Mobile Networks and Applications
https://doi.org/10.1007/s11036-018-1177-x

Distributed Deep Learning-based Offloading for Mobile Edge Computing Networks

Liang Huang[1] · Xu Feng[1] · Anqi Feng[1] · Yupin Huang[1] · Li Ping Qian[1]

Abstract

This paper studies mobile edge computing (MEC) networks where multiple wireless devices (WDs) choose to offload their computation tasks to an edge server. To conserve energy and maintain quality of service for WDs, the optimization of joint offloading decision and bandwidth allocation is formulated as a mixed integer programming problem. However, the problem is computationally limited by the curse of dimensionality, which cannot be solved by general optimization tools in an effective and efficient way, especially for large-scale WDs. In this paper, we propose a distributed deep learning-based offloading (DDLO) algorithm for MEC networks, where multiple parallel DNNs are used to generate offloading decisions. We adopt a shared replay memory to store newly generated offloading decisions which are further to train and improve all DNNs. Extensive numerical results show that the proposed DDLO algorithm can generate near-optimal offloading decisions in less than one second.

Keywords Mobile edge computing · Offloading · Deep learning · Distributed learning

1 Introduction

With the development of wireless communication technology, transmitting tremendous computation tasks from wireless devices to nearby access points or base stations is possible, which triggers meaningful cloud-computing applications, e.g., online gaming, virtual/augmented reality, and real-time media streaming. By deploying computation servers at user side and avoiding backhauling traffic generated by applications to a remote data center, mobile edge computing (MEC) [1–3] provides an efficient approach to bridge user and edge server. It reduces the delay in executing the computation tasks and saves the energy consumption for those delay-sensitive cloud-computing applications.

The decision whether a WD offloads its computation task to an MEC server or not should be carefully studied. If computation tasks are aggressively offloaded to the edge server, a severe congestion will occur on the uplink wireless channels, which leads to a significant delay in executing computation tasks. Therefore, to exploit the computation offloading, we need a joint management for offloading decisions and the associated radio bandwidth allocation. However, due to the binary property of offloading decisions, directly enumerating all those possible solutions is computationally prohibitive.

Different low-complexity algorithms are proposed to solve the binary computation offloading problem in the literature [4–12]. A distributed algorithm based on game theory is proposed for MEC system in [4], which requires multiple iterations of communications between the edge server and WDs. Another iterative algorithm to solve joint task offloading and resource allocation in MEC networks is to iteratively update the binary offloading decision [5, 6], where the traditional resource allocation problem is solved when the binary offloading decision is present. By relaxing the binary constraints to real variables, [7] proposes an eDors algorithm for MEC systems. The algorithm is further extended in [8], which

✉ Li Ping Qian
lpqian@zjut.edu.cn

Liang Huang
lianghuang@zjut.edu.cn

Xu Feng
xfeng_zjut@163.com

Anqi Feng
aqfeng_zjut@163.com

Yupin Huang
yphuang_zjut@163.com

[1] College of Information Engineering, Zhejiang University of Technology, Hangzhou, China

Published online: 29 November 2018

iteratively improves the binary offloading decisions after relaxation. Separable semidefinite relaxation is applied to jointly optimize the binary offloading decisions of all users and the communication bandwidth allocation in [9], whose complexity is still too high for real-time offloading. However, all those algorithms are limited by the trade-off between optimality and computational complexity, which is not applicable for real-time computing offloading under the MEC networks with time-varying environments.

Deep learning that uses a deep neural network (DNN) with multiple processing layers to learn representations of data [13] has achieved many breakthroughs in different areas, e.g., robot control [14], natural language process [15], and gaming [16]. For systems with large-scale state-action space, using a DNN to approximate the state-action relationship can obtain near-optimal performance [16, 17]. Deep learning has also been applied to solve computationally expensive problems in wireless communications [18], e.g., resource allocation [19, 20], signal detections [21, 22], interference alignment [23], and caching [24]. There exist few recent works on deep reinforcement learning-based offloading for MEC networks [25–28]. However, most of existing works for MEC networks are based on the deep Q-network, whose tabular-search nature is not suitable for handling problems with high dimensional space.

In this paper, we resort to deep learning to propose an effective and efficient offloading framework for MEC networks. We consider an MEC network with one edge server and multiple wireless devices (WDs), where each WD can choose to offload its computation tasks to the edge server. To conserve energy and maintain quality of service for WDs, we obtain the following results:

1. We model the system utility of MEC networks as the weighted sum of energy consumption and task completion delay for all WDs. To minimize the system utility, we present a joint task offloading and bandwidth allocation problem for MEC networks that jointly optimizes the offloading decisions for each task of WDs and the transmission bandwidth for each WD.
2. We propose a distributed deep learning-based offloading algorithm for MEC networks, which is composed of offloading action generation and deep learning. The algorithm uses multiple parallel deep neural networks (DNNs) to effectively and efficiently generate offloading decisions. Those generated offloading decisions are stored in a shared memory to further train and improve DNNs.
3. We numerically demonstrate that the proposed algorithm converges to optimal when two or more DNNs are used. Under a wide range of parameter settings, the algorithm provides near-optimal offloading decisions in less than a second.

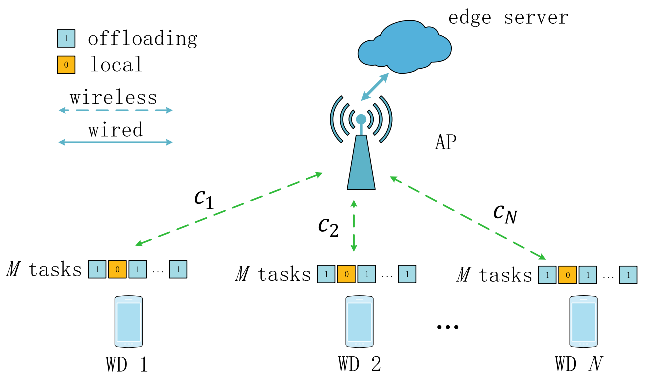

Fig. 1 System Model of an MEC network with multiple WDs and multiple tasks

The distributed deep learning-based offloading algorithm provides a new parallel method to effectively and efficiently generate offloading decisions for MEC networks. Although our work jointly solves offloading decision and bandwidth allocation, the proposed algorithm can be easily extended to other bandwidth allocation problems, e.g. transmission power allocation at WDs and computing resource allocation at edge server, by replacing the bandwidth allocation block in the algorithm structure with other dedicated ones.

The remainder of this paper is organized as follows. In Section 2, we present the system model and problem formulation. We propose distributed deep learning-based offloading algorithm in Section 3. Numerical results are presented in Section 4, and a conclusion is provided in Section 5.

2 System model and problem formulation

2.1 System model

In this work, we consider an MEC network composed by one edge server, one wireless access point (AP), and N WDs, denoted by a set $\mathcal{N} = \{1, 2, \ldots, N\}$, as shown in Fig. 1. The AP and the edge server is connected by an optical fiber, whose transmission delay can be ignored. Each WD has multiple tasks to be processed locally or be offloaded to the edge server via the AP. Without loss of generality, we assume that each WD has M independent tasks, denoted by a set $\mathcal{M} = \{1, 2, \ldots, M\}$. Denote d_{nm} as the workload of the m-th task of user n. Each WD n can determine whether to offload its task m to the edge server or not, and the offloading decision is denoted by a binary variable $x_{nm} \in \{0, 1\}$. Specifically, $x_{nm} = 1$ denotes that user n decides to offload its task m to the edge server, and $x_{nm} = 0$ means that user n decides to execute its task m locally. The detailed operations of edge computing and local computing are illustrated as follows.

2.2 Edge computing

When a task is offloaded to the edge server, the WD n transmits its workload d_{nm} to the AP via wireless channels which is further forwarded to and processed at the edge server. We neglect the energy consumption and delay when the edge server transmits the computing results back to WDs, because the data size of feedback information is small in general [4, 6, 7, 29]. We denote E^{t}_{nm} as the energy consumed by WDs for uploading its workload to the edge server and model the energy cost for data process at edge server as a linear function of workload d_{nm} [9]. Specifically, we denote the total cost for user n to offload its task m to the edge server as:

$$E^{\mathrm{c}}_{nm} = E^{\mathrm{t}}_{nm} + \alpha d_{nm}, \tag{1}$$

where α is the weight of the energy consumption at edge server. When $\alpha = 0$, we only consider the energy consumption at WDs. Notice that the cost E^{c}_{nm} includes the energy consumptions for sending the task and the server's utility cost for executing this task.

We next model the delay in computation offloading. Specifically, we use c_n to denote the allocated bandwidth to user n for transmitting its offloaded task to the edge server. Therefore, the transmission delay when user n offloads its task m to the edge server is given by:

$$T^{\mathrm{t}}_{nm} = \frac{d_{nm}}{c_n}. \tag{2}$$

In addition, the edge processing delay is given by

$$T^{\mathrm{c}}_{nm} = \frac{d_{nm}}{f^{\mathrm{c}}}, \tag{3}$$

where we denote f^{c} as the edge processing rate. In summary, given the offloading decisions $\{x_{nm}\}$, the total delay of user n when it executes MEC can be given by:

$$T^{\mathrm{c}}_{n} = \sum_{m=1}^{M} \left(T^{\mathrm{t}}_{nm} + T^{\mathrm{c}}_{nm}\right) x_{nm}, \forall n. \tag{4}$$

We assume that the edge server can only start to process user n's task m after this task is completely received by the edge, and the edge-server can only start to send back the output data after the entire task m is completed [9].

2.3 Local computing

We next model the case when user n decides to execute its task locally. Specifically, we use e^{l}_n to denote the local energy consumption per data bit of user n. Thus, user n's energy consumption for executing its task m locally is given by:

$$E^{\mathrm{l}}_{nm} = d_{nm} e^{\mathrm{l}}_n. \tag{5}$$

Meanwhile, we denote user n's local processing time per data bit as t^{l}. As a result, the total processing time for user n to execute its task m is given by:

$$T^{\mathrm{l}}_{nm} = d_{nm} t^{\mathrm{l}}. \tag{6}$$

Thus, given user n's offloading decision $\{x_{nm}\}$, the total delay for user n to finish its tasks locally is given by:

$$T^{\mathrm{l}}_{n} = \sum_{m=1}^{M} T^{\mathrm{l}}_{nm}(1 - x_{nm}). \tag{7}$$

2.4 Problem formulation

To minimize both the total delay finishing all users' tasks and the corresponding energy consumptions, we first introduce a system utility $Q(\boldsymbol{d}, \boldsymbol{x}, \boldsymbol{c})$ defined as the weighted sum of energy consumption and task completion delay, as

$$Q(\boldsymbol{d}, \boldsymbol{x}, \boldsymbol{c}) = \sum_{n=1}^{N} \left(\sum_{m=1}^{M} \left(E^{\mathrm{l}}_{nm}(1 - x_{nm}) + E^{\mathrm{c}}_{nm} x_{nm}\right) + \beta \max\{T^{\mathrm{l}}_n, T^{\mathrm{c}}_n\} \right),$$

where $\boldsymbol{d} = \{d_{nm} | n \in \mathcal{N}, m \in \mathcal{M}\}$, $\boldsymbol{x} = \{x_{nm} | n \in \mathcal{N}, m \in \mathcal{M}\}$, $\boldsymbol{c} = \{c_n | n \in \mathcal{N}\}$, and β denotes the weight of energy consumption and task completion. Then, we formulate an optimization problem (P1) to minimize $Q(\boldsymbol{d}, \boldsymbol{x}, \boldsymbol{c})$ by jointly optimizing each user n's offloading decisions $\{x_{nm}\}$ and the bandwidth allocations c_n for user n's task transmission, which is expressed as follows:

$$\begin{aligned}
(\mathrm{P1}) : Q^*(\boldsymbol{d}) = \underset{\boldsymbol{x}, \boldsymbol{c}}{\text{minimize}} \quad & Q(\boldsymbol{d}, \boldsymbol{x}, \boldsymbol{c}) && \text{(9a)}\\
\text{subject to:} \quad & \sum_{n=1}^{N} c_n \le C, && \text{(9b)}\\
& c_n \ge 0, \forall n \in \mathcal{N}, && \text{(9c)}\\
& x_{nm} \in \{0, 1\}. && \text{(9d)}
\end{aligned}$$

Here, constraint (9b) means that the total uplink bandwidth allocation for all users cannot exceed the maximum bandwidth C. The allocated bandwidth for each user c_n is either 0 or positive as given in (9c). The binary constraint on x_{nm} is given in (9d). Table 1 lists the important notations used in this paper.

The optimization problem (P1) is a mixed-integer programming problem, which is difficult to solve in general. In the next section, we study an approximate algorithm based on deep learning to efficiently and effectively solve (P1).

Table 1 Notations used in this paper

Notation	Definition
x_{nm}	$x_{nm} = 1$ if user n offloads its task m to the edge server. Otherwise, $x_{nm} = 0$
d_{nm}	Data size of user n's m-th task
E^{t}_{nm}	Data transmission energy consumption of user n's m-th task
c_n	Bandwidth assigned to user n
C	Total bandwidth
T^{t}_{nm}	Transmission time of user n's m-th task
T^{c}_{nm}	Edge processing time of user n's m-th task
f^{c}	Edge processing rate
T^{c}_{n}	Edge processing time consumption of user n
e^{l}_{n}	Local energy consumption per data bit of user n
E^{l}_{nm}	Local energy consumption of user n's m-th task
t^{l}	Local processing time per data bit
T^{l}_{nm}	Local processing time consumption of user n's m-th task
α	Weight of the system utility cost
β	Weight between energy consumption and processing delay in the system cost
E^{c}_{nm}	Energy consumption of the edge sever for executing user n's m-th task
T^{l}_{n}	Local processing delay of user n

3 DDLO algorithm

In this section, we propose a DDLO algorithm for MEC networks, where K parallel multiple DNNs are used to generate binary offloading decisions.

Given the sizes of output data and input data of all users, denoted as $\boldsymbol{d}$, our goal is to find an offloading policy function π to generate the optimal offloading action $\boldsymbol{x}^* \in \{0,1\}^{NM}$ for (P1), as

$$\pi : \boldsymbol{d} \mapsto \boldsymbol{x}^*. \tag{10}$$

The size of target binary offloading decision set $\{\boldsymbol{x}\}$ is 2^{NM}, which exponentially increases with the number users N in MEC networks and the number of tasks per user M. Since it is NP-hard to find the optimal offloading action, in this paper we approximately express π by a parameterized function based on DNN.

We propose a DDLO algorithm for MEC networks, which is composed of offloading action generation and deep learning, as illustrated in Fig. 2. Specifically, for each input $\boldsymbol{d}$, K distributed offloading actors are used to efficiently and effectively generate K candidate offloading actions $\{\boldsymbol{x}_k | k \in \mathcal{K}\}$ where $\mathcal{K} = \{1, 2, \ldots, K\}$. Then, the offloading action with the lowest system utility is chosen as the output, denoted as $\boldsymbol{x}^*$. Finally, the data entry $(\boldsymbol{d}, \boldsymbol{x}^*)$ is further stored in a memory structure to train these

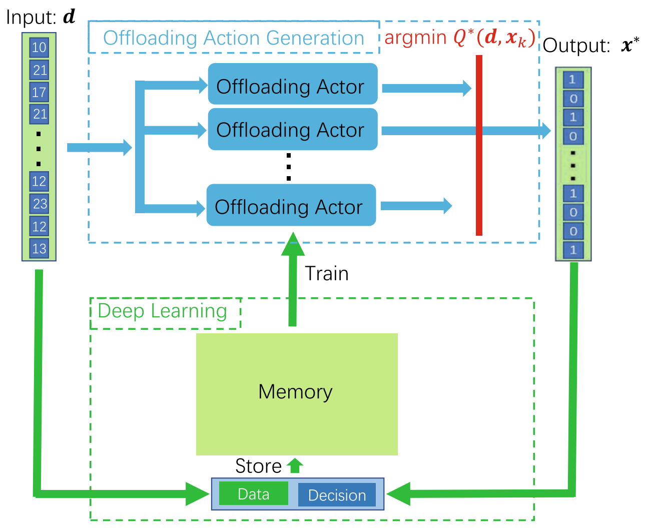

Fig. 2 Architecture of distributed deep learning-based partial offloading algorithm

K distributed offloading actors. The detailed procedures of offloading action generation and deep learning are explained as follows.

3.1 Offloading action generation

For each input $\boldsymbol{d}$, we use K offloading actors to generate K candidate offloading actions with one action per actor, as shown in Fig. 3. Inside each offloading actor, a DNN is used to generate binary offloading action $\boldsymbol{x}_k$, which can be represented by a parametrized function f_{θ_k}, as

$$f_{\theta_k} : \boldsymbol{d} \to \boldsymbol{x}_k, \tag{11}$$

where θ_k denotes the parameters of the k-th DNN. All those K DNNs have the same structure but with different parameter values θ_k.

Once a binary offloading decision $\boldsymbol{x}_k$ is given, the original optimization problem (P1) becomes a bandwidth allocation problem (P2), as

$$\text{(P2)} : Q^*(\boldsymbol{d}, \boldsymbol{x}) = \underset{\boldsymbol{c}}{\text{minimize}} \quad Q(\boldsymbol{d}, \boldsymbol{x}, \boldsymbol{c}) \tag{12a}$$

$$\text{subject to:} \quad \sum_{n=1}^{N} c_n \le C, \tag{12b}$$

$$c_n \ge 0, \forall n \in \mathcal{N}. \tag{12c}$$

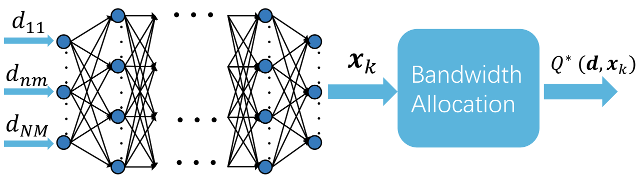

Fig. 3 An offloading actor

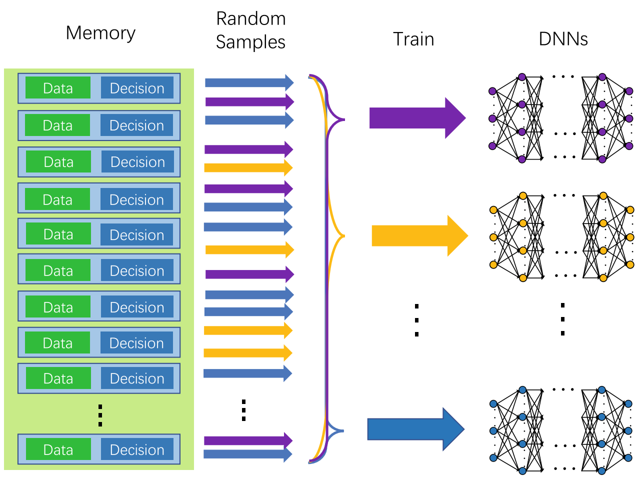

Fig. 4 K DNNS are separately trained with randomly sampled data entries

There are extensive studies on how to efficiently solve bandwidth allocation problems in the literature [30, 31]. In this paper, (P2) is a convex problem which is solved by a standard optimization tool written in Python, known as scipy.[1]

After solving K bandwidth allocation problems (P2), the offloading action with least $Q^*(\boldsymbol{d}, \boldsymbol{x}_k)$ is selected among all K candidates, as

$$\boldsymbol{x}^* = \arg\min_{k\in\mathcal{K}} Q^*(\boldsymbol{d}, \boldsymbol{x}_k). \quad (13)$$

This selected $\boldsymbol{x}^*$ will be output as the binary offloading decision with respect to the input $\boldsymbol{d}$.

3.2 Deep learning

Once the best offloading decision $\boldsymbol{x}^*$ is obtained, we save it as a new entry of labeled data, $(\boldsymbol{d}, \boldsymbol{x}^*)$, in the finite-size memory structure, where the oldest data entry is discarded when the memory is full. Those generated labeled data are then used to train all K DNNs, as illustrated in Fig. 4.

Instead of training DNNs with the whole memory of data, an experience relay technology [16, 32, 33] is used in the proposed algorithm. Specifically, all DNNs share the same memory and each DNN randomly extracts a batch of data samples from the memory. A gradient descent algorithm is performed to optimize parameter values θ_k of each DNN by minimizing the cross-entropy loss, as

$$L(\theta_k) = -\boldsymbol{x}^T \log f_{\theta_k}(\boldsymbol{d}) - (1-\boldsymbol{x})^T \log(1 - f_{\theta_k}(\boldsymbol{d})).$$

The finite-size design of the memory structure helps to improve data efficiency, since newly generated data entries are more preferred than older ones. Other more advanced techniques, such as distributed importance sampling [34, 35] and prioritized experience replay [33, 36], can be applied to further speed up training.

The proposed DDLO algorithm for MEC networks is shown in Algorithm 1, which is implemented using TensorFlow [37]. At the beginning, all K DNNs are initialized with random parameter values θ_k and the memory is empty. By choosing a $K \geq 2$, the algorithm is expected to converge to optimal offloading actions. We numerically study the convergence and performance for the proposed algorithm in the next section.

Algorithm 1 DDLO algorithm

```
1:  Input: Input different workloads $\boldsymbol{d}_t$ at time $t$
2:  Output: Optimal offloading decision $\boldsymbol{x}_t^*$ at time $t$
3:  Initialization:
4:    Initialize the $K$ DNNs with random parameters $\theta_k$,
      $k \in \mathcal{K}$;
5:    Empty the memory structure
6:  for $t = 1, 2, \ldots, G$ do
7:      Replicate different workloads $\boldsymbol{d}_t$ to all $K$ DNNs.
8:      Generate $k$-th offloading action candidate $\boldsymbol{x}_k$ from
        the $k$-th DNN in a parallel way, as $\boldsymbol{x}_k = f_{\theta_{k,t}}(\boldsymbol{d}_t)$;
9:      Solve all $K$ bandwidth allocation optimization (P2)
        in a parallel way when the offloading action
        candidates $\{\boldsymbol{x}_k\}$ are present;
10:     Select the best offloading decision as the output
        $\boldsymbol{x}_t^* = \arg\min_{k\in\mathcal{K}} Q^*(\boldsymbol{d}_t, \boldsymbol{x}_k)$;
11:     Store $(\boldsymbol{d}_t, \boldsymbol{x}_t^*)$ into the memory structure;
12:     Randomly Sample $K$ batches of training data from
        the memory structure;
13:     Train the DNNs and update $\theta_{k,t}$;
14: end for
```

4 Performance evaluation

In this section, we show the numerical results for our proposed algorithm for solving Problem (P1). In our simulation, we set the number of WDs $N = 3$ and each user has $M = 3$ different tasks. In addition, we set the local computation time of the mobile device as 4.75×10^{-7} s/bit, and the processing energy consumption as 3.25×10^{-7} J/bit [9]. We assume that the input data size of all tasks is randomly distributed between 10MB and 30MB. In all those simulations, we set the uplink bandwidth limit as 150 Mbps. The receiving energy consumption and transmission energy consumption of the mobile device are both 1.42×10^{-7} J/bit.

[1]The source code of scipy is available at https://www.scipy.org.

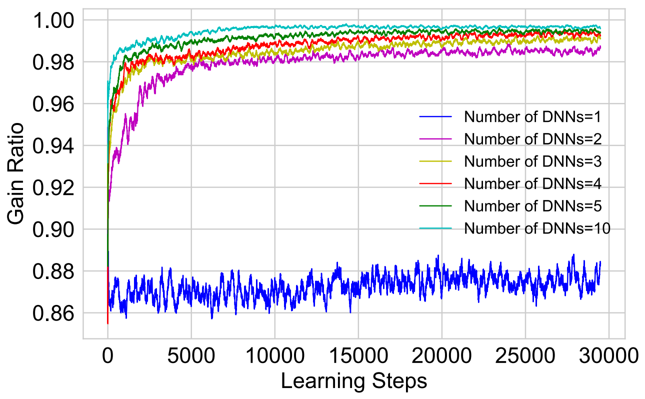

Fig. 5 Convergence performance under different number of DNNs

The CPU rate of the edge sever is 10×10^9 cycle/s. We further set $\alpha = 1.5 \times 10^{-7}$ J/bit, and $\beta = 1$ J/s.

4.1 Convergence performance

Figure 5 shows the effects of number of DNNs on the convergence of our DDLO algorithm. Because the system utility $Q(\boldsymbol{d}, \boldsymbol{x}, \boldsymbol{c})$ varies under different input $\boldsymbol{d}$, we introduce a greedy method as a benchmark, which generates the optimal system utility by enumerating all 2^{NM} binary offloading decisions, as

$$\max_{\boldsymbol{x}' \in \{0,1\}^{NM}} Q^*(\boldsymbol{d}, \boldsymbol{x}'). \tag{14}$$

For better comparison, we plot the gain ratio between DDLO and the greedy method, as shown in Fig. 5. DDLO algorithm converges to 1 with the increase of learning steps. A gain ratio 0.98 is achieved within 1000 learning steps when 5 DNNs are used. However, when only one DNN is used, the DDLO learns nothing from the data generated by itself and cannot converge. Therefore, at least 2 DNNs are required by DDLO. The more DNNs are used, the faster DDLO converges.

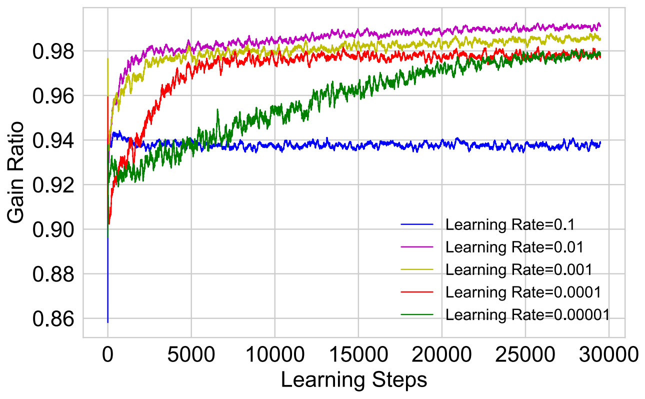

Fig. 6 Convergence performance under different learning rates

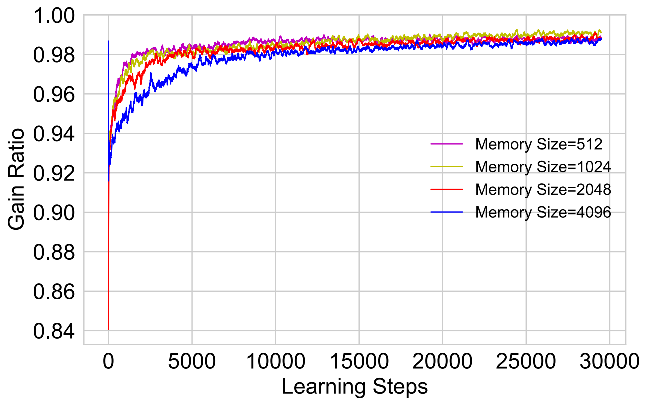

Fig. 7 Convergence performance under different memory sizes

We study the convergence of our DDLO algorithm under different learning rates and different memory sizes in Figs. 6 and 7, respectively. The higher the learning rate, the faster the convergence speed of DDLO is. However, when the learning rate increases, we suffer from a larger possibility of obtaining a local optimal solution instead of the global optimal one. Hence, we need to choose an appropriate learning rate regarding to specific situations. We study the effects of different memory sizes in Fig. 7. A smaller size of memory leads to faster convergence but may fall into local optimum. In the considered MEC network, we select an experience replay memory with size 1024 as a compensation between convergence and performance.

4.2 System utility

We compare the performance of our proposed DDLO method with other four schemes, specifically, the local processing only scheme, the edge processing only scheme, the greedy scheme, and the deep Q-network based algorithm

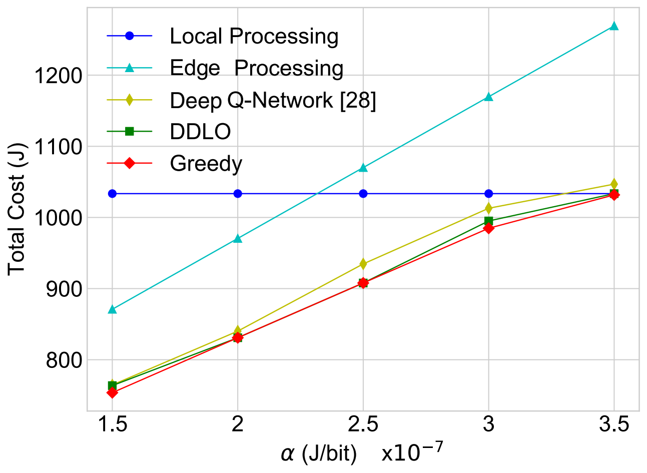

Fig. 8 Energy cost under different α for different offloading algorithms

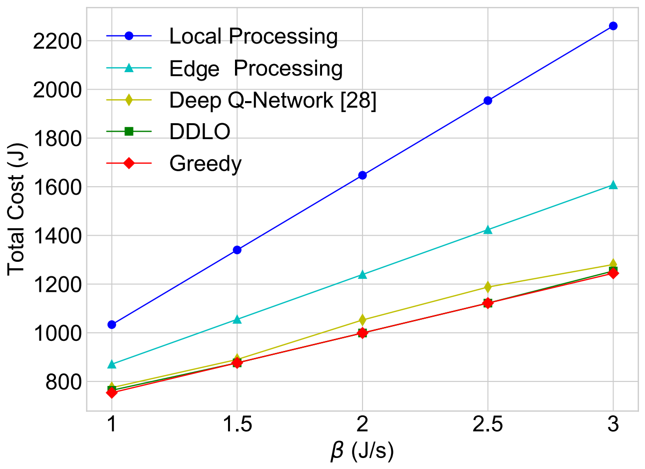

Fig. 9 Energy cost under different β for different offloading algorithms

[28]. The performance is studied under different α and β parameters in Figs. 8 and 9, respectively. The local processing only scheme means that all users' tasks are processed locally. The edge processing only scheme means that the edge processes all users' tasks. For the greedy scheme, we enumerate all offloading decision combinations and then adopt the best one. However, it is noticed that the greedy scheme is very time-consuming, especially when the number of users and tasks are large. As illustrated in Figs. 8 and 9, as α or β increases, the total energy cost of all schemes increases. Our DDLO method outperforms the deep Q-network based method, which is close to the greedy scheme.

4.3 Computation time

We further study the computation time of DDLO under different number of DNNs in Fig. 10. For DDLO employed with different number of DNNs, the computation time

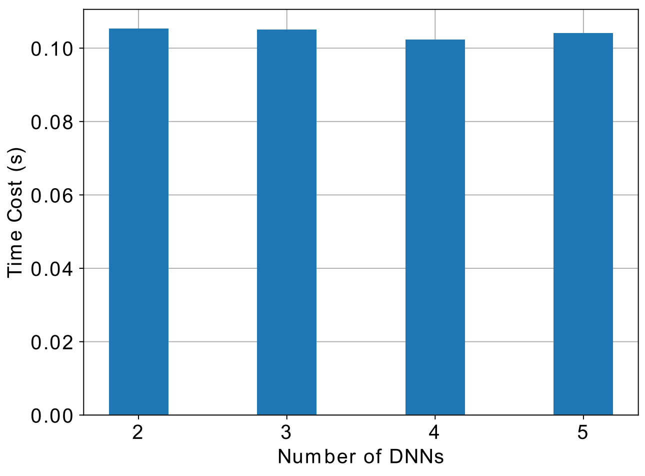

Fig. 10 Time cost for each input under different number of DNNs

for each input is almost identical, around 0.1 second for an MEC network with 3 WDs with 3 tasks. Since the convergence performance of DDLO can be improved by increasing the number of DNNs as shown in Fig. 5, we can deploy DDLO with more parallel DNNs depending on the computing hardware at edge server. Nowadays, all those server-oriented CPUs have multi-core support, e.g. a single Intel Xeon W-2195 processor owns 18 cores.[2] Therefore, it is applicable to implement DDLO on multiple cores to boost its performance.

5 Conclusion

In this work, we have proposed a distributed deep learning-based offloading algorithm, DDLO, for MEC networks, to minimize the overall system utility including both the total energy consumption and the delay in finishing the task. The algorithm takes advantages of multiple DNNs and generates close-to-optimal solutions without manually labeled data. Numerical results have validated the accuracy of the proposed algorithm and the performance advantage compared with the existing deep Q-network algorithm. Furthermore, the proposed DDLO algorithm can generate near-optimal offloading decisions in less than one second, whose computation time is independent of the number of DNNs. We expect that such a distributed deep learning-based framework can be further extended to optimize real-time offloading in future implementation of MEC networks.

Acknowledgments This work was supported in part by the National Natural Science Foundation of China under Grant No.61502428, and in part by the Zhejiang Provincial Natural Science Foundation of China under Grants No.LR16F010003 and No.LY19F020033.

References

1. Chiang M, Zhang T (2016) Fog and IoT: an overview of research opportunities. IEEE Internet J 3(6):854–864
2. Mao Y, You C, Zhang J, Huang K, Letaief K (2017) A survey on mobile edge computing: the communication perspective. IEEE Commun Surv Tutorials 19(4):2322–2358
3. Abbas N, Zhang Y, Taherkordi A, Skeie T (2018) Mobile edge computing: a survey. IEEE Internet J 5(1):450–465
4. Chen X, Jiao L, Li W, Fu X (2016) Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans Netw 24(5):2795–2808
5. Tran TX, Pompili D (2017) Joint task offloading and resource allocation for multi-server mobile-edge computing networks. arXiv:1705.00704
6. Bi S, Zhang Y (2018) Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. IEEE Trans Wirel Commun 17(6):4177–4190

[2] https://www.intel.com/

7. Guo S, Xiao B, Yang Y, Yang Y (2016) Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing. In: IEEE international conference on computer communications, pp 1–9
8. Zhang J, Xia W, Zhang Y, Zou Q, Huang B, Yan F, Shen L (2017) Joint offloading and resource allocation optimization for mobile edge computing. In: IEEE global communications conference, pp 1–6
9. Chen M, Liang B, Dong M (2016) Joint offloading decision and bandwidth allocation for multi-user multi-task mobile edge. In: 2016 IEEE international conference on communications, Kuala Lumpur, pp 1–6
10. Qian LP, Zhang Y, Huang H, Wu Y (2013) Demand response management via real-time electricity price control in smart grids. IEEE J Sel Areas Commun 31(7):1268–1280
11. Qian LP, Wu Y, Zhou H, Shen XS (2017) Joint uplink base station association and power control for small-cell networks with non-orthogonal multiple access. IEEE Trans Wirel Commun 16(9):5567–5582
12. Qian LP, Wu Y, Zhou H, Shen XS (2017) Dynamic cell association for non-orthogonal multiple-access V2S networks. IEEE J Sel Areas Commun 35(10):2342–2356
13. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436
14. Phaniteja S, Dewangan P, Guhan P, Sarkar A, Krishna K (2017) A deep reinforcement learning approach for dynamically stable inverse kinematics of humanoid robots. In: 2017 IEEE international conference on robotics and biomimetics (ROBIO), Macau, pp 1818–1823
15. Sharma A, Kaushik P (2017) Literature survey of statistical, deep and reinforcement learning in natural language processing. In: International conference on computing, communication and automation, Greater Noida, pp 350–354
16. Mnih V, Kavukcuoglu K et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533
17. Dulac-Arnold G, Evans R, Hasselt H, Sunehag P, Lillicrap T, Hunt J, Mann T, Weber T, Degris T, Coppin B (2015) Deep reinforcement learning in large discrete action spaces. arXiv:1512.07679
18. Zhang C, Patras P, Haddadi H (2018) Deep learning in mobile and wireless networking: a survey. arXiv:1803.04311
19. Sun H, Chen X, Shi Q, Hong M, Fu X, Sidiropoulos ND (2017) Learning to optimize: Training deep neural networks for wireless resource management. In: Proceedings of IEEE international workshop on signal processing advances in wireless communications, pp 1–6
20. Xu Z, Wang Y, Tang J, Wang J, Gursoy MC (2017) A deep reinforcement learning based framework for power-efficient resource allocation in edge RANs. In: Proceedings of IEEE international conference on communications, pp 1–6
21. Samuel N, Diskin T, Wiesel A (2017) Deep MIMO detection. In: IEEE 18th Int. Workshop Signal Process. Adv. Wireless Commun., pp. 690–694
22. Ye H, Li GY, Juang BH (2018) Power of deep learning for channel estimation and signal detection in OFDM systems. IEEE Wireless Commun Lett 7(1):114–117
23. He Y, Zhang Z, Yu FR, Zhao N, Yin H, Leung VC, Zhang Y (2017) Deep-reinforcement-learning-based optimization for cache-enabled opportunistic interference alignment wireless networks. IEEE Trans Veh Technol 66(11):10433–10445
24. Zhong C, Gursoy MC, Velipasalar S (2018) A deep reinforcement learning-based framework for content caching. In: IEEE 52nd Annual Conference on Information Sciences and Systems (CISS), pp. 1–6
25. He Y, Yu FR, Zhao N, Leung VC, Yin H (2017) Software-defined networks with mobile edge computing and caching for smart cities: a big data deep reinforcement learning approach. IEEE Commun Mag 55(12):31–37
26. Min M, Xu D, Xiao L, Tang Y, Wu D (2017) Learning-based computation offloading for IoT devices with energy harvesting. arXiv:1712.08768
27. Chen X, Zhang H, Wu C, Mao S, Ji Y, Bennis M (2018) Performance optimization in mobile-edge computing via deep reinforcement learning. arXiv:1804.00514
28. Huang L, Feng X, Qian LP, Wu Y (2018) Deep reinforcement learning-based task offloading and resource allocation for mobile edge computing. In: 3rd EAI International Conference on Machine Learning and Intelligent Communications, pp 1–10
29. Huang L, Bi S, Qian LP, Xia Z (2018) Adaptive scheduling in energy harvesting sensor networks for green cities. IEEE Trans Ind Inf 14(4):1575–1584
30. You C, Huang K, Chae H, Kim B (2017) Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading. IEEE Trans Wirel Commun 16(3):1397–1411
31. Zhang H, Liu H, Cheng J, Leung MCV (2018) Downlink Energy Efficiency of Power Allocation and Wireless Backhaul Bandwidth Allocation in Heterogeneous Small Cell Networks. IEEE Trans Commun 66(4):1705–1716
32. Lin LJ (1993) Reinforcement learning for robots using neural networks, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, Technical Report
33. Horgan D, Quan J, Budden D, Barth-Maron G, Hessel M, van Hasselt H, Silver D (2018) Distributed prioritized experience replay. arXiv:1803.00933
34. Loshchilov I, Hutter F (2015) Online batch selection for faster training of neural networks. arXiv:1511.06343
35. Alain G, Lamb A, Sankar C, Courville A, Bengio Y (2015) Variance reduction in SGD by distributed importance sampling. arXiv:1511.06481
36. Schaul T, Quan J, Antonoglou I, Silver D (2016) Prioritized experience replay. In International conference on learning representations (ICLR)
37. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S (2016) Tensorflow: large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467