Copyright © 2019 Tech Science Press CMC, vol.59, no.1, pp.89-104, 2019

Deep Q-Learning Based Computation Offloading Strategy for Mobile Edge Computing

Yifei Wei[1, *], Zhaoying Wang[1], Da Guo[1] and F. Richard Yu[2]

Abstract: To reduce the transmission latency and mitigate the backhaul burden of the centralized cloud-based network services, the mobile edge computing (MEC) has been drawing increased attention from both industry and academia recently. This paper focuses on mobile users' computation offloading problem in wireless cellular networks with mobile edge computing for the purpose of optimizing the computation offloading decision making policy. Since wireless network states and computing requests have stochastic properties and the environment's dynamics are unknown, we use the model-free reinforcement learning (RL) framework to formulate and tackle the computation offloading problem. Each mobile user learns through interactions with the environment and the estimate of its performance in the form of value function, then it chooses the overhead-aware optimal computation offloading action (local computing or edge computing) based on its state. The state spaces are high-dimensional in our work and value function is unrealistic to estimate. Consequently, we use deep reinforcement learning algorithm, which combines RL method Q-learning with the deep neural network (DNN) to approximate the value functions for complicated control applications, and the optimal policy will be obtained when the value function reaches convergence. Simulation results showed that the effectiveness of the proposed method in comparison with baseline methods in terms of total overheads of all mobile users.

Keywords: Mobile edge computing, computation offloading, resource allocation, deep reinforcement learning.

1 Introduction

As smartphones are getting more and more popular, a variety of new mobile applications such as face recognition, natural language processing, augmented reality are becoming an increasing part of daily life, and thus people need high rate computation and high amount of computational resource [Wang, Liang, Yu et al. (2017)]. As we all know, cloud computing depends on its powerful centralized computing capability which meets the demands of resource-limited end users for effective computation. However, moving all the distributed data and high demand computation applications to the cloud server will result in heavy burden on network performance and the long latency for resource

[1] Beijing Key Laboratory of Work Safety Intelligent Monitoring, School of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing, 100876, China.

[2] Department of Systems and Computer Engineering, Carleton University, Ottawa, ON K1S 5B6, Canada.

[*] Corresponding Author: Yifei Wei. Email: weiyifei@bupt.edu.cn.

transmission between users and cloud computing devices, which will degrade the quality of service [Shi, Cao, Zhang et al. (2016); Bao and Ding (2016)].

In order to further reduce the latency and enhance the network performance while provide powerful computational capability for end users, the mobile edge computing (MEC) has been proposed to deploy computing resources closer to the end users. As a remedy to the problem which cloud computing exists, mobile edge computing makes the function of cloud at the edge of the networks which obtains a tradeoff between computation-intensive and latency-critical for mobile users [Mao, You, Zhang et al. (2017)]. Mobile edge computing enables the mobile user's equipment (UEs) execute computation offloading by sending computation tasks to the MEC server through wireless cellular networks [Wang, Liang, Yu et al. (2017)], which means the MEC server executes the computational task on behalf of the UE. In mobile edge computing, network edge devices such as base stations, access points and routers, are empowered with computing and storage capabilities to serve users' requests as a substitute of clouds [Patel, Naughton, Chan et al. (2014)]. In this paper, we consider an edge system as the combination of an edge device (macro-cell) and the associated edge servers which provides IT services, environments and cloud computing capabilities to meet mobile users' low-latency and high-bandwidth service requirements.

The survey of computation offloading illustrated two objectives for computation offloading: reduce the execution time and mitigate the energy consumption [Kumar, Liu, Lu et al. (2013)]. Computation offloading decisions are classified into two parts: what computation to offload, and where to offload computation. The decision in regard to what computation to offload is generally considered as the partitioning problem which computation task are partitioned into different components and make decision of whether to offload each component or not [Huang, Wang and Niyato (2012); Alsheikh, Hoang, Niyato et al. (2015)]. Another decision of where to offload computation tasks focuses on the binary decision of local computation or offloading the computation task to computing devices, which is similar to the decision of what to offload.

A number of previous works have discussed the computation offloading and resource allocation problem in mobile edge computing scenarios [Yu, Zhang and Letaief (2016); Mao, Zhang, Song et al. (2017); Mao, Zhang and Letaief (2016)]. Wang et al. [Wang, Liang, Yu et al. (2017)] considered the computation offloading decision, physical resource block (PRB) allocation, and MEC computation resource allocation as optimization problems in wireless cellular networks by graph coloring method. Xu et al. [Xu and Ren (2017)] presented the optimal policy of dynamic workload offloading and edge server provisioning to minimize the long-term system cost (including both service delay and operational cost) by using online learning algorithm which including value iteration and reinforcement learning (RL). While Liu et al. [Liu, Mao, Zhang et al. (2016)] proposed a two-timescale stochastic optimization problem as the Markov decision process in MEC scene and solved the problem using linear programming problem.

In this paper, we focus on the computation offloading decision making problem of whether to compute on local equipment or to offload the task to the MEC server for cloud computing and propose an efficient deep reinforcement learning (DRL) scheme. By making the right decision of computation offloading, mobile user can enhance the computation efficiency and decrease the energy consumption. Each agent learns through interactions with the environment and evaluates its performance in the form of value function. Since wireless network states and computing requests have stochastic properties which causes the value function is intractable to evaluate by traditional RL algorithm, we apply the deep neural network (DNN) to approximate the action-value function with a reinforcement learning method deep Q-learning. Each agent chooses action in state and receives an immediate reward, then it uses DNN to approximate the value functions. After the value functions reach convergence, the user is capable to select the overhead-aware optimal computation offloading strategy based on its state and learning results. We aim to minimize the total overheads in terms of computational time and energy consumption of all users. Simulation results have proved that the proposed deep reinforcement learning based computation offloading policy performances effectively compared with baseline methods in this work.

2 System models and problem formulation

In this section we will introduce the system models including network model, communication model and computation model adopted in this work.

2.1 Network model

An environment of one macrocell and N small cells in the terminology of LTE standards is considered here. An MEC server is placed in the macro eNodeB (MeNB), and all the small cell eNodeBs (SeNBs) are connected to the MeNB as well as the MEC server. In this paper, it is assumed that the SeNBs are connected to the MeNB in wired manner [Jafari, López-Pérez, Song et al. (2015)]. The set of small cells is denoted as $\mathcal{N} = \{1, 2, \ldots, N\}$, and we let $\mathcal{M} = \{1, 2, \ldots, M\}$ denotes the set of mobile user equipment and define that each single-antenna UE is associated with one SeNB. We assume that each UE has a computation-intensive and latency-sensitive task to be completed at each time slot t. Each UE can execute the computation task locally, or offload the computation task to the MEC server via the SeNB with which it is connected.

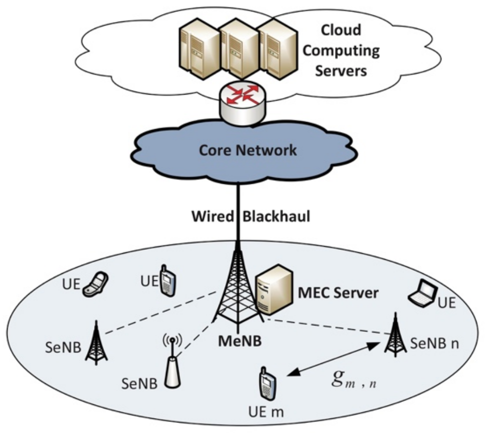

Figure 1: Network model

MEC server can handle all computing tasks because of its multi-tasking capability. Similar to many previous works in mobile cloud computing [Barbera, Kosta, Mei et al. (2013)] and mobile networking [Iosifidis, Gao, Huang et al. (2013)] to get tractable analysis, we consider a quasi-static scenario where the set of mobile users m remain unchanged during a computation offloading period (e.g., within several seconds), whereas may change during different periods. The network model is shown in Fig. 1.

2.2 Communication model

Because every SeNB is connected to the MEC server, UE can offload computation tasks to the MEC server through the SeNB with which it is connected. The computation offloading decision of UE m is denoted as $a_m \in \{0,1\}, \forall m$. Specifically, we set $a_m = 0$ when UE m decides to compute its task on its local equipment and $a_m = 1$ when UE decides to offload its computation task to MEC server by wireless manner. There are K orthogonal FDM (Frequency Division Multiplexing) sub-channels without interference to each other between UEs and SeNBs, and each sub-channel bandwidth is assumed as w. By given the computation offloading decision profile of all the UEs as $a = \{a_1, a_2, \ldots, a_m\}$, we describe the Signal to Interference plus Noise Ratio (SINR) $\gamma_m(t)$ and uplink data rate $r_m(t)$ of UE m at time slot t as

$$\gamma_m(t) = \frac{p_m(t)G_{m,n}(t)}{\sigma(t) + \sum_{i\in\mathcal{M}\setminus\{m\}:a_i=a_m} p_i(t)G_{i,n}(t)}, \tag{1}$$

$$r_m(t) = a_m(t)k_m(t)w\log_2\left(1+\gamma_m(t)\right), \tag{2}$$

where $k_m(t) \in [0,1,...,K]$ denotes the number of sub-channels allocated by SBS to users and $p_m(t)$ is transmission power of UE m. $G_{m,n}(t)$, $G_{i,n}(t)$ denote the channel gain between UE m and SeNB n, UE i and SeNB n. And $\sigma(t)$ is the additive white Gaussian noise. For the sake of simplicity, we omit (t) in the following expressions, e.g., r_m stands for $r_m(t)$, unless time slot t is emphasized.

2.3 Computation model

We consider each mobile user m has a computational task $J_m \overset{\Delta}{=} \left(B_m, D_m, T_m^{max}\right)$ at each time slot t, which can be computed either locally on the mobile user's equipment or remotely on the MEC server by computation offloading, as in Chen [Chen (2014)]. B_m (in KB) denotes the computation input data which including program codes and input parameters and D_m (in Megacycles) stands for the total number of CPU cycles to complete the computational task J_m. T_m^{max} stands for the maximum tolerable delay for executing the computation task J_m. A user can apply the methods in [Yang, Cao, Tang et al. (2012)] to obtain the information of B_m, D_m and T_m^{max}. Next we will discuss the overhead in terms of computation time and energy consumption for both local computation and MEC computation offloading cases.

2.3.1 Local computing

In this case, the computational task J_m is executed on local mobile equipment. $f_m^{(l)}$ represents the computational capacity (e.g., CPU cycles per second) of UE m. The situation is allowed here that different UEs may have different computational capabilities. The computational time executed locally by UE m is expressed as

$$T_m^{(l)} = \frac{D_m}{f_m^{(l)}}, \tag{3}$$

and the energy consumption for computation is given as

$$E_m^{(l)} = \rho_m D_m, \tag{4}$$

where ρ_m is the coefficient representing the energy consumed by each CPU cycle. According to the realistic measurements in Wen et al. [Wen, Zhang, Luo et al. (2012)], we set $\rho_m = 10^{-11}\left(f_m^{(l)}\right)^2$.

According to the computational time in Eq. (3) and energy consumption in Eq. (4), we can describe the total overheads of the local computation by UE m as

$$K_m^{(l)} = \lambda_m^{(\mathrm{t})} T_m^{(l)} + \lambda_m^{(\mathrm{e})} E_m^{(l)}, \tag{5}$$

where $\lambda_m^{(\mathrm{e})}$, $\lambda_m^{(\mathrm{e})}$ denote the weighting factor of computational time and energy consumption for mobile user's decision making, respectively. On account of modeling flexibility and meeting user-specific demands, it is considered that a user can take both computational time and energy consumption into decision making at the same time. For latency-sensitive task, we can increase the proportion of $\lambda_m^{(\mathrm{e})}$. While for energy-sensitive task, the value of $\lambda_m^{(\mathrm{e})}$ should be set high.

2.3.2 MEC server computing

We will state the case where the computational task J_m is offloaded to the MEC server in this section. UE m would generate the extra overhead of transmission time and energy consumption for offloading the computation input data to MEC server and downloading the computation outcome data to local equipment. The transmission time and energy consumption of UE m are computed respectively as

$$T_{m,off}^{(c)} = \frac{B_m}{r_m}, \tag{6}$$

$$E_{m,off}^{(c)} = \frac{p_m B_m}{r_m}. \tag{7}$$

When the computation input data are uploaded to the MEC server, MEC server will execute the computation task on behalf of UE. Let $f_m^{(c)}$ denotes the computational capability (i.e., CPU cycles per second) of the MEC server assigned to UE m. We assume that $\sum_{m \in M} f_m^{(c)} \leq f_c$ for computing resources allocated to all users can not exceed the computational capability of the MEC server f_c. So the computation time and corresponding energy consumption of the MEC server on task J_m are given as

$$T_{m,exe}^{(c)} = \frac{D_m}{f_m^{(c)}}, \tag{8}$$

$$E_{m,\text{exe}}^{(c)} = \frac{p_m D_m}{f_m^{(c)}}. \tag{9}$$

After the completion of the computing task executing by MEC server, the computation outcome data B_m^o needs to be transmitted back to the mobile users. Therefore, the downlink transmission time $T_{m,dow}^{(c)}$ and energy consumption $T_{m,dow}^{(c)}$ are given as

$$T_{m,dow}^{(c)} = \frac{\mathrm{B}_m^o}{r_m^d}, \tag{10}$$

$$E_{m,dow}^{(c)} = \frac{p_m^r \, \mathrm{B}_m^o}{r_m^d}, \tag{11}$$

where r_m^d is the download data rate of UE m, and p_m^r denotes the received power of UE m.

Similar to the study in Chen [Chen (2014)], the downlink transmission time and energy consumption for offloading computation outcome data from MEC server to UE m is not considered in this work. Owing to the size of computation outcome data is much smaller than size of computation input data and the download data rate is very high in general. According to the transmission time in Eq. (6), offloading energy consumption in Eq. (7) , computation time in Eq. (8) and executing energy consumption in Eq. (9), the time and energy consumption generated by offloading the computational task of UE m to MEC server as

$$T_{\mathrm{m}}^{(c)} = T_{m,off}^{(c)} + T_{m,exe}^{(c)}, \tag{12}$$

$$E_{\mathrm{m}}^{(c)} = E_{m,off}^{(c)} + E_{m,exe}^{(c)}, \tag{13}$$

we can compute the total overheads by offloading the computational task of UE m to MEC server as

$$K_m^{(c)} = \lambda_m^{(\mathrm{t})} T_m^{(c)} + \lambda_m^{(\mathrm{e})} E_m^{(c)}. \tag{14}$$

2.4 Problem formulation

Our goal is to optimize expected long-term utility performance of all users, and at each time slot t which is a decision step each user has only one task to perform. Specifically, we aim at minimizing the total overheads of all the users which can execute tasks on local mobile users' equipment or perform computation offloading with mobile edge computing. By minimizing the total overheads, users can make the overhead-aware optimal decision of computation offloading which of great importance in augmenting the computational efficiency and reducing the latency. We can model the optimization formulation of the problem as follows:

$$\min_{a} \sum_{t=0}^{T} \left[\sum_{m\in\mathcal{M}} \left(1-a_m\right) K_m^{(l)}\left(t\right) + \sum_{m\in\mathcal{M}} a_m K_m^{(c)}\left(t\right) \right] \tag{15}$$

subject to

$$a_m \in \{0,1\}, \forall m \in \mathcal{M}$$

$$\left(1-a_m\right) T_m^{(l)}\left(t\right) + a_m T_m^{(c)}\left(t\right) \le T_m^{\max}\left(t\right), \forall m \in \mathcal{M}$$

$$\sum_{m\in\mathcal{M}} f_m^{c}\left(t\right) \le f_c, \forall m \in \mathcal{M}$$

$$\sum_{m\in\mathcal{M}} k_m\left(t\right) w \le W, \forall m \in \mathcal{M}$$

The first term of Eq. (15) is the overheads generated by local computing and the second term is the overheads due to the computation offloading. The first constraint ensures that an overhead-aware solution can be obtained by finding the optimal values of the offloading decision profile a . The second constraint means that the delay for performing each calculation task cannot exceed the maximum tolerance delay. The third constraint manifests that the computing resources allocated to all users for offloading computation tasks cannot exceed the total amount of computing resources of the MEC server. And the last constraint specifies that the bandwidth allocated to all users cannot exceed the total spectrum bandwidth W . However, objective formula is difficult and impractical to solve due to the fact that a is binary variable, the feasible set of problem is non-convex and the optimization formulation is not a convex function. From another perspective, the problem can be viewed as sequence decision problem which need make continuous decisions to achieve the ultimate goal. In the following section, we will propose deep reinforcement learning algorithm to optimize the computation offloading problem.

3 Computation offloading algorithm based on deep RL

The reinforcement learning algorithm aims at solving the sequential decision problem and general sequential decision problems can be expressed in the framework of the Markov decision process (MDP). The MDP describes a stochastic decision process of an agent interacting with an environment or system. At each decision time, the system stays in a certain state s and the agent chooses an action a that is available at this state. After the action is performed, the agent receives an immediate reward R and the system transits to a new state s' according to the transition probability $P_{s,s'}^{a}$. The goal of an MDP or RL is to find an optimal policy which is a mapping from state to action to maximize or minimize a certain objective function [Alsheikh, Hoang, Niyato et al. (2015)].

3.1 Definitions using RL

To model this problem using RL, we set the following definitions:

Agent: the mobile user m who has computation-intensive and delay-sensitive tasks to complete.

State: $s_m = \left(\gamma_m, f_m^{(l)}\right)$ stands for the state of the agent m which constitutes SINR and computational capability of agent m. Let s_t denotes the system state at time slot t, where $s(t) = \left\{s_1(t), s_2(t), \ldots s_n(t)\right\}$.

Action: $a_m \in \{0,1\}$ where $a_m = 0$ represents for the UE m chooses to compute task on local equipment, while $a_m = 1$ means the UE m chooses to offload the computation task to MEC server. And $a_t = \left\{a_1(t), a_2(t), \ldots a_n(t)\right\}$ denotes the computation offloading decision profile of all UEs at time slot t.

Reward: The reward of all mobile users with computation tasks at time slot t denotes as $R_t(a) = -\sum_{m \in \mathcal{M}} (1 - a_m) K_m^{(l)}(t) - \sum_{m \in \mathcal{M}} a_m K_m^{(c)}(t)$. The first term denotes the minus of the total overheads of local computation by UE m, and second term denotes the minus of the total overheads of computation offloading of UE m with MEC.

An agent chooses an action a at a particular state s, and evaluates its performance in the form of state-value function based on the received immediate reward R and its estimate value of the state to which it is taken. After the convergence of state-value functions, it learns the optimal policy π^* judged by long-term discounted reward [Watkins and Dayan (1992)]. The discounted expected reward is defined by Bellman expectation equation as follows [Wei, Yu and Song (2010); Wei, Yu, Song et al. (2018)]

$$V(s) = R(s,a) + \gamma \sum_{s' \in s} P\left(s' \middle| s,a\right) V\left(s'\right), \tag{16}$$

where $R(s,a)$ is the immediate reward received by the agent when it selects action a at state s and $\gamma \in (0,1)$ is a discount factor, $P\left(s' \middle| s,a\right)$ is the transition probability from state s to s' when the agent chooses the action a. The discounted expected reward for taking action a includes the immediate reward and the future expected return.

According to the theory of Bellman's optimal equation, if we denote the $V^*(s)$ as the maximum total discounted expected reward at every state and it can be solved recursively by solving the following equation:

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in s} P\left(s' \middle| s,a\right) V\left(s'\right) \right], \tag{17}$$

then the optimal policy π^* can be obtained when the total discounted expected reward is maximum as follows:

$$\pi^* = \arg\max_a \left[R(s,a) + \gamma \sum_{s' \in s} P\left(s' \middle| s,a\right) V\left(s'\right) \right]. \tag{18}$$

However, the reward and probability are unknown in RL method which means it is a model-free based policy. For finite state MDP, action-value functions are usually stored in a lookup table and can be recursively learned. So we have to learn the Q-value which is defined as

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in s} P\left(s' \middle| s,a\right) V\left(s'\right). \tag{19}$$

The Q-value stands for the discounted expected reward for taking action a at state s and following policy π thereafter. The update of Q-values for an optimal policy π^* in conventional RL method Q-learning is performed as

$$Q^t = r + \gamma \max_a Q\left(s_{t+1}, a\right), \tag{20}$$

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + \alpha\left(Q^t - Q\left(s_t, a_t\right)\right). \tag{21}$$

where Q^t is the target value including current reward r and the maximum Q-value $\max_a Q\left(s_{t+1}, a\right)$ in next state and the $Q\left(s_t, a_t\right)$ is estimated value. $\alpha \in [0,1]$ is the learning rate.

3.2 Value function representation and approximation using DNN

In conventional RL method Q-learning, Q-table can be used to store the Q-value of each state action pair when the state and action spaces are discrete and the dimension is not high. However the state spaces are high-dimensional in our work, it's unrealistic to use Q-table mentioned in the previous section. Accordingly, function $Q_{\mathbf{w}}(s,a)$ is used to represent and approximate value function $Q\left(s,a\right)$ in RL to reduce the dimension in our work. Deep neural network has the advantage of extracting complex features in feature learning or representation learning [Bengio, Courville and Vincent (2013); Khatana, Narang and Thada (2018)], so we use DNN which is a nonlinear approximation to approximate the value function and improve the Q-learning method.

Deep reinforcement learning combines RL with deep learning (DL). The Q-value can be represented as $Q_{\mathbf{w}}(s,a)$ using DNN with two convolutional layers and two fully connected layers that are parameterized by a set of parameters $\mathbf{w} = \{w_1, w_2, \cdots, w_n\}$. Each hidden layer is composed of nonlinear analog neurons which can transform linear combination into an output value using non-linear activation functions (e.g., sigmoid, tanh, ReLU, etc). The output of the j th neuron in layer i can be formulated as

$$a_j^{(i)} = f_a(\mathbf{w}^{(i)} \cdot \mathbf{x}^{(i)} + b_j^{(i)}), \tag{22}$$

where $a_j^{(i)}$ is the output value in layer i, f_a is the activation function. $\mathbf{w}^{(i)} \cdot \mathbf{x}^{(i)} + b_j^{(i)}$ is the linear combination of input vector $\mathbf{x}^{(i)}$ with corresponding weights vector $\mathbf{w}^{(i)}$ and bias $b_j^{(i)}$ of neuron j in layer i. The DNN input is the original state, and the output $\mathbf{y}$ is the value function (Q-value) corresponding to each action,

$$\mathbf{y} = \mathbf{a}^{(\mathbf{i})} = \mathbf{f_a}(\mathbf{w}^{(\mathbf{i})} \cdot \mathbf{x}^{(\mathbf{i})} + \mathbf{b}^{(\mathbf{i})}). \tag{23}$$

The DNN can be trained to update the value function by updating the parameters w including weights $\mathbf{w}$ and biases $\mathbf{b}$. And the best fitting weights $\mathbf{W_{opt}}$ can be learned by iteratively minimizing the loss function $L(\mathbf{w})$, which is the mean-squared error (MSE) between the estimated value and the target value, i.e.,

$$Q_{\mathbf{w}}^t = r_t + \gamma \max_{a_{t+1}} Q_{\mathbf{w}}(s_{t+1}, a_{t+1}), \tag{24}$$

$$L(\mathbf{w}) = E\left[Q_{\mathbf{w}}^t - Q_{\mathbf{w}}(s_t, a_t)\right]^2, \tag{25}$$

where w are the parameters of the neural network, and $Q_{\mathbf{w}}^t$ is the target value. The error between the target value and the estimated value $Q_{\mathbf{w}}(s_t, a_t)$ is called temporal-difference (TD) error, denoted as $Q_{\mathbf{w}}^t - Q_{\mathbf{w}}(s_t, a_t)$.

Since DNN may cause the training of RL algorithm unstable and diverge due to the non-stationary targets and the correlations between samples. We adopt target network with fixed parameters $\mathbf{w}^-$ updated in a slower cycle, and experience replay which stores experience $\left\langle s_t, a_t, r_t, s_{t+1} \right\rangle$ in a replay buffer D and randomly sample a mini-batch of the experience to train the network, so the target value and loss function become:

$$Q_{\mathbf{w}^-}^t = r_t + \gamma \max_{a_{t+1}} Q_{\mathbf{w}^-}(s_{t+1}, a_{t+1}), \tag{26}$$

$$L(\mathbf{w}) = E_D\left[Q_{\mathbf{w}^-}^t - Q_{\mathbf{w}}(s_t, a_t)\right]^2, \tag{27}$$

where the parameters $\mathbf{w}$ used for approximating the estimated value updates at every step while the fixed parameters $\mathbf{w}^-$ for approximating target value updates at each fixed steps. Stochastic gradient descent method is applied to minimize the loss function, and the update of the parameters $\mathbf{w}$ defined as follows:

$$\omega_{t+1} = \omega_t + \alpha\left[Q_{\mathbf{w}^-}^t - Q_{\mathbf{w}}(s_t, a_t)\right]\nabla_{\omega} Q_{\mathbf{w}}(s_t, a_t), \tag{28}$$

where $\nabla_{\omega} Q_{\mathbf{w}}(s_t, a_t)$ is the gradient of $Q_{\mathbf{w}}(s_t, a_t)$.

The deep reinforcement learning algorithm performed by mobile users for computation offloading decision making is presented in Tab. 1. For each state, the agent chooses action randomly with the probability of $1-\varepsilon$ and with the probability of ε chooses the action with maximum action value function which called ε greedy strategy. When the agent performs the action in state s_t and receives the immediate reward r, it will observes the subsequent state s_{t+1} and approximate the action-value function $Q_{\mathbf{w}}(s_t, a_t)$ by DNN. After the convergence of action-value functions, each mobile user can select the overhead-aware optimal computation offloading action based on its state to minimize the total overheads of all users.

Table 1: Deep reinforcement learning based computation offloading algorithm

```
Initialization: experience replay buffer $D$, action-value function $Q$ with random parameters $\omega$, target network parameters $\omega^{-} = \omega$
1. For episode=1, $E_{\max}$ do
2.   Initialize state $s_0$ and reset reward $r = 0$
3.   For step=1, $T_{\max}$ do
4.     With the probability $1-\varepsilon$ choose a random action $a_t$
5.     Otherwise choose the action with the maximum action-value function $Q_{\mathbf{w}}(s_t, a_t)$
6.     Execute action $a_t$, receive an immediate reward $r$ and observe the next state $s_{t+1}$
7.     Replay buffer stores the tuple $\langle s_t, a_t, r_t, s_{t+1} \rangle$ in $D$
8.     Sample a random mini-batch of tuple $\langle s_t, a_t, r_t, s_{t+1} \rangle$ from $D$
9.     $Q_{\mathbf{w}^{-}}^{t} = \begin{cases} r_t & t = T_{\max} \\ r_t + \gamma \max_{a_{t+1}} Q_{\mathbf{w}^{-}}(s_{t+1}, a_{t+1}) & t < T_{\max} \end{cases}$
10.    Perform a gradient descent on $\left(Q_{\mathbf{w}^{-}}^{t} - Q_{\mathbf{w}}(s_t, a_t)\right)^2$ with the parameter $\omega$
11.    Every C steps, update the target network parameters $\omega^{-} = \omega$
12.  End For
13. End For
```

4 Simulation results and discussion

In this section, we assess the performance of the proposed deep RL based computation offloading decision method compared with two baseline schemes. Simulation scenarios are presented that there are 10 small cells randomly deployed. The transmission power of UE m is set to be $p_m = 100$ mWatts. The spectrum bandwidth is set as $W = 10$ MHZ, while the additive white Gaussian noise $\sigma = -100$ dBm. The channel gain model presented in 3GPP standardization is adopted here. We applied the face recognition as the computation task here [Soyata, Muraleedharan, Funai et al. (2012)]. The size of computation input data B_m (KB) and the total number of CPU cycles D_m (Megacycles) randomly distributed in the range $[1000,\ 10000]$. The computational capability $f_m^{(l)}$ of a mobile user m is assigned from the set $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ GHz at random which reveals the heterogeneity of mobile user's computational capability. The total computational capability of the MEC server is $f_c = 100$ GHz. We assume that the weighting factor of computation time as $\lambda_m^{(t)} = 0.5$ and $\lambda_m^{(e)}$ for energy weighting factor correspondingly.

Firstly, we demonstrate the convergence of the proposed deep RL algorithm. Fig. 2 shows the total rewards of all UEs at every episode with different learning rates. As we can see, the proposed learning strategy with learning rate of 0.01 obtain the reward per episode fluctuates around -400 after 1000 episodes, while algorithm with learning rate of 0.001 and 0.0001 obtain the rewards per episode fluctuates around -500 and -450 after 1000 episodes. As expected, different learning rates result in different convergence performance, and the algorithm with 0.01 learning rate outperforms compared with other learning rates. The fluctuation of the curve after algorithm converges is for the ε greedy strategy adopted here which users not always choose the action with maximum action value function and have the possibility to choose action randomly.

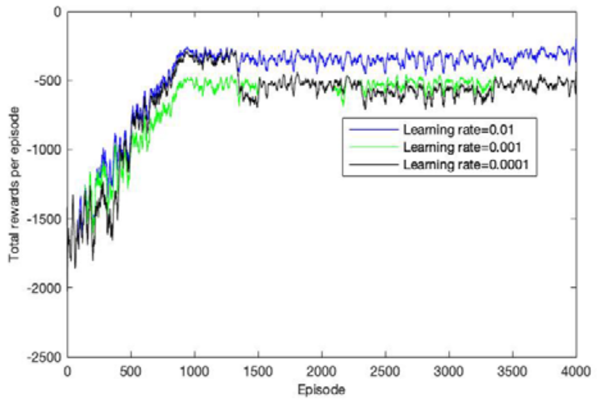

Figure 2: The total rewards of each episode with different learning rates

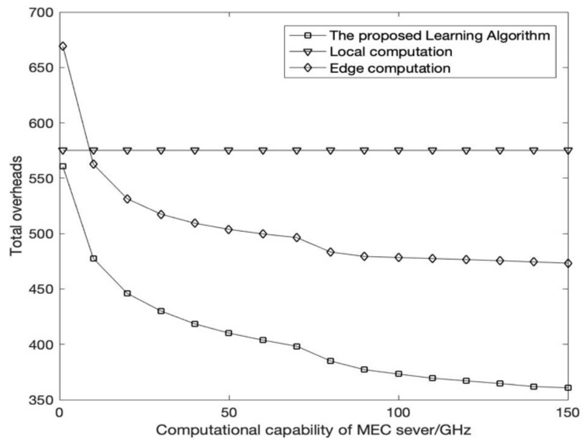

Figure 3: Computational capability versus total overheads with different schemes

We will show the performance of proposed scheme in comparison with baseline methods, including the local computation policy, which executes all the computational task on local mobile user's equipment, and the edge computation policy which offloads all the tasks of UEs to the MEC server for edge computing. Fig. 3 demonstrates that with the increase of computational capability, the total overheads for edge computation policy and proposed learning algorithm decreases due to the change of MEC server's computational capability will influence the computation offloading policy of mobile users. With the increasing computational capability of MEC server, edge computation strategy performs better than local computation due to its multi-tasking capability. However, baseline methods are not effective than proposed learning method on account of proposed method can obtain the optimal overhead aware policy according to its learning result.

Fig. 4 shows the relationship between the number of mobile users and the total overheads of all the mobile users. The total overheads increase gradually with the number of users grows. The overhead generated by edge computation method is less than overhead of local computation method gradually due to the increasing of number of users with more computation tasks to execute. While local computation policy consumes more time and energy than the baseline schemes on account of the limited computational capability when the number of users increases. Contrasted with baseline methods, the proposed learning algorithm always obtains the minimum overhead which means the proposed scheme can achieve the optimal computation offloading decision for reducing the latency, energy consumption and improving the efficiency.

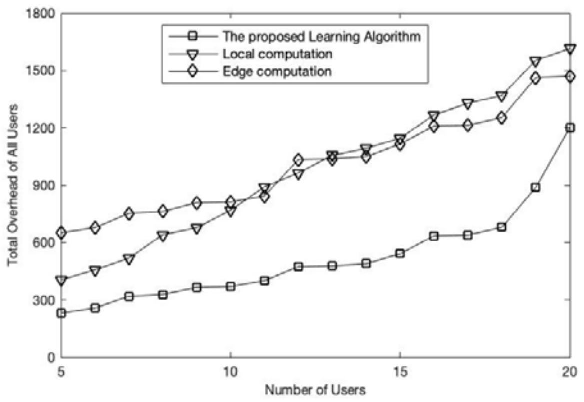

Figure 4: Number of mobile users versus total overheads with different schemes

The assignment of weighting factor which will represent different states of users. Mobile user which is sensitive to delay will take more proportion of $\lambda_m^{(t)}$ into account while user in low-battery state will consider more proportion of $\lambda_m^{(e)}$ for the overhead computation. Fig. 5 presents that when the weighting factor of time increases from 0 to 1 (while the proportion of energy decreases from 1 to 0 accordingly), total overheads rise due to the fact that the computational and transmission time occupy more proportion in total overheads. As we can see from the above results, the decision-making performance of the proposed learning algorithm performances better than baseline methods in terms of total overheads of all the mobile users.

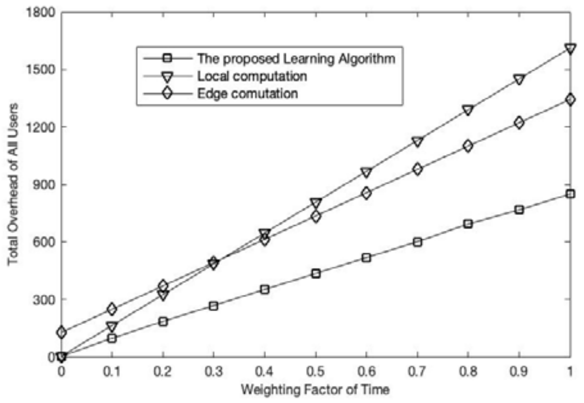

Figure 5: Weighting factor of time versus total overheads with different schemes

5 Conclusion

In this paper, we propose a deep reinforcement learning approach for computation offloading decision issue with mobile edge computing. The problem is formulated as minimizing the total overheads of all the users which can execute tasks on local mobile users' device or offload the computation to MEC server. In order to solve this problem, we apply deep neural network in RL framework to approximate action-value action and obtain the overhead-aware optimal computation offloading strategy based on deep Q-learning method. The performance evaluation of proposed method is compared with two baseline methods. Simulation results showed that the proposed policy can achieve better performance than baseline methods in terms of total overheads which reduces the latency, energy consumption and enhances the computation efficiency.

Acknowledgement: This work was supported by the National Natural Science Foundation of China (61571059 and 61871058).

References

Alsheikh, M. A.; Hoang, D. T.; Niyato, D.; Tan, H. P.; Lin, S. (2015): Markov decision processes with applications in wireless sensor networks: a survey. *IEEE Communications Surveys* & *Tutorials*, vol. 17, no. 3, pp. 1239-1267.

Bao, L.; Ding, S. (2016): Cloud computing: a new computing paradigm. *International Journal of High Performance Computing and Networking*, vol. 1, no. 3, pp. 167.

Barbera, M. V.; Kosta, S.; Mei, A.; Stefa, J. (2013): To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. *Proceedings IEEE INFOCOM*, pp. 1285-1293.

Bengio, Y.; Courville, A.; Vincent, P. (2013): Representation learning: a review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798-1828.

Chen, X. (2014): Decentralized computation offloading game for mobile cloud computing. *IEEE Transactions on Parallel* & *Distributed Systems*, vol. 26, no. 4, pp.

974-983.

Patel, M.; Naughton, B.; Chan, C.; Sprecher, N.; Abeta, S. et al. (2014): Mobile-edge computing-Introductory technical white paper. *ETSI White Paper*.

Huang, D.; Wang, P.; Niyato, D. (2012): A dynamic offloading algorithm for mobile computing. *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 1991-1995.

Iosifidis, G.; Gao, L.; Huang, J.; Tassiulas, L. (2013): An iterative double auction for mobile data offloading. *International Symposium on Modeling* & *Optimization in Mobile, Ad Hoc* & *Wireless Networks*, pp. 154-161.

Jafari, A. H.; López-Pérez, D; Song, H.; Claussen, H.; Ho, L.; Zhang, J. (2015): Small cell backhaul: challenges and prospective solutions. *Eurasip Journal on Wireless Communications* & *Networking*, vol. 2015, no. 1, pp. 206.

Khatana, A.; Narang, V.; Thada, V. (2018): A review of optimization methods in deep learning. *International Journal of Computer Sciences and Engineering*, vol. 6, no. 4, pp. 440-447.

Kumar, K.; Liu, J.; Lu, Y. H.; Bhargava, B. (2013): A survey of computation offloading for mobile systems. *Mobile Networks* & *Applications*, vol. 18, pp. 129-140.

Liu, J.; Mao, Y.; Zhang, J.; Letaief, K. B. (2016): Delay-optimal computation task scheduling for mobile-edge computing systems. *IEEE International Symposium on Information Theory*, pp. 1451-1455.

Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K. B. (2017): A survey on mobile edge computing: the communication perspective. *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322-2358.

Mao, Y.; Zhang, J.; Letaief, K. B. (2016): Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590-3605.

Mao, Y.; Zhang, J.; Song, S. H.; Letaief, K. B. (2017): Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems. *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994-6009.

Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. (2016): Edge computing: vision and challenges. *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637-646.

Soyata, T.; Muraleedharan, R.; Funai, C.; Kwon, M.; Heinzelman, W. (2012): Cloud-vision: real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. *Computers and Communications*.

Wang, C.; Liang, C.; Yu, F. R.; Chen, Q.; Tang, L. (2017): Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 4924-4938.

Watkins, C. J. C. H.; Dayan, P. (1992): Technical note: Q-learning. *Machine Learning*, vol. 8, no. 3-4, pp. 279-292.

Wei, Y.; Yu, F. R.; Song, M. (2010): Distributed optimal relay selection in wireless cooperative networks with finite-state markov channels. *IEEE Transactions on Vehicular Technology*, vol. 59, no. 5, pp. 2149-2158.

Wei, Y.; Yu, F. R.; Song, M.; Han, Z. (2018): User scheduling and resource allocation in hetnets with hybrid energy supply: an actor-critic reinforcement learning approach. *IEEE Transactions on Wireless Communications*, vol.17, no. 1, pp. 680-692.

Wen, Y.; Zhang, W.; Luo, H. (2012): Energy-optimal mobile application execution: taming resource-poor mobile devices with cloud clones. *Proceedings IEEE INFOCOM*, pp. 2716-2720.

Xu, J.; Ren, S. (2017): Online learning for offloading and autoscaling in renewable-powered mobile edge computing. *Global Communications Conference*, pp. 1-6.

Yang, L.; Cao, J.; Tang, S.; Li, T.; Chan, A. T. S. (2012): A framework for partitioning and execution of data stream applications in mobile cloud computing. *IEEE International Conference on Cloud Computing*, pp. 23-32.

Yu, Y.; Zhang, J.; Letaief, K. B. (2016): Joint subcarrier and CPU time allocation for mobile edge computing. *Global Communications Conference*, pp. 1-6.