Decoupling Offloading Decision and Resource Allocation via Deep Reinforcement Learning and Sequential Least Squares Programming

Zhihao Xuan[1(✉)], Guiyi Wei[1], Zhengwei Ni[1], and Jifa Zhang[2]

[1] Zhejiang Gongshang University, Hangzhou 310018, China
xuanzhihao1997@outlook.com,
{weigy,zhengwei.ni}@zjgsu.edu.cn
[2] Zhejiang University, Hangzhou 310027, China
jfzhang@zju.edu.cn

Abstract. Edge computing is to generate faster network service response and meet the basic needs of the industry in real-time business, application intelligence, security and privacy protection. This paper studies the mobile edge computing network, where the computing power of the edge server (ES) is limited, and multiple user equipment (UE) can offload the thinking to the ES in order to save energy consumption and computing delay. The ES needs to determine which UEs can upload its tasks and need to allocate computing resources for these UEs, so this problem is highly coupled and difficult to calculate. This paper proposes an algorithm based on deep reinforcement learning and Sequential Least Squares Programming (SLSQP), which decouples and solves the problem. Experiments show that the algorithm works well and can be dynamically adjusted according to environmental changes. The comparison with other algorithms also proves that the algorithm has better results and less time-consuming.

Keywords: Edge computing · Offloading · Resource allocation · Reinforcement learning

1 Introduction

1.1 A Subsection Sample

In recent years, with the rapid development of IoT technology and the increasing requirements of software for hardware, IoT devices are facing more and more computer-intensive and delay-sensitive tasks. However, IoT devices are often constrained by their size and power. Factors such as these are not suitable for running these tasks on the device itself.

Edge computing can offload the computing tasks required by the user equipment to the edge computing server with rich computing resources for calculation, in order to reduce the energy consumption of the user equipment and the delay of the computing tasks. Compared with the existing cloud computing, the edge computing server is deployed on the edge of the network, such as a base station or an access point of a

H. Gao et al. (Eds.): ChinaCom 2020, LNICST 352, pp. 554–566, 2021.
https://doi.org/10.1007/978-3-030-67720-6_38

wireless network, and is closer to the user, which can avoid long-distance data transmission from the cloud computing center, thereby reducing the time delay and transmission energy required for computing tasks, and improves the user experience.

However, unlike cloud computing, edge computing servers usually have limited computing power and bandwidth. Therefore, offloading decisions and resource allocation for user equipment has become a hot research issue and a difficulty in edge computing systems. In the case of limited resources, unreasonable judgment and allocation of unloading decisions and resources may lead to increased delay and energy consumption, and may also make the system load unbalanced, affecting the stability of the system.

In order to solve the above problems, many scholars have conducted research in this area. [1] proposes a mobile edge computing scheduling decision algorithm based on multi-round auctions. Its implementation requires multiple rounds of communication between the user equipment and the edge server. There are also some studies to solve the above problems through game theory [2], the method still requires multiple rounds of communication between the device and the server. However, the complexity and communication time are still too high for tasks with delay sensitivity in current edge computing system. There are also many studies that conduct separate studies on computing offloading or resource allocation [3, 4]. However, the two issues should be considered and solved jointly due to embroiled relation between them.

In recent years, machine learning methods have made some breakthroughs in different fields, such as natural language processing [5], data mining [6] and other fields. However, at present, there are few researches on edge computing systems using machine learning methods, and most of the research is still based on Q learning [7], but the table search structure inside Q learning is actually not suitable for multidimensional and highly coupled problems. Some scholars have also used deep learning methods to carry out research on offload decision-making [8], but they require supervised learning of neural network training, and the results cannot adapt to changes in various conditions.

This paper proposes an offloading decision and resource allocation method based on reinforcement learning and convex optimization for the goal of task completion delay and energy consumption optimization in the case of edge computing servers with limited computing power. This method decouples the highly coupled problem of offloading decision and resource allocation into two sub-problems of offloading decision and resource allocation, and uses reinforcement learning and convex optimization methods to solve them. Experiments show that this method has achieved good results, can make decisions in a short time, and has good scalability.

2 System Model and Problem Description

2.1 System Model

The template is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them.

The structure of the edge computing system in this article is shown in Fig. 1, which includes an edge computing server (ES) and N user equipments (UE). These N UEs are the same, denoted by $N = \{1, 2, \ldots, N\}$. The ES and UE are connected through a wireless network, and the transmission delay between them can be ignored. Generally speaking, ES has a stable power supply and fast calculation speed, while UE is the opposite. Therefore, UE can offload the tasks it needs to calculate to ES for calculation, and then receive the calculation result from ES to reduce the energy consumption and time delay required by calculating the task. But for ES, it has limited computing resources $F_{\max}^{es}$, so it needs to allocate computing resources for the UE that decides to perform task offloading. The allocation of computing resources will affect energy consumption and delay. In this article, a binary offloading strategy is used. For a certain UE, all its tasks are either calculated locally or offloaded to ES for calculation. The decision to calculate offloading is represented by a binary variable $x_i \in \{0, 1\}$. Specifically, when $x_i = 0$, it means that the i-th user equipment decides to perform calculations locally, and when $x_i = 1$, it means that the i-th user equipment decides to perform task offloading.

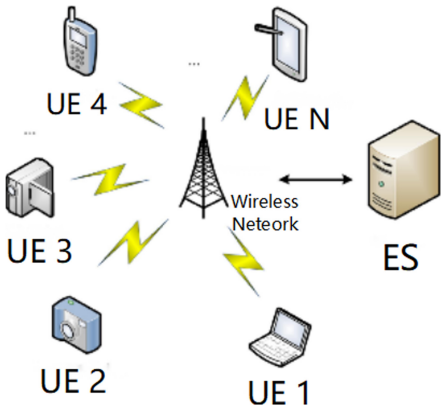

Fig. 1. Edge computing model.

2.2 Task Offloading

The template is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them.

When the UE_i decides to offload all its tasks to the ES for calculation, it first needs to upload data of size d_i to the ES via the wireless network, and suppose the uplink speed of the UE_i and ES direct wireless network in the system model is r_i.

In general, the amount of data in the calculation results is very small and can be ignored compared to the amount of uploaded data. Therefore, this article ignores the energy consumption and time delay caused by downloading the calculation results. The same assumption also exists in [9].

First, model the time required to unload the task. The time it takes for UE_i to upload data of size d_i to ES under the condition of transmission speed r_i is:

$$T_i^t = \frac{d_i}{r_i} \tag{1}$$

The time required for computing in the ES server is:

$$T_i^c = \frac{d_i}{f_i^{es}} \tag{2}$$

Where f_i^{es} represents the computing resources allocated by the ES server to the UE_i, that is, the amount of data that can be processed per unit time. In summary, the total time consumed for task uninstallation is:

$$T_i^{offload} = T_i^t + T_i^c \tag{3}$$

Next, model the energy consumption of task offloading. Assuming that the upload power of each UE is p_{upload}, the energy consumed during the upload task is:

$$E_i^{upload} = p_{upload} * T_i^t \tag{4}$$

From the task is uploaded to the ES to the calculation result is obtained, the UE_i will be in a waiting state. Assuming that the waiting state power of all user equipment during this period is $p_{waiting}$, then the energy consumed by the UE_i during this period is:

$$E_i^{waiting} = p_{waiting} * T_i^c \tag{5}$$

Then the total energy consumed when perform offloading is:

$$E_i^{offload} = E_i^{upload} + E_i^{waiting} \tag{6}$$

2.3 Local Computing

Now to model the delay and energy consumption of local calculations, we use f_{local} to represent the amount of data that the UE_i itself can process per second, so for the UE, the time required for local calculations is:

$$T_i^{local} = \frac{d_i}{f_{local}} \tag{7}$$

Next, model the energy consumption of the local calculation. Assuming that the operating power when computing locally is p_{local}, then the energy consumed is:

$$E_i^{local} = T_i^{local} p_{local} \tag{8}$$

2.4 Problem Description

In order to minimize the delay and energy consumption of all user equipment, this paper uses a linear weighting method to define a weighted function $Cost(\mathbf{d}, \mathbf{x}, \mathbf{r})$ composed of delay and energy consumption to evaluate the performance of the system:

$$Cost(\boldsymbol{d}, \boldsymbol{x}, \boldsymbol{r}) = \sum_{i=1}^{N} \{(1 - x_i)\left(T_i^{local} + \alpha E_i^{local}\right) + x_i\left(T_i^{offload} + \alpha E_i^{offload}\right)\}$$

Where $\boldsymbol{d} = \{d_i | i \in \mathrm{N}\}$,$x = \{x_i | i \in \mathrm{N}\}$,$\boldsymbol{r} = \{r_i | i \in \mathrm{N}\}$, α represents the weight of energy consumption in the weighting function. Then the question is:

$$Q(\boldsymbol{d}) = \text{minimize } \mathrm{Q}(\mathbf{d}, \mathbf{x}, \mathbf{r}) \tag{9a}$$

Subject to:

$$\sum_{i=1}^{N} r_i \leq R_{\max} \tag{9b}$$

$$r_i \geq 0, \forall i \in N \tag{9c}$$

$$x_i \in \{0, 1\} \tag{9d}$$

3 Algorithms Based on Deep Reinforcement Learning and Convex Optimization

For problem (9a), it is a highly coupled problem to find the offloading decision and the allocation of computing resources at the same time. There are multiple parameters that affect each other. Generally speaking, it is difficult to solve such problems, so we decouple the problem, Decomposed into the problem of offloading decision-making and the problem of computing resource allocation.

For the generation of offloading decisions, it is necessary to take the amount of data **d** required to be computed for all devices as input, and find the most appropriate offloading strategy x considered by the system. For N devices, there are a total of 2^N types of offloading strategies. If the brute force search method is used in actual situations, the number of selectable offloading strategies will increase exponentially with the increase of the number of devices N. Therefore, the brute force search method is not applicable in actual situations. In order to solve the above problems, this paper uses a method based on deep reinforcement learning to generate unloading decisions.

The structure of deep reinforcement learning in this article is shown in Fig. 2. It roughly contains two modules, namely the offloading decision generation module and the improvement module for offloading decision-making. The offloading decision generation module contains the solution to the problem of computing resource allocation.

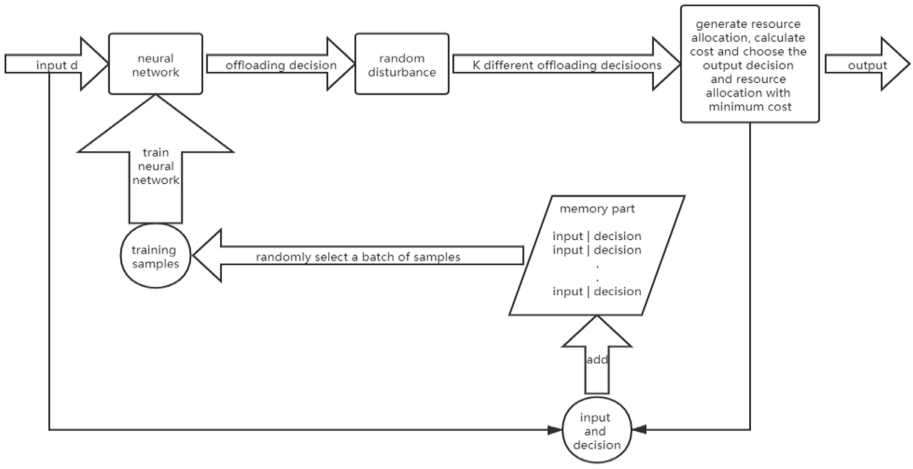

Fig. 2. Overview of our algorithm.

Among them, the offloading decision generation module contains a neural network model, which will generate an offloading decision x according to the input task data $\mathbf{d}$ that each user needs to compute. For this neural network, according to the well-known universal approximation theorem, for a feedforward neural network, even if there is only one hidden layer, it can infinitely approximate any bounded continuous function, so here, we use ReLU function as the hidden layer activation function, and in the output layer, the Sigmoid activation function is used.

After an unloading decision is generated, the system will randomly perturb the unloading decision. Each disturbance will randomly select c users for the unloading decision. If the user's decision is to uninstall, it will be calculated locally, and vice versa. A total of K different unloading decisions are generated.

When a different offloading decision is generated, problem (9a) becomes a computing resource allocation problem:

$$Q(\mathbf{d}, \mathbf{x}) = \text{minimize } \mathrm{Q}(\mathbf{d}, \mathbf{x}, \mathbf{r}) \tag{10a}$$

Subject to:

$$\sum_{i=1}^{N} r_i \leq R_{\max} \tag{10b}$$

$$r_i \geq 0, \forall i \in N \tag{10c}$$

$$x_i \in \{0, 1\} \tag{10d}$$

According to these K different offloading decisions, the system calculates the optimal computing resource allocation and the corresponding cost according to the convex optimization scheme and cost function, and outputs the offloading decision with the lowest cost.

In view of the fact that the predecessors have carried out extensive and in-depth research on convex optimization, there are many efficient algorithms in the current research on convex optimization. For the problem of computing resource allocation, this paper uses a convex optimization algorithm named "SLSQP". Method SLSQP uses Sequential Least Squares Programming to minimize a function of several variables with any combination of bounds, equality and inequality constraints. The method wraps the SLSQP Optimization subroutine originally implemented by Dieter Kraft [10].

As for the improved module for offloading decision-making, every time the offloading decision-making module produces the optimal offloading decision, the offloading decision and the calculated task data required by the user will be saved as a sample in the memory. The storage capacity is limited. If the memory is full when the new optimal unloading decision is added, the sample that was added to the memory earliest will be eliminated.

In the improved module for offloading decision-making, every time the offloading decision is generated, if the current number of decisions is exactly a multiple of δ, a batch of samples will be randomly selected from the memory for training, instead of the traditional method Use all data for training. In the training process, this article uses Adam algorithm to update the parameters of the neural network to reduce the average cross entropy loss. Because it uses its own decisions instead of the optimal decisions obtained, the neural network model can continuously improve its own model without supervision and produce better offloading decisions, unlike traditional deep learning that requires optimal offloading decisions for learning.

In particular, limited memory capacity can help improve the efficiency of training, because the newly generated samples are generally better than the old samples. In fact, there are still some other techniques that can help accelerate training, such as distributed importance sampling [11] And priority experience replay [12].

4 Performance Evaluation

In this section, we use simulation experiments to evaluate the performance of the proposed algorithm. In the simulation experiment, we set the number of user equipment N = 10. The speed of computing locally or computing in the server is determined regardless of the task type. The task data volume of each user equipment is uniformly distributed between 5 MB and 25 MB. Edge computing server can process 30 MB of data per second. The communication speed between the user equipment and the edge computing server is 2.5 MB/s. The user's power when sending data is 1 W, and the power when waiting is 0.5 W, while the user's local computing The data processing speed is 3 MB/s, the computing power is 3 W, and the coefficient of energy consumption in the cost function is $\alpha = 1$; we set the capacity of the memory module to 256 and the neural network model to four layers, each time $K = 7$ offloading strategies will be generated, the number of randomly disturbed users is $c = 4$, one learning is performed every 10 predictions, 128 samples are randomly selected from the memory module for each learning, and the learning rate is 0.01. For neural networks, this paper uses a fully connected network consisting of one input layer, two hidden layers, and one output layer. Two hidden layers contain 120 and 80 neurons, respectively. The convex optimization "SLSQP" method is implemented using the corresponding function in the scipy library.

4.1 Convergence Performance

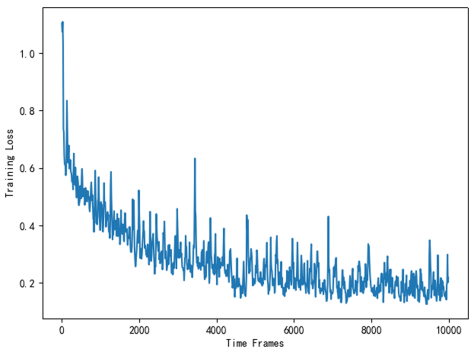

Fig. 3. Training losses in iterations.

Figure 3 and Fig. 4 are the training loss of the neural network and the ratio of the minimum cost in the iteration to the current cost, respectively. It can be seen from Fig. 3 that the training loss has been continuously reduced from the beginning, and a total of 10,000 iterations have been carried out. When the number of iterations reaches about 4500, the loss basically stabilizes, and the rate of decline slows down significantly. When it reaches about 6500, It basically no longer declines, maintaining between 0.1–0.2. Figure 4 shows the ratio of the minimum cost to the cost of the

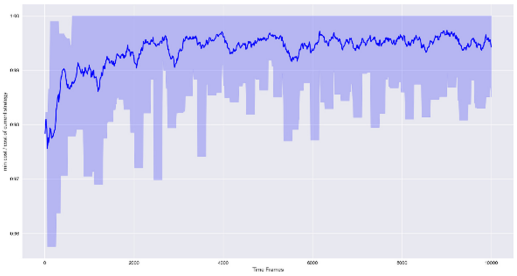

Fig. 4. The ratio of the minimum cost in an iteration to the current policy cost. (Color figure online)

current strategy in the iteration. The minimum cost is obtained by finding all kinds of offloading strategies, and then generating the resource allocation of each strategy according to the "SLSQP" convex optimization algorithm and calculating its cost, and finally taking the smallest cost one; the ratio in Fig. 4 is recorded every 10 iterations, the blue line represents the average ratio of the nearly 20 records, and the blue shading represents the maximum and minimum values of the nearly 20 records; you can see from the figure It can be seen that after 4000 iterations, the ratio basically stabilizes, changing around 0.995.

4.2 Situation When Parameters Change

Figures 5 and 6 are respectively the training loss and the ratio of the minimum cost in the iteration to the current cost when α changes. In particular, when the number of iterations is 5000, α will become 3, and when the number of iterations is 7000, a will Change back to 1.It can be found that the training loss in Fig. 5 has increased significantly at 5000 and 7000 times, and then quickly converged, while the ratio in Fig. 6 is also the same, there has been a significant decline, and both are resumed after about 1000 iterations.

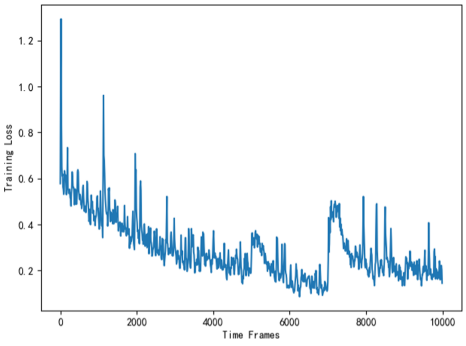

Fig. 5. Training losses in iterations (Change α).

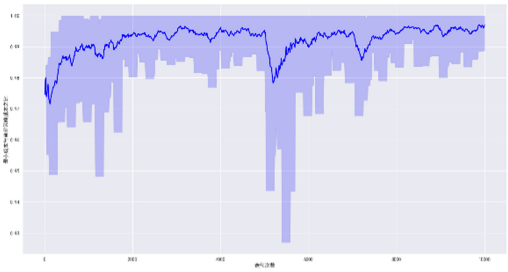

Fig. 6. The ratio of the minimum cost in an iteration to the current policy cost (Change α).

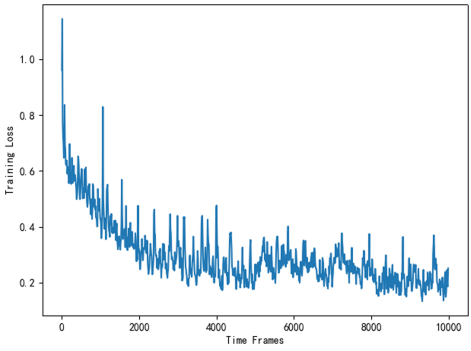

Fig. 7. Training losses in iterations (User equipment switch on and off).

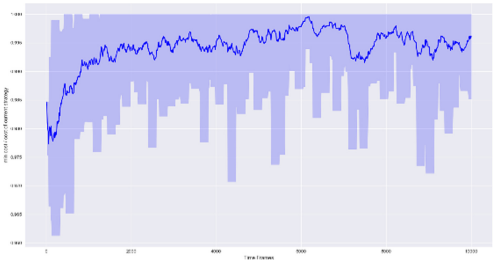

Fig. 8. The ratio of the minimum cost in an iteration to the current policy cost (User equipment switch on and off).

Figures 7 and 8 are respectively the training loss and the ratio of the minimum cost in the iteration to the current cost when shutting down and turning on three user devices when the number of iterations is 5000 and 7000. It can be seen that the impact of switching on and off the user equipment is less than the impact of a change on α. After a small increase in loss and a small decrease in cost ratio, it quickly returns to the previous level.

It can be seen from the above two experiments that the model can quickly adapt to changes in the environment.

4.3 Comparison with Other Algorithms

Wherever Times is specified, Times Roman or Times New Roman may be used. If neither is available on your word processor, please use the font closest in appearance to Times. Avoid using bit-mapped fonts if possible. True-Type 1 or Open Type fonts are preferred. Please embed symbol fonts, as well, for math, etc.

This section compares the proposed algorithm with the following strategies, which are:

ALL_LOCAL: All user devices calculate tasks locally.

ALL_OFFLOAD: All user devices offload tasks to the edge computing server for calculation, and all user devices are equally offloaded.

CD_AVERAGE: The offloading strategy is calculated by the gradient descent method [9], and the offloading resources are equally divided among all user devices.

CD_SLSQP: The offloading strategy is calculated by the gradient descent method, and the computing resources are allocated using the "SLSQP" algorithm.

This paper compares the above algorithms for 10,000 iterations, and the comparison results are shown in Fig. 9.

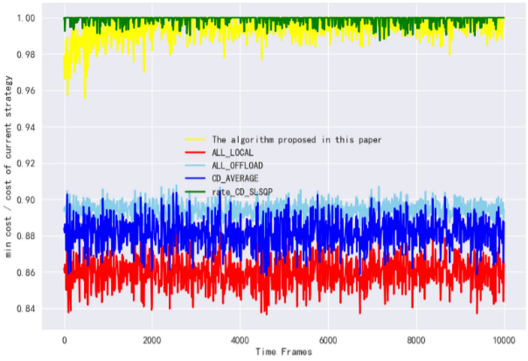

Fig. 9. Results compared with other algorithms

It can be seen from Fig. 9 that ALL_LOCAL, ALL_OFFLOAD, and CD_AVERAGE perform poorly, and the ratio of the minimum cost to the current strategy cost is between 0.9 and 0.84; and the CD_SLSQP algorithm performs best, with the cost ratio closest to 1, which is stable at Around 0.998, and the algorithm proposed in this paper, after a period of convergence, the cost ratio stabilizes at around 0.995.

Table 1. Time comparison between the proposed algorithm and CD_SLSQP

Algorithm	Number of UE		
	10	15	20
Algorithm this paper proposed	0.069 s	0.174 s	0.435 s
CD_SLSQP	0.470 s	2.247 s	9.333 s

It can be seen from Table 1 that the time consumed by the algorithm proposed in this paper to generate an offloading strategy and computing resource allocation plan is much smaller than that of the CD_SLSQP algorithm, but the number of user devices is 10, 15 and 20, respectively. The time-consuming of CD_SLSQP is 6.8, 12.9, and 21.5 times of the algorithm proposed in this paper. It can be seen that the time complexity of CD_SLSQP algorithm is relatively high, and the time consumed will increase greatly when the number of user equipment increases.

5 Conclusion

In this paper, a computational offload decision and resource allocation algorithm based on deep reinforcement learning and convex optimization is proposed to solve a highly coupled problem. The results show that the proposed algorithm has good performance and less time consumption. We also think that this method has good extensibility, and I can easily use this algorithm to solve other similar problems.

References

1. Zhang, H., Guo, F., Ji, H., Zhu, C.: Combinational auction-based service provider selection in mobile edge computing networks. IEEE Access **5**, 13455–13464 (2017). https://doi.org/10.1109/ACCESS.2017.2721957
2. Messous, M., Sedjelmaci, H., Houari, N., Senouci, S.: Computation offloading game for an UAV network in mobile edge computing. In: 2017 IEEE International Conference on Communications (ICC), Paris, pp. 1–6 (2017). https://doi.org/10.1109/icc.2017.7996483
3. Sun, W., Liu, J., Yue, Y., Zhang, H.: Double auction-based resource allocation for mobile edge computing in industrial Internet of Things. IEEE Trans. Industr. Inf. **14**(10), 4692–4701 (2018). https://doi.org/10.1109/TII.2018.2855746.minutes
4. Chen, M., Hao, Y.: Task offloading for mobile edge computing in software defined ultra-dense network. IEEE J. Sel. Areas Commun. **36**(3), 587–597 (2018). https://doi.org/10.1109/JSAC.2018.2815360
5. Otter, D.W., Medina, J.R., Kalita, J.K.: A survey of the usages of deep learning for natural language processing. IEEE Trans. Neural Netw. Learn. Syst., 1–21 (2020). https://doi.org/10.1109/tnnls.2020.2979670
6. Atluri, G., Karpatne, A., Kumar, V.: Spatio-temporal data mining: a survey of problems and methods. ACM Comput. Surv. **51**(4), 41 (2018). Article 83. https://doi.org/10.1145/3161602
7. Wang, X., Wang, C., Li, X., Leung, V.C.M., Taleb, T.: Federated deep reinforcement learning for Internet of Things with decentralized cooperative edge caching. IEEE Internet Things J. **7**, 9441–9455 (2020). https://doi.org/10.1109/jiot.2020.2986803

8. Chen, J., Ran, X.: Deep learning with edge computing: a review. Proc. IEEE **107**(8), 1655–1674 (2019). https://doi.org/10.1109/JPROC.2019.2921977
9. Bi, S., Zhang, Y.J.: Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. IEEE Trans. Wireless Commun. **17**(6), 4177–4190 (2018). https://doi.org/10.1109/TWC.2018.2821664
10. Kraft, D.: A software package for sequential quadratic programming. 1988. Tech. rep. DFVLR-FB 88-28, DLR German Aerospace Center – Institute for Flight Mechanics, Koln, Germany
11. Loshchilov, I., Hutter, F.: Online batch selection for faster training of neural networks (2015). arXiv:1511.06343
12. Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., van Hasselt, H., Silver, D.: Distributed prioritized experience replay (2018). arXiv:1803.00933