shown in this figure, our proposed algorithm 2 still achieves the best performance. So, our proposed algorithm is scalable and can work well under complicated scenarios with large cache nodes density.

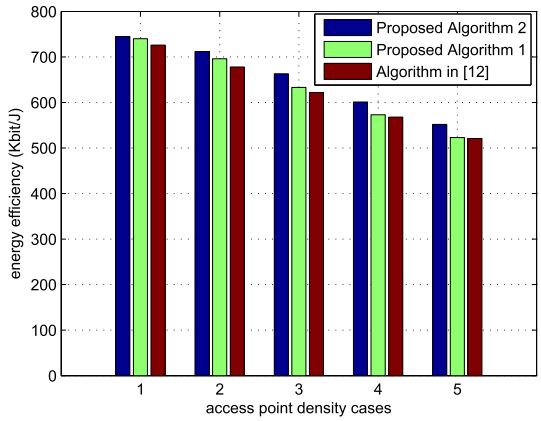


FIGURE 4. Converged energy efficiency under different access point density cases. with $\lambda_u = 50$, $F = 100$, $\lambda_{req} = 8$, $N_= 12$, $N_M = 17$, $P_{lr} = 20mW$, $P_{br} = 500mW$.

The other important parameters affecting the performance of the cache algorithms include: the ε-greedy policy factor ε, the learning rate η and the neural network parameter update step interval S. Table 3 shows the convergence performance and time cost under different parameter tuples $\{\varepsilon, \eta, S\}$ of our proposed algorithm 2 for MovieLens dataset based real-world

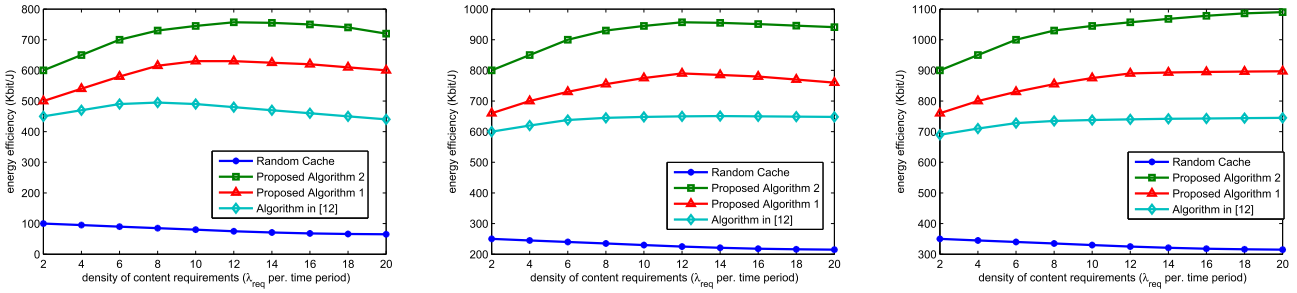


FIGURE 5. The average energy efficiency vs. content request densities with MovieLens dataset based real-world content popularity distribution and $P_{lr} = 20mW$, $P_{br} = 500mW$: (a) $F = 100, N_s = 12, N_m = 17$, (b) $F = 100, N_s = 17, N_m = 22$, (c) $F = 100, N_s = 22, N_m = 27$.

TABLE 3. Convergence performance & time cost comparison under different parameter tuples with $F = 100, N_s = 12, N_m = 17, P_{lr} = 20mW, P_{br} = 500mW, \lambda_{req} = 8$. (a) Convergence performance & time cost for different ε, $\eta = 0.001, S = 20$. (b) Convergence performance & time cost for different η, $\varepsilon = 0.15, S = 20$. (c) Convergence performance & time cost for different S, $\varepsilon = 0.15, \eta = 0.001$.

(a)

ε-value	Conv. steps	converged EE (kbit/J)	time cost (Sec.)
0.05	427	356	167
0.10	1835	434	466
0.15	3174	732	665
0.20	4895	721	975
0.25	8356	731	1289
0.30	14370	733	2053

(b)

η-value	Conv. steps	converged EE (kbit/J)	time cost (Sec.)
0.1	836	472	329
0.05	1316	505	413
0.01	1793	584	467
0.005	2467	603	566
0.001	3174	732	665
0.0005	6245	733	919
0.0001	7986	729	1374

(c)

S-value	Conv. steps	converged EE (kbit/J)	time cost (Sec.)
5	7645	495	1217
10	4991	519	1027
15	4227	602	978
20	3174	732	665
25	3018	701	633
30	3928	647	845

content popularity distribution. Table 3(a) shows the convergence performance and time cost represented by the simulation computation time for different probability ε of random action choice with fixed learning rate and neural network update step interval, i.e., $\eta = 0.001, S = 20$. It can be concluded that a smaller probability ε can result in a faster convergence rate. However, as shown in this table, the smaller probability ε prevents the strategy exploration so that the algorithm may converge to a local optimum. Table 3(b) presents the convergence performance and time cost for different learning rate η with fixed probability ε of random action choice and neural network parameter update interval, i.e., $\varepsilon = 0.15, S = 20$. From Table 3(b), we can see that, the larger the learning rate is, the faster the algorithm converges. However, a larger learning rate may result in a sub-optimal performance. Based on our simulation results, the optimal learning rate $\eta = 0.001$ for given $\varepsilon = 0.15$ and $S = 20$. Table 3(c) provides the convergence performance and time cost for different neural network update step interval under fixed random action probability and learning rate, i.e., $\varepsilon = 0.15, \eta = 0.001$. It can be seen that it will result in a sub-optimal performance if the neural network update interval S is too large or too small. Therefore, by taking into consideration of both the performance and the complexity of the proposed algorithm, we set the parameter tuple to be $\{\varepsilon = 0.15, \eta = 0.001, S = 20\}$.

Table 4 compares the needed learning steps, the obtained energy efficiency and the time cost of the three learning based cache algorithms when they converge. Note that the DNN, RNN and CNN are adopted in the proposed algorithm 1, algorithm 2, and the algorithm in [12], respectively. Similar to the proposed algorithm 2, we have obtained the optimal parameter tuple ε, η, S for the proposed algorithm 1 and the algorithm in [12] via simulations. It can be found that our proposed algorithm 2 can obtain the best EE with the smallest learning steps and computational time cost.

Fig. 5 shows the average energy efficiency of MovieLens dataset based real-world content popularity distribution under different content request densities with three different cache capacity settings. As the cache capacity increases, it can be seen that the average energy efficiency of all the four cache algorithms is significantly improved. Meanwhile, our proposed algorithm 1 and algorithm 2 can achieve better performance. In fact, as the local cache capacity increases, the requested content can be retrieved from the local cache device with a larger probability. This leads to a lower data retrieval power consumption and higher system energy efficiency. On the other hand, the underlying useful information between the user requirements and the unknown content popularity can be learned and extracted by the DRL based algorithms. Therefore, the DRL based cache algorithms with well trained deep neural network can make a better cache combination selection and then obtain better system energy efficiency. Especially, the proposed algorithm 2 can achieve the best system performance by optimizing the parameters and structure of the adopted neural network.

Fig. 6 shows the average energy efficiency of MovieLens dataset based real-world content popularity distribution under different content request densities with three different data

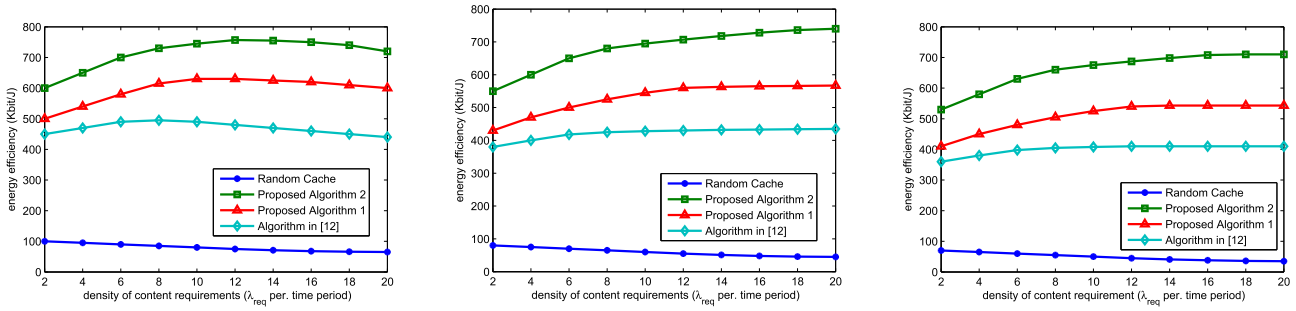


FIGURE 6. The average energy efficiency vs. content request densities under different retrieval power with $F = 100$, $N_s = 12$, $N_m = 17$ and MovieLens dataset based real-world content popularity distribution: (a) $P_{lr} = 20mW$, $P_{br} = 500mW$, (b) $P_{lr} = 30mW$, $P_{br} = 600mW$, (c) $P_{lr} = 40mW$, $P_{br} = 700mW$.

TABLE 4. Convergence performance & time cost comparison under different algorithms (neural networks) with $\lambda_u = 50$, $\lambda_s = 20$, $\lambda_m = 8$, $F = 100$, $N_s = 12$, $N_m = 17$, $P_{lr} = 20mW$, $P_{br} = 500mW$, $\lambda_{req} = 8$.

algorithms	Conv. steps	converged EE (kbit/J)	time cost (Sec.)
Alg. 2	3174	732	665
Alg. 1	11209	692	1322
Alg. [12]	29831	694	5446

retrieval power setting, including: $\{P_{lr} = 20mW, P_{br} = 500mW\}$, $\{P_{lr} = 30mW, P_{br} = 600mW\}$, $\{P_{lr} = 40mW, P_{br} = 700mW\}$. As shown in this figure, the larger the data retrieval power, the smaller the energy efficiency. Meanwhile, our proposed algorithms can still have significant advantage over the algorithm proposed in [12] for different data retrieval power setting. Concretely, our proposed algorithm 2 and algorithm 1 can improve the energy efficiency by 69.8%/73.8%/75.0% and 41.9%/47.2%/48.5% comparing with algorithm in [12] under the simulation scenarios, respectively. Therefore, our proposed DRL based cache algorithms can be applied in real systems for which the data retrieval power is sensitive.

V. CONCLUSIONS

In this paper, we investigate the EE of CUDNs. We consider a realistic scenario where the popularity of the cache content is dynamic and unknown. We propose novel online learning cache algorithms based on deep reinforcement learning. We further optimize the structure and the parameters of the deep Q-network according to the latest findings in deep reinforcement learning, including prioritized experience replay for history experience tuple selection, dueling architecture for Q-function, and deep RNN for deep Q-network, to improve the performance. We use Google TensorFlow to implement our cache algorithm. Simulation results show that our proposed cache algorithms can achieve considerable performance improvement comparing with existing schemes for both stationary and dynamic popularity distributions.

REFERENCES

[1] J. Park, S. Y. Jung, S.-L. Kim, M. Bennis, and M. Debbah, "User-centric mobility management in ultra-dense cellular networks under spatio-temporal dynamics," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Washington, DC, USA, Dec. 2016, pp. 1–6.

[2] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya, "Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 337–368, 1st Quart., 2014.

[3] C. Wang, Y. He, F. Yu, Q. Chen, and L. Tang, "Integration of networking, caching, and computing in wireless systems: A survey, some research issues, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 7–38, 1st Quart., 2017.

[4] M. Kamel, W. Hamouda, and A. Youssef, "Ultra-dense networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2522–2545, 4th Quart., 2016.

[5] Y. Cui and D. Jiang, "Analysis and optimization of caching and multicasting in large-scale cache-enabled heterogeneous wireless networks," *IEEE Trans. Wireless Commun.*, vol. 16, no. 1, pp. 250–264, Jan. 2017.

[6] B. Zhou, Y. Cui, and M. Tao, "Optimal dynamic multicast scheduling for cache-enabled content-centric wireless networks," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2015, pp. 1412–1416.

[7] B. Zhou, Y. Cui, and M. Tao, "Stochastic content-centric multicast scheduling for cache-enabled heterogeneous cellular networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 9, pp. 6284–6297, Sep. 2016.

[8] Y. Chiang and W. Liao, "ENCORE: An energy-aware multicell cooperation in heterogeneous networks with content caching," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[9] B. N. Bharath, K. G. Nagananda, and H. V. Poor. (Aug. 2015). "A Learning-Based Approach to Caching in Heterogenous Small Cell Networks." [Online]. Available: http://arxiv.org/abs/1508.03517

[10] S. Müller, O. Atan, M. van der Schaar, and A. Klein, "Smart caching in wireless small cell networks via contextual multi-armed bandits," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2016, pp. 1–7.

[11] A. Tatar, M. D. de Amorim, S. Fdida, and P. Antoniadis, "A survey on predicting the popularity of Web content," *J. Internet Services Appl.*, vol. 5, no. 1, pp. 1–20, 2014.

[12] Y. He *et al.*, "Deep-reinforcement-learning-based optimization for cache-enabled opportunistic interference alignment wireless networks," *IEEE Trans. Veh. Technol.*, vol. 66, no. 11, pp. 10433–10445, Nov. 2017.

[13] S. O. Somuyiwa, A. György, and D. Gündüz, "A reinforcement-learning approach to proactive caching in wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1331–1344, Jun. 2018.

[14] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, "Optimal and scalable caching for 5g using reinforcement learning of space-time popularities," *IEEE J. Sel. Topics Signal Process.*, vol. 12, no. 1, pp. 180–190, Feb. 2018.

[15] V. Mnih *et al.* (2013). "Playing Atari with deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/1312.5602

[16] H. Van Hasselt, A. Guez, and D. Silver, "Deep feinforcement learning with double Q-learning," in *Proc. 13th AAAI Conf. Artif. Intell.* Menlo Park, CA, USA: AAAI Press, 2016, pp. 2094–2100.

[17] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. (2015). "Prioritized experience replay." [Online]. Available: https://arxiv.org/abs/1511.05952

[18] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. (2015). "Dueling network architectures for deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/1511.06581

[19] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

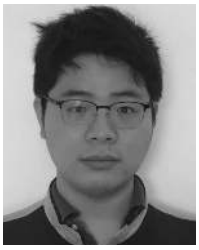
[20] M. Adadi *et al.*, "TensorFlow: A system for large-scale machine learning," Google Brain, Mountain View, CA, USA, Tech. Rep. 2016, vol. 16.

[21] Y. Gao, L. Dai, and X. Hei, "Throughput optimization of multi-BSS IEEE 802.11 networks with universal frequency reuse," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3399–3414, May 2017.
[22] B. Perabathini, E. Ba tuğ, M. Kountouris, M. Debbah, and A. Conte, "Caching at the edge: A green perspective for 5G networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, London, U.K., Jun. 2015, pp. 2830–2835.
[23] M. Tao, E. Chen, H. Zhou, and W. Yu, "Content-centric sparse multicast beamforming for cache-enabled cloud RAN," *IEEE Trans. Wireless Commun.*, vol. 15, no. 9, pp. 6118–6131, Sep. 2016.
[24] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1999.
[25] X. Chen, "Smoothing methods for nonsmooth, nonconvex minimization," *Math. Programm.*, vol. 134, no. 1, pp. 71–99, Aug. 2012.
[26] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2016, pp. 1–7.
[27] S. Zhang *et al.*, "Architectural complexity measures of recurrent neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1822–1830.
[28] J. Kouthník, J. Schmidhuber, and F. Gomez, "Online evolution of deep convolutional network for vision-based reinforcement learning," in *From Animals to Animats*, vol. 13. Cham, Switzerland: Springer, pp. 260–269. International Publishing, Cham, pp, 260-269.2014.
[29] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos. (2017). "Learning to optimize: Training deep neural networks for wireless resource management." [Online]. Available: https://arxiv.org/abs/1705.09412
[30] A. Graves. (2016). "Adaptive computation time for recurrent neural networks." [Online]. Available: https://arxiv.org/abs/1603.08983
[31] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2016. [Online]. Available: https://arxiv.org/abs/1605.07678
[32] S. Zhang *et al.* (2016). "Architectural complexity measures of recurrent neural networks." [Online]. Available: https://arxiv.org/abs/1602.08210
[33] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 5353–5360.
[34] F. M. Harper and J. A. Konstan, "The MovieLens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 1–19, Dec. 2015.
[35] S. Müller, O. Atan, M. van der Schaar, and A. Klein, "Context-aware proactive content caching with service differentiation in wireless networks," *IEEE Trans. Wireless Commun.*, vol. 16, no. 2, pp. 1024–1036, Feb. 2017.

WEI LI was born in Chongqing, China, in 1988. He received the B.S. and M.S. degrees in communication engineering from the Chongqing University of Posts and Telecommunications (CQUPT), Chongqing, in 2012 and 2015, respectively. He is currently pursuing the Ph.D. degree with the National Key Laboratory of Science and Technology on Communications, University of Electronic Science and Technology of China (UESTC). His current research interests include ultra-dense networks' resource management and deep reinforcement learning.

JUN WANG (S'03–M'09) received the B.S. degree in communication engineering and the M.S. and Ph.D. degrees in communication and information systems from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 1997, 2000, and 2009, respectively.

Since 2000, he has been with the National Key Laboratory of Science and Technology on Communications, UESTC, where he is currently a Professor. His research interests include signal processing for wireless communications and machine learning for wireless communications.



GUOYONG ZHANG received the B.S. degree from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2014, where he is currently pursuing the Ph.D. degree with the National Key Laboratory of Science and Technology on Communications. His current research interests include cognitive radio and signal processing for wireless communications.

LI LI was born in Chongqing, China, in 1994. He received the B.S. degree from the University of Electronic Science and Technology of China (UESTC), in 2016, where he is currently pursuing the Ph.D. degree with the National Key Laboratory of Science and Technology on Communications. His current research interests include signal processing algorithms and anti-jamming technology based on machine learning.

ZE DANG received the B.S. degree from Dalian University, Dalian, China, in 2017. She is currently pursuing the M.S. degree with the National Key Laboratory of Science and Technology on Communications, University of Electronic Science and Technology of China (UESTC). Her current research interests include interference cognition and machine learning.

SHAOQIAN LI (M'02–SM'12–F'16) received the B.E. degree in communication technology from Xidian University, Xi'an, China, in 1981, and the M.E. degree in information and communication systems from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 1984.

In 1984, he joined UESTC as an Academic Member, where he has been a Professor of information and communication systems, since 1997, and a Ph.D. Supervisor, since 2000. He is currently the Director of the National Key Laboratory of Science and Technology on Communications, UESTC. He holds more than 60 granted and filed patents. His general interests include the areas of wireless and mobile communications, anti-jamming technologies, and signal processing for communications' subjects. He has published more than 100 journal papers, 100 conference papers, and two edited books in the above-mentioned fields. His current research topics focus on multiple-antenna signal processing technologies for mobile communications, cognitive radios, and coding and modulation for the next-generation mobile broadband communications systems.

Mr. Li has been a member of the Communication Expert Group, National 863 Plan, since 1998, and the Future Project, since 2005. He is currently a member of the Board of Communications and Information Systems, Academic Degrees Committee, State Council of China, and the Expert Group of Key Special-Project on Next-Generation Broadband Wireless Mobile Communications of China (approved by the State Council, since 2007). He has served as a Technical Program Committee (TPC) Member for various IEEE conferences. He is also an Editorial Board Member of the *Chinese Science Bulletin* and the *Chinese Journal of Radio Science*. He was a TPC Co-Chair of 2005, 2006, and 2008 IEEE International Conference on Communications, Circuits, and Systems.