Deep Reinforcement Learning-Based Task Offloading and Resource Allocation for Mobile Edge Computing

Liang Huang, Xu Feng(✉), Liping Qian, and Yuan Wu

College of Information Engineering, Zhejiang University of Technology, Hangzhou 310023, China
{lianghuang,lpqian,iewuy}@zjut.edu.cn, xfeng_zjut@163.com

Abstract. We consider a mobile edge computing system that every user has multiple tasks being offloaded to edge server via wireless networks. Our goal is to acquire a satisfactory task offloading and resource allocation decision for each user so as to minimize energy consumption and delay. In this paper, we propose a deep reinforcement learning-based approach to solve joint task offloading and resource allocation problems. Simulation results show that the proposed deep Q-learning-based algorithm can achieve near-optimal performance.

Keywords: Mobile edge computing · Deep reinforcement learning
Task offloading · Resource allocation · Deep Q-learning

1 Introduction

Mobile edge computing (MEC) provides a distributed computing environment for mobile users [1], such that users are able to offload their computing tasks to MEC servers. When tasks are offloaded, users can obtain higher quality of service (QoS) the adequate cloud resources. But when users offload their tasks to the cloud, the quality of service will be interfered because of the existence of communication delay.

Offloading data to the MEC system is convenient for mobile devices. [2] investigated data offloading from mobile devices to MEC system and proposes a coalitional game-based pricing scheme. A reformulation-linearization-technique-based branch-and-bound (RLTBB) method is proposed in [3] to minimize the energy consumption on mobile devices. When there are two mobile devices energized by wireless power transfer (WPT) in MEC system, [4] minimized the total transmit energy of access point (AP) by a two-phase method. Multi-user MEC systems with one task per user are addressed in [5] and [6]. [7] proposed an advanced algorithm multi-user multi-task offloading (MUMTO) to solve joint offloading decision and resource allocation in multi-user multi-tasking mobile edge computing system.

To minimize the weighted sum of the costs of computation and communication energies as well as transmission and processing delays, we concentrate on searching the optimal offloading decision and resource allocation for all users. Specifically, we propose a deep reinforcement approach, which is an enhanced version of Q-learning. We use deep Q network on behalf of Q value-action function [8]. Deep Q network has already been also used in the cache-enabled opportunistic IA wireless networks [9]. With this advanced reinforcement learning algorithm, the almost optimal binary offloading decision and resource allocation will be found all at once. Simulation results show that the reinforcement learning-based algorithm obtains the almost optimal offloading decision and resource allocation under varieties of parameter settings.

The rest of this paper is organized as follows. In Sect. 2, system model and problem formulation are presented. Deep Q network algorithm is presented in Sect. 3. In Sect. 4, simulation results are discussed. Finally, we present conclusions in Sect. 5.

2 System Model and Problem Formulation

2.1 Mobile Edge Offloading

We consider an ordinary cloud access network composed by one AP, one edge server and N mobile users, where each user has M independent tasks. The AP is wired to the edge server, and mobile users are connected to the edge server by wireless channels. The system model is shown in Fig. 1. Every task of each user can be processed locally or offloaded to the edge server. Let x_{nm} denote the offloading decision of task m of user n, where $x_{nm} = 0$ denotes that task m of user n is processed locally and $x_{nm} = 1$ denotes that task m of user n is offloaded.

2.2 Cost of Remote Processing

The input data size of task m of user n is denoted by L^{d}_{nm} while the output data size is denoted by L^{u}_{nm}. When mobile users offload their tasks to the edge server, the energy consumption of task m of user n is divided into two parts, data transmission and receiving, which are denoted by E^{t}_{nm} and E^{r}_{nm}, respectively. For the wireless transmission among mobile users and the edge server, we denote the uplink and downlink bandwidths assigned to user n by C^{u}_{n} and C^{d}_{n}. The uplink transmission time is denoted by $T^{\mathrm{u}}_{nm} = L^{\mathrm{u}}_{nm}/C^{\mathrm{u}}_{n}$, accordingly, and the downlink transmission time is denoted by $T^{\mathrm{d}}_{nm} = L^{\mathrm{d}}_{nm}/C^{\mathrm{d}}_{n}$. The values of C^{u}_{n} and C^{d}_{n} are limited by the abilities of the corresponding wireless links, and the total uplink and downlink bandwidths are denoted by C^{U} and C^{D}, respectively.

The cloud processing time is denoted by

$$T^{\mathrm{C}}_{nm} = \frac{L^{\mathrm{d}}_{nm} N^{\mathrm{C}}_{nm}}{f^{\mathrm{c}}} \tag{1}$$

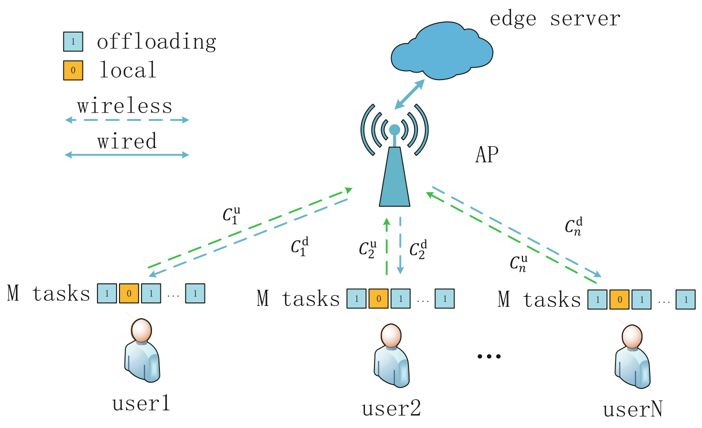

Fig. 1. System model

The cloud processing rate f^c is pre-fixed for every user. The size of T_{nm}^{C} only lies on the size of each task of mobile users. In this paper, N_{nm}^{C} denotes the number of processing cycles for every input data. When some or all tasks of user n are offloaded to the remote cloud, the system utility cost is denoted by C_{nm}^{c}. Because of the overlaps in the time of communication and processing, for making it acceptable, the offloading decision is set to be the offloading delay. Thus, we first provide the worst-case delay formulation:

$$T_n^{W} \triangleq \sum_{m=1}^{M} \left(T_{nm}^{u} + T_{nm}^{d} + T_{nm}^{C}\right) x_{nm}, \forall n, \tag{2}$$

which is a summation of all delays of transmission and processing.

2.3 Cost of Local Processing

When task m of user n is decided to be processed locally, the corresponding energy consumption per data bit is denoted by e_l. Therefore, the energy consumption of task m of user n is denoted by $E_{nm}^{l} = L_{nm}^{d} e_l$, where the local processing time per data bit is denoted by t_l. Thus, the processing time consumption of task m of user n is denoted by $T_{nm}^{l} = (L_{nm}^{d} + L_{nm}^{u}) t_l$.

2.4 Problem Formulation

We aim to reduce the total energy cost and keep the quality of service of every user at the same time. Our objective is to minimize the weighted sum of the costs of computation and communication energies as well as transmission and processing delays. The optimization problem is given as follows:

$$\min_{x_{nm}, C_n^{\mathrm{u}}, C_n^{\mathrm{d}}} \sum_{n=1}^{N} \left[\sum_{m=1}^{M} (E_{nm}^{\mathrm{l}}(1 - x_{nm}) + E_{nm}^{\mathrm{c}} x_{nm}) + \omega_n \max\{T_n^{\mathrm{L}}, T_n^{\mathrm{W}}\} \right], \tag{3}$$

$$\text{subject to:} \quad \sum_{n=1}^{N} C_n^{\mathrm{u}} \leq C^{\mathrm{U}}, \tag{4}$$

$$\sum_{n=1}^{N} C_n^{\mathrm{d}} \leq C^{\mathrm{D}}, \tag{5}$$

$$C_n^{\mathrm{u}}, C_n^{\mathrm{d}}, \geq 0, \forall n, \tag{6}$$

$$x_{nm} \in \{0, 1\}, \forall n, m, \tag{7}$$

where

$$E_{nm}^{\mathrm{c}} \triangleq (E_{nm}^{\mathrm{t}} + E_{nm}^{\mathrm{r}} + \lambda C_{nm}^{\mathrm{c}}), \tag{8}$$

and

$$T_n^{\mathrm{L}} \triangleq \sum_{m=1}^{M} T_{nm}^{\mathrm{l}}(1 - x_{nm}). \tag{9}$$

In the formula above, E_{nm}^{c} is the weighted sum of transmission energy, receiving energy, and system utility cost when task m of user n is offloaded to the remote cloud. λ is the corresponding weight, T_n^{L} is the processing delay when task m of user n is processed locally, and ω_n is the weight between energy consumption and processing delay in the whole system cost. The uplink and downlink bandwidth are limited by constraints (2) and (3).

It is difficult to solve the mixed-integer programming problem in (3) in a general way. In the next section, we propose a reinforcement learning algorithm based on deep Q network.

3 Deep Reinforcement Learning

3.1 Deep Q Network

Reinforcement learning is a machine learning algorithm, composed by an environment and an agent. The agent is a brain, which selects the most appropriate action from the state given by the environment. The role of the environment is to generate the state of the next step and the reward generated after the action is taken by the agent. Then, the agent updates its parameters according to the reward, so as to improve its prediction accuracy. Deep Q network is an advanced reinforcement learning algorithm, which is evolved from Q-learning, by replacing the Q table of Q-learning with a deep neural network. It can solve complex problems with lager state space. The Q-function is updated as:

$$Q^{\theta}(s_t, a) = r(s_t, a) + \gamma \max Q'(s_{t+1}, a'; \theta'), \tag{10}$$

where s_t is the system state at time shot t, and $r(s_t, a)$ is the reward after the agent choices action a at state s_t. $Q^{\theta}(s_t, a)$ is the Q value under selecting action a at state s_t. The larger $Q^{\theta}(s_t, a)$ is, the more reward to taking action a in its current state s_t.

3.2 Formulation of the Network's Optimization Problem

In our system, there are N mobile users, each with M tasks, and each task can be offloaded to the remote server. Each user's task size is different. When the task is offloaded to the remote server, the corresponding speed may be different, and there is a limit to their uplink and downlink direction. To minimize energy consumption, we use deep Q network to find the almost optimal offloading decision x_{nm} of task m of user n as well as resource allocation C_n^{u} and C_n^{d} of user n. x_{nm}, C_n^{u} and C_n^{d} are programmed into the system state as input for Q network. The output of the Q network is the Q value of the corresponding action. Each time the agent chooses the appropriate action with respect to the Q value. The execution result of the action is to make corresponding adjustment to offloading decision x_{nm} and resource allocation C_n^{u} and C_n^{d}.

In deep Q network, there are two networks, specifically, an evaluation network and a target network. Their inputs are the current state and the next state after the action execution respectively. The output of the target network that has been modified by reward will be used as a label of evaluation network. Then, a gradient descent algorithm is applied to the error between them. By continuously reducing the error and updating the parameters of the evaluation network, it can predict more accurately. The structures of the evaluation network and the target network are completely consistent but with different parameters. After every other N cycles, all the parameters of evaluation network will be assigned to the target network to cut off the relevance between them.

There is a replay memory structure in the deep Q network. The system records a memory (s_t, a_t, r_t, s_{t+1}) every step after performing an action. During the network training, a small batch of memory is extracted from the memory pool, so that the Q network can learn the previous experience. An ε-greedy policy is used to determine the intensity of exploration and learning, i.e., how likely it is to take advantage of existing knowledge or to try new actions. The training process of deep Q network is described in Algorithm 1.

In order to get the optimal offloading decision and resource allocation, we need to appropriately design the system state, action and reward mechanism, which are described below:

System State. The current state of the system $x(t)$ represents the location of the agent, determined by the state of N users and their M tasks. The system state at time slot t is defined as,

$$\begin{aligned} s(t) &= \{x_{11}(t), x_{12}(t) \ldots x_{ij}(t) \ldots x_{NM}(t), C_1^{\mathrm{u}}(t), C_1^{\mathrm{d}}(t) \ldots C_l^{\mathrm{u}}(t), \\ &\quad C_l^{\mathrm{d}}(t) \ldots C_N^{\mathrm{u}}(t), C_N^{\mathrm{d}}(t)\} \end{aligned} \tag{11}$$

The system state is consisted of two parts, offloading decision x_{nm} of task m of user n and resource allocation C_n^{u} and C_n^{d} of user n. The offloading decision $x_{ij}(t) \in \{0, 1\}$, where $i = 1, 2, \ldots N$, $j = 1, 2, \ldots M$. The subscript l stands for the lth user, and $l = 1, 2, \ldots N$. The number of system states is proportional to the number of users. When the number of system states is very large, the advantage of replacing Q-function with Q network emerges.

System Action. Each action of the system determines which of the user's tasks are processed in the remote server, and what are their speed assignments when they are processed in the remote server. The selected action $a(t)$ of the agent is denoted by

$$a(t) = \{a_1(t), a_2(t) \ldots a_k(t) \ldots a_{NM}(t), a_1(t), a_2(t) \ldots a_g(t) \ldots a_{2N}(t)\} \quad (12)$$

We first pull all the tasks in the system into one dimension. $a_k(t)$ represents the decision of the kth task i.e., $a_k(t) = 1$ represents processing in the remote server, and $a_k(t) = 0$ represents local processing. $a_g(t)$ represents the speed distribution when processing in the remote server. Speed adjustment is achieved by stride, they meet the condition $\sum_{g=1}^{N} a_g(t) \leq C_U$ when the action adjusts the uplink speed. This restriction also occurs when the downlink speed is adjusted.

Reward Function. System rewards represent our optimization goals. We save the calculation result S_{t-1} of formula (4) with the current parameter, before executing the action. After executing the action, we get the latest calculation result S_t, if S_t is smaller than S_{t-1} we give a positive reward $r(t) = +1$, if S_t is bigger than S_{t-1} we give a negative reward $r(t) = -1$, otherwise, $r(t) = 0$. This allows the agent to constantly search the optimal offloading decision and resource allocation to minimize total energy consumption.

4 Performance Evaluation

We use TensorFlow to evaluate the performance of deep reinforcement learning. Then, Matlab is used to demonstrate greedy algorithm performance. We assume that the number of mobile users $N = 5$, and the number of tasks for each user $M = 4$. Detailed parameters for reinforcement learning are listed in Table 1. We set the local computation time of the mobile device as 4.75×10^{-7} s/bit, and processing energy consumption as 3.25×10^{-7} J/bit. We assume that the input data size of all tasks is randomly distributed between 10MB and 30MB, and the output data size is randomly distributed between 1MB and 3MB.

In the simulations, we set both the uplink bandwidth and the downlink bandwidth limit between the user and the edge server as 150 Mbps. The receiving energy consumptions and transmission energy consumptions of the mobile device are both 1.42×10^{-7} J/bit. The CPU rate of remote cloud sever is 10×10^9

Table 1. Parameter values used in the simulations

Parameter	Value	Description
Episode	4000	Number of main cycles
Replay memory size	2000	The size of memory pool
Frequency of learning	5	How often the training step is performed
Mini-batch size	32	How many memories are used for each training step
Learning rate	0.0001	The learning rate of Adam optimizer
Reward decay	0.9	The degree of emphasis of previous experience
ε-greedy increment	0.005	The growth rate of ε-greedy at every training step
Max ε-greedy	0.9	The maximum of ε-greedy
Target network update frequency	50	How many steps the target network is updated
Pre-training steps	200	How many memories are stored before the training begins

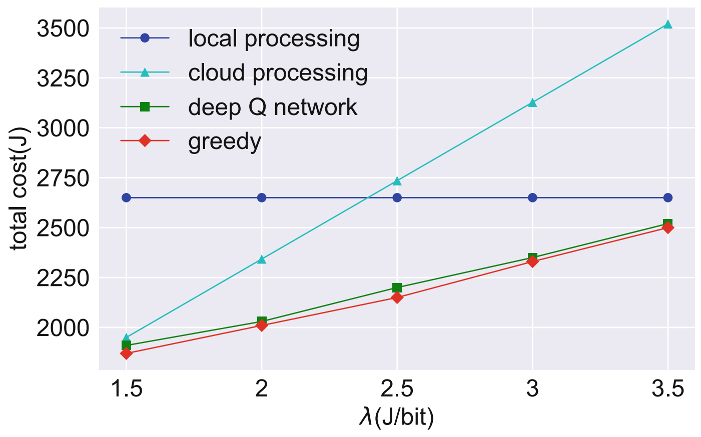

Fig. 2. The total cost under different λ (J/bit)

cycle/s. When a mobile user's task is offloaded to the cloud, the system utility cost is denoted by

$$C_n^{\mathrm{c}} = D_n^{\mathrm{d}} + \frac{\varphi_1}{f^{\mathrm{c}}} + \frac{\varphi_2}{C^{\mathrm{U}}} + \frac{\varphi_3}{C^{\mathrm{D}}} \tag{13}$$

where $\varphi_1 = 10^{18}$bit$\times$cycle/s and $\varphi_2 = \varphi_3 = 10^{16}bit\times$bps. We further set $\lambda = 2.5 \times 10^{-7}$ J/bit, and $\omega_n = 1$ J/s refer to [7].

We compare the proposed algorithm with local processing only algorithm, cloud processing only algorithm, and greedy algorithm. The local processing only

Algorithm 1. Deep reinforcement learning algorithm in task offloading and resource allocation

```
1: **Initialization:**
2:   Initialize the evaluation and target Q network parameters with $\theta$.
3:   Initialize replay memory.
4: **for** episode $k \leq 1, 2, \ldots, K$ **do**
5:    **if** $mod(k, 100) == 0$ **then**
6:       Change the initialization to the current best result.
7:    **end if**
8:    Choose a random probability number $p$.
9:    **if** $p < \varepsilon$ **then**
10:      $a^*(t) = arg \max_a Q(s, a; \theta)$.
11:   **else**
12:      Choose $a(t)$ randomly.
13:   **end if**
14:   Calculate $S_t$ according to (3)
15:   **if** $S_t < S_{t+1}$ **then**
16:      Set $r(t) = 1$
17:   **else if** $S_t > S_{t+1}$ **then**
18:      Set $r(t) = -1$
19:   **else**
20:      Set $r(t) = 0$
21:   **end if**
22:   Get the reward $r(t)$ and next state $s(t+1)$ after execute $a(t)$.
23:   Save this memory formed as $(s(t), a(t), r(t), s(t+1))$ in the replay memory.
24:   Extract a min-batch of memories from the replay memory.
25:   Calculate the target Q-value $y(t)$ from the target deep-Q network,
            $y(t) = r(t) + \gamma \max_{a'} \hat{Q}(s(t+1), a'; \theta^-)$.
26:   Perform gradient descent algorithm to minimize $(y(t) - Q(s(t), a(t); \theta))^2$.
27:   Update the parameters $\theta$ of the evaluation network.
28:   Copy the parameters of the evaluation network to the target network, every $S$
      step.
29: **end for**
```

method means that all user tasks are processed locally. The cloud processing only method processes all user tasks in the cloud. The greedy algorithm means that all the offloading decision combinations are enumerated to select the optimal policy. Greedy method is time-consuming, but the optimal solution can be found. In our simulations, each simulation result is obtained through 100 repetitions, but the data size of each input and output is randomly generated.

The system total cost under different weights λ is shown in Fig. 2. The deep reinforcement learning algorithm can get the almost optimal solution, where the gap between the deep reinforcement learning and greedy algorithm is very small. As λ increases, all user tasks tend to be processed locally.

The performance of the system cost with different learning rate is plotted in Fig. 3. Convergence process is faster when the learning rate is 0.001, compared

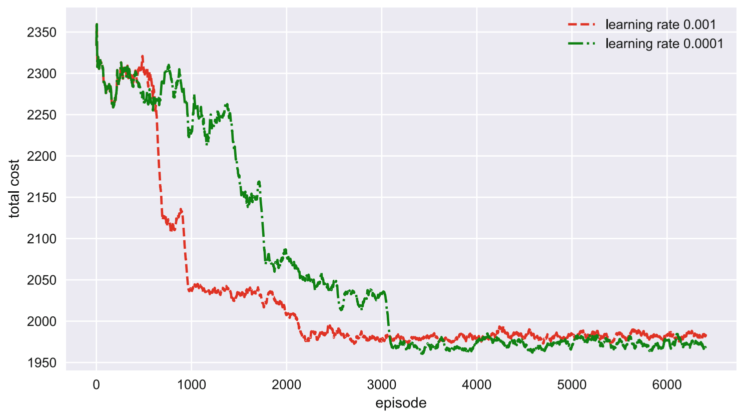

Fig. 3. Convergence performance under different learning rate

to the case when the learning rate is 0.0001. However, when the learning rate increases, it is more possible to find the local optimal solution instead of the global optimal. Hence, we need to choose an appropriate learning rate with respect to dedicated situations.

5 Conclusion

In a mobile edge computing system that every user has multiple tasks being offloaded to edge server via wireless networks, all users occupy the common communication resource when their tasks are offloaded. In order to minimize the total consumption of computation and communication energies cost, and transmission delays between mobile users and AP as well as processing delays on local devices and could, we use deep reinforcement learning to find the near optimal offloading decision and resource allocation. Compared with greedy algorithm, reinforcement learning method can find an almost optimal solution.

Acknowledgments. This work was supported in part by the National Natural Science Foundation of China under Grant 61572440 and Grant 61502428, in part by the Zhejiang Provincial Natural Science Foundation of China under Grants LR17F010002 and LR16F010003, in part by the Young Talent Cultivation Project of Zhejiang Association for Science and Technology under Grant 2016YCGC011.

References

1. Liu, J., Zhang, Q.: Offloading schemes in mobile edge computing for ultra-reliable low latency communications. IEEE Access **6**, 12825–12837 (2018)
2. Zhang, T.: Data offloading in mobile edge computing: a coalition and pricing based approach. IEEE Access **6**, 2760–2767 (2018)

3. Zhao, P., Tian, H., Qin, C., Nie, G.: Energy-saving offloading by jointly allocating radio and computational resources for mobile edge computing. IEEE Access **5**, 11255–11268 (2017)
4. Hu, X., Wong, K.K., Yang, K.: Wireless powered cooperation-assisted mobile edge computing. IEEE Trans. Wirel. Commun. **17**(4), 2375–2388 (2018)
5. Chen, X.: Decentralized computation offloading game for mobile cloud computing. IEEE Trans. Parallel Distrib. Syst. **26**, 974–983 (2015)
6. Meskar, E., Todd, T., Zhao, D., Karakostas, G.: Energy efficient offloading for competing users on a shared communication channel. In: Proceedings of IEEE International Conference on Communications (ICC), pp. 3192–3197 (2015)
7. Chen, M.H., Liang, B., Dong, M.: Joint offloading decision and resource allocation for multi-user multi-task mobile cloud. In: 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, pp. 1–6 (2016)
8. Mnih, V., Kavukcuoglu, K., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
9. He, Y., et al.: Deep-reinforcement-learning-based optimization for cache-enabled opportunistic interference alignment wireless networks. IEEE Trans. Veh. Technol. **66**(11), 10433–10445 (2017)