Fair Resource Allocation Based on Deep Reinforcement Learning in Fog Networks

Huihui Xu[1](✉), Yijun Zu[2], Fei Shen[3,4], Feng Yan[2], Fei Qin[5], and Lianfeng Shen[2]

[1] Wuhan University, Wuhan 430072, China
xuhuis@whu.edu.cn

[2] National Mobile Communications Research Laboratory, Southeast University, Nanjing 210096, China
zyj@seu.edu.cn

[3] Key Lab of Wireless Sensor network and Communication, Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai 200050, China
fei.shen@mail.sim.ac.cn

[4] Shanghai Research Center for Wireless Communications, Shanghai 201210, China

[5] The School of Electronic and Electrical Communication Engineering, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract. As the terminal devices grow explosively, the resource in a fog network may not satisfy all the requirement of them. Thus scheduling the resource reasonably becomes a huge challenge in the future 5G network. In the paper, we propose a fair resource allocation algorithm based on deep reinforcement learning, which makes full use of the computational resource in a fog network. The goal of the algorithm is to complete processing the tasks fairly for all the user nodes (UNs). The fog nodes (FNs) are expected to assign their central processing unit (CPU) cores to process offloading tasks reasonably. We apply the Deep Q-Learning Network (DQN) to solve the problem of resource scheduling. Firstly, we establish an evaluation model of a priority to set the priority for the offloading tasks, which is related to the reward in the reinforcement learning. Secondly, the model of reinforcement learning is built by taking the situation of UNs and resource allocation scheme as the state of environment and the action of the agent, respectively. Subsequently, a loss function is analysed to update the parameters of a deep neural network. Finally, numerical simulations demonstrate the feasibility and effectiveness of our proposed algorithm.

This work is supported in part by the National Natural Science Foundation of China (No. 61871370, No. 61601122 and No. 61773266), the Natural Science Foundation of Shanghai, China (No. 18ZR1437500 and No. 19ZR1423100), the Hundred Talent Program of Chinese Academy of Sciences (No. Y86BRA1001), the Fundamental Research Funds for the Central Universities (No. 2242019K40188), the Key Research & Development Plan of Jiangsu Province (No. BE2018108), and the Postdoctoral Science Foundation of China (No. 2019M651476).

J. Zheng et al. (Eds.): ADHOCNETS 2019, LNICST 306, pp. 135–148, 2019.
https://doi.org/10.1007/978-3-030-37262-0_11

Keywords: Fog computing · Deep reinforcement learning · Resource allocation

1 Introduction

As thebibliography 5G era comes, there are an increasing number of terminal devices connecting to the network [1]. In addition, most of mobile applications generate massive data, such as augmented reality and autopilot. This brings a heavy burden on the transmission link to the cloud data server. Therefore, the fog computing is proposed to relieve the pressure of link delay and congestion in the network [2]. In a fog network, fog nodes (FNs) are expected to be deployed in the whole network and help process the data tasks offloaded from user nodes (UNs). However, as the data tasks grow explosively in the fog network, the FNs with limited resources, such as computational resources, storage resources and network resources, can not process the tasks in time, which will cause some tasks to be unprocessed for a long time. Therefore, in order to guarantee the data tasks to be processed effectively and fairly [4], it is essential to allocate the resources reasonably in the fog network.

In past few years, numerous researches focusing on resource allocation have been carried out. For example, among them, a double-matching strategy for the resource allocation problem in fog computing network is proposed [6]. The strategy analyses the utility and cost of resource allocation to maximize the cost efficiency, which is an extension of classic matching algorithm. A novel mechanism named Gaussian Process Regression for Fog-Cloud Allocation (GPRFCA) is introduced [3]. The GPR method is utilized to predict the future requests to improve the utilization of limited resource in the FNs and the Power Usage Effectiveness is considered as the metric to improve the energy efficiency of FNs. The joint radio and computational resource allocation in fog networks is formulated as a mixed integer nonlinear programming problem [5], with the constraints including transmission power, service delay and so on. To improve the user experience, the authors propose a matching game framework to solve the resource allocation problem.

Although there exist a large amount of literatures about resource allocation in fog networks, few literatures solve the resource allocation problem with deep reinforcement learning [9]. In this paper, we focus on the computational resource allocation and solve the problem of fair computational resource allocation with a classic deep reinforcement learning algorithm, Deep Q-Learning Network (DQN) [7]. After modeling computational resource allocation problem into Markov decision process, we propose a fog computing resource allocation strategy based on DQN. The strategy first sets priorities for different tasks, and then utilizes the deep reinforcement learning framework to perceive dynamical resource environments. Finally, computational resources are reallocated to different tasks in real time according to the task priorities. The strategy can not only alleviate the computing pressure of the UNs and dynamically sense the environment but also realize the automatic allocation of resources and determine the optimal resource allocation scheme.

Therefore, our work can be concluded as follows:

(1) We determine the user priorities and define different task states. The task priority model is established according to different user priorities and task states. Under the model of task priority, resource utilization and task execution rate are improved, which meets the quality of experience (QOE) of different users.
(2) We solve the resource allocation problem in the fog network with deep reinforcement learning. The state and reward of environment and the action of agent are set to establish the model of resource allocation reinforcement learning. The value function is defined to update the neural network and the analysis of the update is provided.
(3) The algorithm based on deep reinforcement learning is realized, and the resource allocation simulation experiment is completed. The effectiveness and feasibility of our proposed algorithm are verified.

The remainder of this paper is organized as follows. The preliminary, including system model and task priority, is introduced in Sect. 2. We formulate the problem in Sect. 3, where we transform the resource allocation in the fog network into the problem of deep reinforcement learning. In Sect. 4, we apply the classic deep reinforcement learning DQN to solve the problem. Numerical simulations are provided to evaluate the performance of our proposed strategy in Sect. 5. At last, the paper is concluded in Sect. 6.

2 Preliminary

2.1 System Model

We consider a fog network consisting of the cloud data server, a FN and multiple UNs, as depicted in Fig. 1. The FN is available with M total central processing unit (CPU) cores for processing the offloading tasks. The computational capacity of each CPU core is denoted as C, measured in [bit/s]. We denote the set of UNs as $\mathcal{U} = \{U_1, \dots, U_n, \dots, U_N\}$. The operating time of the fog network system is slotted and we denote the index of time slot as k with $k \in \mathcal{K} = \{1, 2, 3, \dots\}$. Each time slot is denoted by t_s (in seconds). For the UNs, the possibility of data generation satisfies discrete-time On/Off Markov arrival model [8]. At each time slot, the data from UN U_n is transmitted to the FN and cached in the buffer n, where the queues in the buffer are first-in-first-out (FIFO) queues. Subsequently, the FN allocates the computational resources (CPU cores) to the cached data in the buffer. At the beginning of time slot, the CPU cores are reallocated to the different buffers according to the size of data to be processed in the different buffers. After processed by the FN, the result of the data is transmitted back to the UNs or forwarded to the cloud data server.

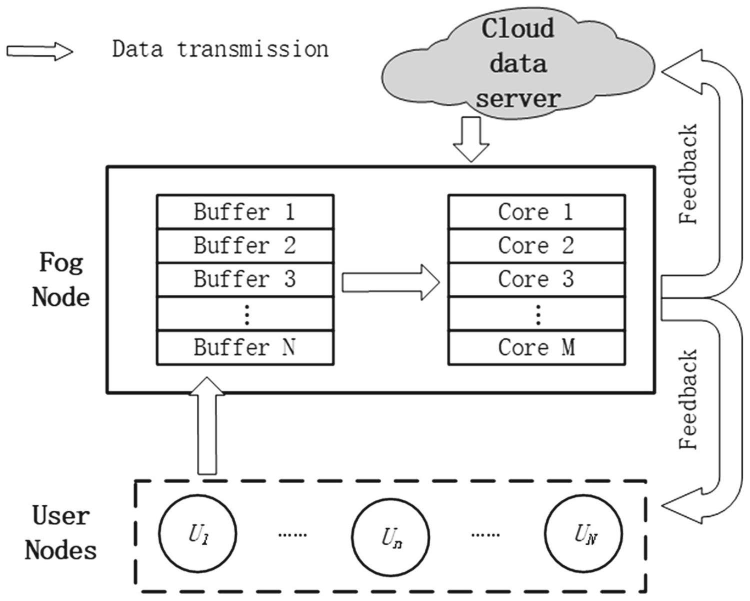

Fig. 1. System model.

2.2 Task Priority Model

Due to the multi-source nature of the UNs and the large number of data tasks, there exists the competition between the UNs. To prevent the link congestion problem caused by resource competition, it is especially essential to arrange the order of task processed by the FN reasonably. Therefore, it is considered to set a priority level for the data task from UNs before the data tasks are processed. The data tasks with higher priority are assigned computational resources preferentially.

We take the UN priority, the waiting time and the size of data tasks to be processed into consideration. The specific priority strategy are as follows. The U_n priority is represented as ζ_n. Different UNs have different user priority levels. For example, a car node has a higher user priority than a sensor node. Additionally, considering the task response speed, too long waiting time is not allowed, which results in the link congestion and poor quality of experience (QoE). Therefore, the longer the waiting time, the higher the task priority. We denote W_n^k, $W_{\max}^k$ as the waiting time of the data task from U_n, the maximum of the waiting time among the data tasks during the time slot k, respectively. At last, the quantity of the cumulative tasks in the buffer should be considered. Similarly, the more the cumulative tasks in the buffer, the higher the task priority. We denote D_n^k, $D_{\max}^k$ as the cumulative tasks from U_n, the maximum of cumulative tasks during the time slot k, respectively.

Above all, we can obviously obtain the priority of the data task from U_n as follow.

$$\theta_n^k = \left(\left\lfloor \frac{D_n^k}{D_{\max}^k} \right\rfloor \times \alpha + \left\lfloor \frac{W_n^k}{W_{\max}^k} \right\rfloor \times \beta \right) \times \zeta_n, \tag{1}$$

where α, β represent the weight coefficient of the cumulative tasks and the waiting time, respectively.

Under consideration of fair resource allocation, the priority θ_n^k is adjusted dynamically according to the state of the fog network, including the cumulative tasks and the waiting time.

3 Problem Formulation

Resource allocation in the fog network is a decision-making process where to allocate computational resources to process the data tasks from the UNs during each time slot. Thus, we model this decision-making process into a Markov decision process. Because the data tasks generate randomly in the UNs, the Markov transition probabilities are uncertain. The environment's dynamics are unknown when the agent selects the action, thereby we use model-free reinforcement learning to obtain the best policy. Therefore, the fog computing resource allocation strategy based on deep reinforcement learning is to continuously sample by interacting with the external environment, and to find the optimal strategy by maximizing the cumulative reward. The resource allocation model based on reinforcement learning is established as follows.

3.1 State Vector

At the beginning of each slot time, the FN caches the data tasks, which generates at the previous time slot, to the corresponding buffers. The system state can be obtained as follows.

- $\boldsymbol{D}^k$: $\boldsymbol{D}^k$ is the vector of the cumulative tasks in the buffers and can be represented as $\left[D_1^k, \ldots, D_n^k, \ldots, D_N^k\right]$, where D_n^k is the cumulative tasks in the nth buffer during the time slot k.
- $\boldsymbol{W}^k$: $\boldsymbol{W}^k$ is the vector of the waiting time. And $\boldsymbol{W}^k$ can be represented as $\left[W_1^k, \ldots, W_n^k, \ldots, W_N^k\right]$, where W_n^k is the waiting time of data tasks from U_n in the time slot k.

Finally, the state of the fog network can be denoted as

$$\boldsymbol{s}^k = \{\boldsymbol{D}^k, \boldsymbol{W}^k\}. \tag{2}$$

3.2 Action Vector

After sensing the state of the fog network, the agent selects the action from the action set $\mathcal{A}$ according to the strategy π, namely, $\boldsymbol{a}^k = \pi\left(\boldsymbol{s}^k\right)$. $\boldsymbol{a}^k$ is the vector of action and can be expressed as

$$\boldsymbol{a}^k = \left[a_1^k, \ldots, a_n^k, \ldots, a_N^k\right], \tag{3}$$

where $\sum_{n=1}^{N} a_n^k = M$ and a_n^k represents the number of CPU cores allocated to process the data tasks from U_n.

After determining $\boldsymbol{a}^k$, if there exists $a_n^k = 0$, meaning the data tasks from U_n are not processed during the time slot k, the waiting time increases. The waiting time state of U_n can be expressed as

$$W_n^{k+1} = \begin{cases} W_n^k + 1, & \text{if } a_n^k = 0 \\ 0, & \text{if } a_n^k \neq 0 \end{cases} \tag{4}$$

3.3 Reward Function

Taking the action $\boldsymbol{a}^k$ under the state of $\boldsymbol{s}^k$, the agent can obtain a reward from the environment. Effectively setting the reward function is very important for deep reinforcement learning to achieve the desired goal. In order to maximize the execution rate of the task and the satisfaction of the user, we set the reward function as follows.

$$r^k = \sum_{n=1}^{N} \theta_n^k \times \phi_n^k, \tag{5}$$

where ϕ_n^k represents the data tasks from UN U_n processed during the time slot k, and can be expressed as

$$\phi_n^k = a_n^k C t_s. \tag{6}$$

At the time slot k, the agent obtains the immediate reward r^k. Our goal is to maximize the total reward, which can be expressed as follows.

$$R^k = \sum_{i=0}^{\infty} \gamma^i r^{k+i}, \tag{7}$$

where the discount factor $\gamma \in (0, 1)$ weighs the myopic or foresighted decisions.

3.4 Value Function

The action $\boldsymbol{a}^k$ is drawn from a stochastic policy $\pi(\boldsymbol{a}|\boldsymbol{s}) = \Pr\left(\boldsymbol{a}^k = \boldsymbol{a}|\boldsymbol{s}^k = \boldsymbol{s}\right)$, which is a mapping from the state of the fog network to the probability of taking actions. The value function is the expected value of cumulative discounted rewards received over the entire process following the policy. The process of reinforcement learning contains two phases: (a) policy evaluation and (b) policy improvement. In the first phase, the agent samples data according to the stochastic policy $\pi(\boldsymbol{a}|\boldsymbol{s})$. In the second phase, the agent updates the policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ according to the value function.

There are two definitions of value function: (a) the state value function $V(\boldsymbol{s})$ and (b) the state-action value function $Q(\boldsymbol{s}, \boldsymbol{a})$. The relationship between $V(\boldsymbol{s})$ and $Q(\boldsymbol{s}, \boldsymbol{a})$ satisfies $V(\boldsymbol{s}) = \sum\limits_{s} \Pr\left(\boldsymbol{a}^k = \boldsymbol{a}|\boldsymbol{s}^k = \boldsymbol{s}\right) Q(\boldsymbol{s}, \boldsymbol{a})$. $Q(\boldsymbol{s}, \boldsymbol{a})$ can be expressed as

$$Q(\boldsymbol{s}, \boldsymbol{a}) = E\left[\sum_{i=0}^{\infty} \gamma^i r^{k+i} | \boldsymbol{s}^k = \boldsymbol{s}, \boldsymbol{a}^k = \boldsymbol{a}\right]. \tag{8}$$

The optimal $Q(\boldsymbol{s},\boldsymbol{a})$ is denoted as $Q^{*}(\boldsymbol{s},\boldsymbol{a})$ and can be calculated by the Bellman optimality equation as follows.

$$Q^{*}(\boldsymbol{s}^{k},\boldsymbol{a}^{k})=E\left[r^{k}+\gamma\max_{\boldsymbol{a}^{k+1}}Q\left(\boldsymbol{s}^{k+1},\boldsymbol{a}^{k+1}\right)\right]. \tag{9}$$

To optimize the long-term performance, the Eq. (9) is adopted to update the value function in the Q-learning algorithm. However, when the space of action is very high or even continuous, it is almost impossible to calculate all $Q(\boldsymbol{s},\boldsymbol{a})$. Consequently, neural network can be used as the function approximator to approximate the value function.

3.5 Q-Value Approximation

We apply deep neural network (DNN) to approximate the value function. Therefore, the Q-value can be represented as $Q_{\boldsymbol{w}}(\boldsymbol{s},\boldsymbol{a})$, which uses a fully-connected DNN that is parameterised by a set of weights $\boldsymbol{w}=\{w_1,w_2,\ldots,w_n\}$. DNN is comprised of input layer, output layer and hidden layers. The calculation of each layer consists of three parts: weight, bias and activation (e.g., sigmoid, tanh, ReLU). For example, let y_{ij} and x_i denote the output value of the jth neuron in the layer i and the input values, respectively. Therefore, we can obtain

$$y_{ij}=f_{act}\left(\boldsymbol{w}_i\cdot\boldsymbol{x}_i+b_{ij}\right), \tag{10}$$

where f_{act} is the activation function, $\boldsymbol{w}_i$ is the weights in the layer i and b_{ij} is the bias.

The network parameters will be updated by minimizing the loss function, which can be expressed as

$$L(\boldsymbol{w})=E\left[r^{k}+\gamma\max_{a^{k+1}}Q_{\boldsymbol{w}}\left(\boldsymbol{s}^{k+1},\boldsymbol{a}^{k+1}\right)-Q_{\boldsymbol{w}}\left(\boldsymbol{s}^{k},\boldsymbol{a}^{k}\right)\right]^{2}. \tag{11}$$

4 Solution with DQN

DQN is an improvement on Q-learning, which replaces Q-value function with a deep neural network. Additionally, experience replay and target network are introduced in the DQN, making the neural network more stable and well-trained.

4.1 Target Network

In order to make the performance of the algorithm more stable, two structurally identical neural networks are established: a network with continuously updated network parameters (evaluation network) and a neural network for target value update (target network). At the initial moment, the parameters of the evaluation network are assigned to the target network, and then the evaluation network continues to update the neural network parameters, while the parameters of the target network are fixed. After several rounds of updates for the evaluation network, the parameters of the evaluation network are assigned to the target network. The setting of the two networks makes the target Q value stable over a period of time, thereby improving the stability of the algorithm update.

Therefore, the loss function in Eq. (11) can be reexpressed as follows.

$$L(\boldsymbol{w}) = E\left[Q_t - Q_{\boldsymbol{w}}\left(\boldsymbol{s}^k, \boldsymbol{a}^k\right)\right]^2, \tag{12}$$

where Q_t is the Q-value generated in the target network and can be expressed as $Q_t = r^k + \gamma \max_{a^{k+1}} Q_{\boldsymbol{w}^t}\left(\boldsymbol{s}^{k+1}, \boldsymbol{a}^{k+1}\right)$. $\boldsymbol{w}^t$ is the weight parameters in the target network.

When the loss function is continuously differentiable with respect to parameters $\boldsymbol{w}$, the parameters $\boldsymbol{w}$ of the evaluation network can be updated with the gradient of the loss function. Therefore the update of $\boldsymbol{w}$ is as follows:

$$\Delta \boldsymbol{w} = \alpha_l \left[Q_t - Q_{\boldsymbol{w}}\left(\boldsymbol{s}^k, \boldsymbol{a}^k\right)\right] \nabla_{\boldsymbol{w}} Q_{\boldsymbol{w}}\left(\boldsymbol{s}^k, \boldsymbol{a}^k\right), \tag{13}$$

where α_l is the learning rate of the evaluation network.

4.2 Experience Replay

In DNN, the data used to train the neural network needs to be guaranteed be independent. However, the data sampled from each episode is related to each other. The experience replay is proposed to break up the temporal correlations within different data in the episode. The main idea of the experience replay is to store the agent's own experience from different episodes and then build a data set to train the neural network.

The data stored in the replay buffer is a tuple of $(\boldsymbol{s}^k, \boldsymbol{a}^k, r^k, \boldsymbol{s}^{k+1})$. DNN samples a mini-batch of I tuples from the replay buffer and utilizes the stochastic gradient descent (SGD) method to update parameters. Therefore, the parameters $\boldsymbol{w}$ updated in Eq. (13) can be reexpressed as follows:

$$\Delta \boldsymbol{w} = \alpha_l \frac{1}{I} \sum_{i=1}^{I} \left[Q_t^i - Q_{\boldsymbol{w}}\left(\boldsymbol{s}_i^k, \boldsymbol{a}_i^k\right)\right] \nabla_{\boldsymbol{w}} Q_{\boldsymbol{w}}\left(\boldsymbol{s}_i^k, \boldsymbol{a}_i^k\right), \tag{14}$$

where $Q_t^i = r_i^k + \gamma \max_{a^{k+1}} Q_{\boldsymbol{w}^t}\left(\boldsymbol{s}_i^{k+1}, \boldsymbol{a}^{k+1}\right)$. The index i refers to the ith sample.

4.3 DQN Based Resource Allocation Algorithm

The architecture of DQN is depicted in Fig. 2. In the process of reinforcement learning, the interaction between the agent and environment generates a large amount of data like $(\boldsymbol{s}^k, \boldsymbol{a}^k, r^k, \boldsymbol{s}^{k+1})$, which are stored in the replay buffer. The data set in replay buffer D is used to train the evaluation neural network.

The main steps of the DQN based resource allocation algorithm with target network and experience replay are as follows.

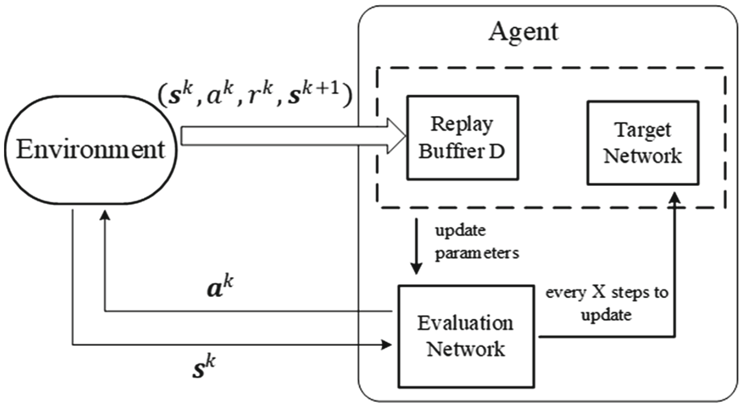

Fig. 2. The architecture of DQN

(1) The evaluation network initializes the parameters. The target network is built with the same parameters as that of the evaluation network. The replay buffer is cleared.
(2) The agent observes the state $\boldsymbol{s}^k$ of the environment and generates action $\boldsymbol{a}^k$ according to the current ϵ-greedy policy.
(3) The agent observes the next state $\boldsymbol{s}^{k+1}$ and the reward r^k. The transition $(\boldsymbol{s}^k, \boldsymbol{a}^k, r^k, \boldsymbol{s}^{k+1})$ is stored in the experience replay buffer.
(4) Sample random mini-batch of I tuples from replay buffer.
(5) The evaluation network is updated with the mini-batch from replay buffer according to Eq. (14).
(6) The target network is updated with the parameters of the evaluation network every X steps.

The specific algorithm is shown in Algorithm 1.

5 Simulation Results

In order to verify the feasibility and efficiency of tasks offloading algorithm based on deep reinforcement learning, we use the task execution rate and waiting time as the evaluation indicators of the simulation experiment. We define task execution rate as follows.

$$ER_n = \frac{O_n}{O_{total}}, \tag{15}$$

where O_n, O_{total} indicate that the processed data size offloaded by U_n and the size of data generated by U_n in a certain period of time. All parameters are provided in Table 1.

For convenience, in Figs. 3, 4 and 5, the number of CPU and UN is 5 and 3, respectively. And the priority of U_1, U_2, U_3 is set as 0.5, 0.3, 0.2.

Figure 3 compares the performance of the random allocation scheme, the Fair and Energy-Minimized Task Offloading (FEMTO) [10] scheme and our

Algorithm 1. DQN based Resource Allocation Algorithm

```
1: Initialize Q-value function with random weights $\boldsymbol{w}$
   Initialize target Q-value function with weights $\boldsymbol{w}^- \leftarrow \boldsymbol{w}$
   Initialize replay memory D
2: for episode=1 to $E_{\max}$ do
3:    Reset environment state $\boldsymbol{s}^1$.
4:    for $k = 1$ to $k_{\max}$ do
5:       With probability $\epsilon$ select a random action $\boldsymbol{a}^k$,
         otherwise select $\boldsymbol{a}^k = \arg\max_{\boldsymbol{a}} Q_{\boldsymbol{w}}\left(\boldsymbol{s}^k, \boldsymbol{a}^k\right)$.
6:       Execute $\boldsymbol{a}^k$, observe reward $r^k$ and next state $\boldsymbol{s}^{k+1}$.
7:       Store $(\boldsymbol{s}^k, \boldsymbol{a}^k, r^k, \boldsymbol{s}^{k+1})$ in D.
8:       Sample random mini-batch of $I$ tuples from D.
9:       Update the parameters of the evaluation network:
         $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_l \frac{1}{I} \sum_{i=1}^{I} \left[Q_t^i - Q_{\boldsymbol{w}}\left(\boldsymbol{s}_i^k, \boldsymbol{a}_i^k\right)\right] \nabla_{\boldsymbol{w}} Q_{\boldsymbol{w}}\left(\boldsymbol{s}_i^k, \boldsymbol{a}_i^k\right)$.
10:      Every X steps, update target network parameters:
         $\boldsymbol{w}^- \leftarrow \boldsymbol{w}$.
11:   end for
12: end for
```

Table 1. Simulation parameters

Parameter	Value
Discount factor γ	0.9
Capacity of replay memory D	300
Number of CPU cores M	4–10
Number of UNs N	2–8
Time slot t_s	5μs
Mini-batch I	30
Weight coefficiency α	2
Weight coefficiency β	5
Computational capacity C	200bit/s
Explore probability ϵ	0.1
Learning rate of network α_l	0.01
Priority of UN ζ_n	0–1

proposed scheme. For the random allocation scheme, the fog node randomly assigns the computing resources of the node in each step size. For the FEMTO schemeit uses the fair scheduling metric mechanism to optimize the allocation of resources with the objective of fair and energy minimization. As shown in Fig. 3, the average task execution rate of our proposed algorithm is significantly higher than the random allocation. In addition, as total step increases, the average task execution rate of our proposed algorithm increases gradually while the execution rate of the FEMTO scheme decreases.

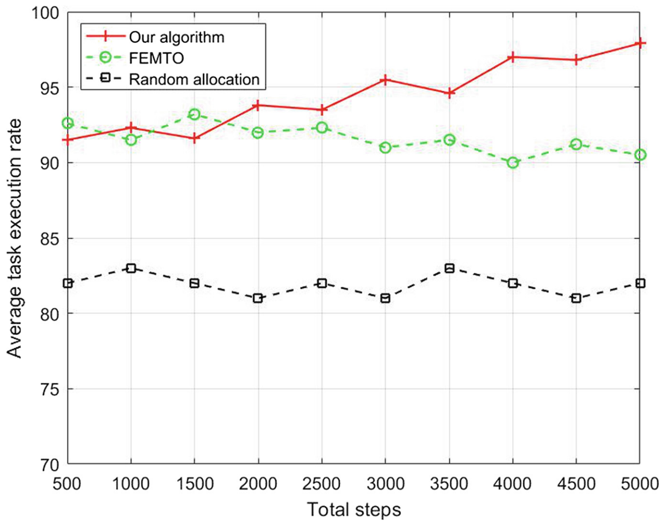

Fig. 3. Comparison between our algorithm and random allocation.

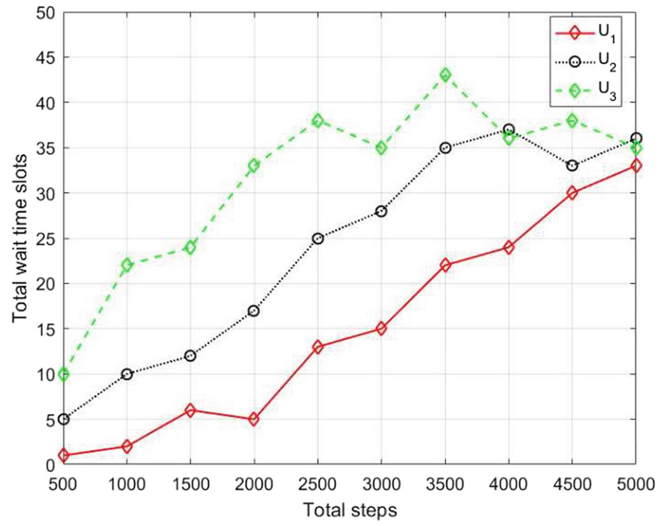

Fig. 4. Relationship between steps and waiting time in different UNs.

As shown in Fig. 4, U_1 with the highest user priority has the least total number of tasks waiting, and U_3 with the lowest user priority has the most waiting time slots, namely, U_3 has the highest probability of not being allocated computing resources in the time slot. As total step increases, the waiting time slots of UNs increase gradually. After reaching a certain peak value, the growth slows down, and the number of waiting time slots of different users gradually approaches.

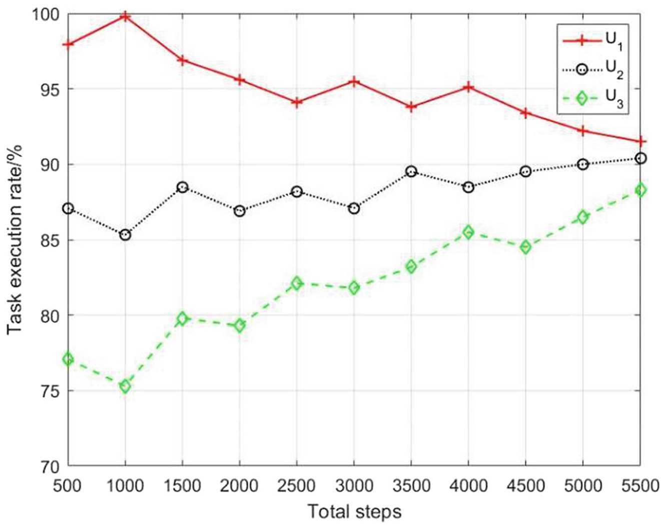

Fig. 5. Relationship between steps and task execution rate in different UNs.

In Fig. 5, U_1 has the highest task execution rate and U_3 has the lowest at the initial moment. As the total number of time slots increases, the task execution rate of different UNs gradually approach. The analysis demonstrates that the task with high priority has a high task execution rate, namely, the task with higher priority is allocated computational resources firstly for task processing such that the task is processed faster. The task with higher priority has higher service satisfaction. As the number of time slots increases, the accumulation of data tasks from UN with low priority and the waiting time of tasks increase such that the task priority of them gradually increases, resulting that the priority of UN has little effect on task execution. From Figs. 4 and 5, we observe that whatever the priority of the UN is, the computational resources are allocated fairly finally, which results in the increment of the average task execution rate in Fig. 3. Additionally, the fluctuation of the curves in Figs. 4 and 5 indicates the learning process of reinforcement learning.

As shown in Fig. 6, when the number of UNs increases, the total waiting time slots of each UN increases obviously. Additionally, we observe that when the number of UNs is twice more than that of CPU cores, the waiting time increases rapidly, which reduces the performance of our proposed algorithm. Therefore, it will be better that there exist enough CPU cores to schedule.

In Fig. 7, we can observe the trend of loss function $L(\boldsymbol{w})$. As the training steps increase, the loss function tends to converge, which satisfies our expectation.

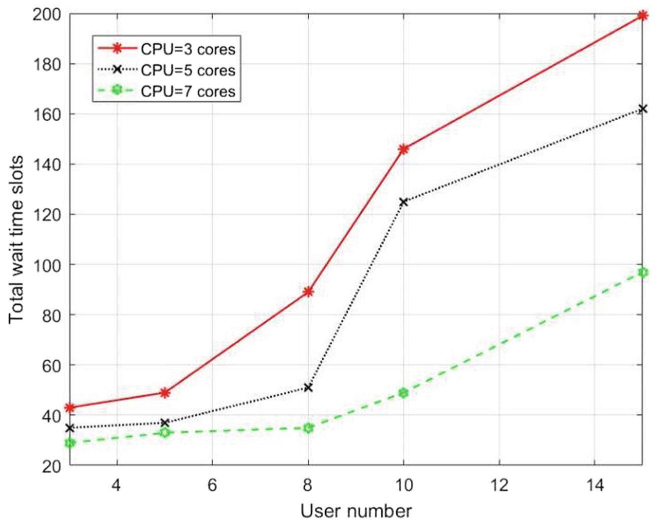

Fig. 6. Relationship between the number of UNs and waiting time with different CPU cores.

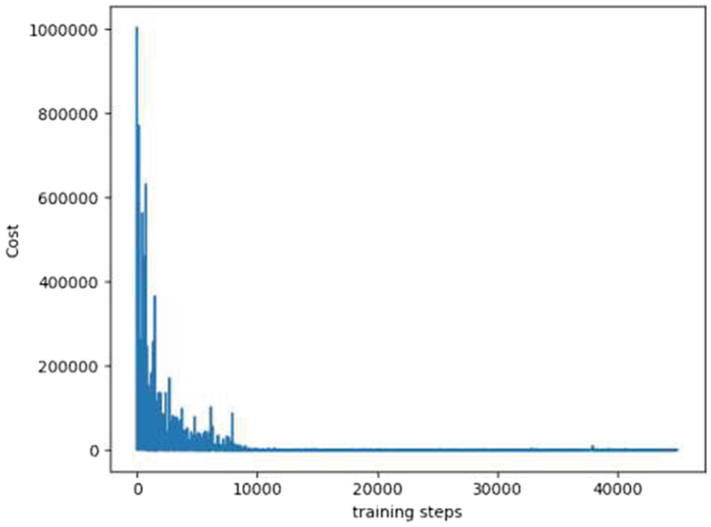

Fig. 7. Trend of loss function.

6 Conclusion

In this paper, we applied deep reinforcement learning to allocate computational resource in fog networks. At the beginning of each time slot, the computational resource is reallocated to process the data tasks. We transformed the process into a Markov decision process. And then, we introduced the state, reward and action of the deep reinforcement learning and explained the specific algorithm of DQN in detail. Subsequently, the analysis of updating the neural network was provided. At last, we analysed the task execution rate and total waiting time slots of each UN. The simulation results demonstrated the fairness and effectiveness of our proposed strategy.

References

1. Abbas, N., Zhang, Y., Taherkordi, A., Skeie, T.: Mobile edge computing: a survey. IEEE Internet Things J. **5**(1), 450–465 (2018). https://doi.org/10.1109/JIOT.2017.2750180
2. Chiang, M., Zhang, T.: Fog and IoT: an overview of research opportunities. IEEE Internet Things J. **3**(6), 854–864 (2016). https://doi.org/10.1109/JIOT.2016.2584538
3. da Silva, R.A.C., da Fonseca, N.L.S.: Resource allocation mechanism for a fog-cloud infrastructure. In: 2018 IEEE International Conference on Communications (ICC), pp. 1–6, May 2018. https://doi.org/10.1109/ICC.2018.8422237
4. Ghazy, A.S., Selmy, H.A.I., Shalaby, H.M.H.: Fair resource allocation schemes for cooperative dynamic free-space optical networks. IEEE/OSA J. Opt. Commun. Netw. **8**(11), 822–834 (2016). https://doi.org/10.1364/JOCN.8.000822
5. Gu, Y., Chang, Z., Pan, M., Song, L., Han, Z.: Joint radio and computational resource allocation in IoT fog computing. IEEE Trans. Vehicular Technol. **67**(8), 7475–7484 (2018). https://doi.org/10.1109/TVT.2018.2820838
6. Jia, B., Hu, H., Zeng, Y., Xu, T., Yang, Y.: Double-matching resource allocation strategy in fog computing networks based on cost efficiency. J. Commun. Netw. **20**(3), 237–246 (2018). https://doi.org/10.1109/JCN.2018.000036
7. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
8. Ozmen, M., Gursoy, M.C.: Wireless throughput and energy efficiency with random arrivals and statistical queuing constraints. IEEE Trans. Inf. Theory **62**(3), 1375–1395 (2016). https://doi.org/10.1109/TIT.2015.2510027
9. Sarigl, M., Avci, M.: Performance comparision of different momentum techniques on deep reinforcement learning. In: 2017 IEEE International Conference on INnovations in Intelligent SysTems and Applications (INISTA), pp. 302–306, July 2017. https://doi.org/10.1109/INISTA.2017.8001175
10. Zhang, G., Shen, F., Liu, Z., Yang, Y., Wang, K., Zhou, M.: FEMTO: fair and energy-minimized task offloading for fog-enabled IoT networks. IEEE Internet Things J. **6**(3), 4388–4400 (2019). https://doi.org/10.1109/JIOT.2018.2887229