Synchronous Federated Learning Latency Optimization Based on Model Splitting

Chen Fang[1,3], Lei Shi[1,3(✉)], Yi Shi[2], Jing Xu[1,3], and Xu Ding[1,3]

[1] School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230009, China
shilei@hfut.edu.cn

[2] Department of ECE, Virginia Tech, Blacksburg, VA 24061, USA

[3] Engineering Research Center of Safety Critical Industrial Measurement and Control Technology, Ministry of Education, Hefei 230009, China

Abstract. Federated Learning (FL) is a distributed machine learning approach which is suitable for edge computing environment. While in this environment, how to take full advantage of the computing resources on end devices and edge servers is still a difficult problem. Especially for the synchronous federated learning, computing resources among different participants will lead to extra time cost and cause resource waste. In this paper, we try to reduce the time cost and the computing resource waste by using model splitting and task scheduling. We first establish the mathematical model and find it can not be solved directly. Then we design our algorithm which we name as the Federated Learning Offloading Acceleration (FLOA) algorithm to obtain a sub-optimal solution. The FLOA algorithm first uses the Partition Points Selection method to reduce the size of the solution space, then proposes a task offloading method based on matching theory. Experiments and simulations show that compared to the other three calculation methods, the single iteration time is reduced by 47%, 28%, 14% under our algorithm in turn.

Keywords: Edge computing · Federated learning · Model splitting

1 Introduction

With the development and popularity of AI applications, it has become a trend for deploying AI applications on smart devices. The key to the deployment of AI applications is to use the rich data distributed on smart devices for training AI models. The data on smart devices contains a large amount of the user's privacy [1,2]. Traditional cloud computing requires these data on devices to be transferred to the cloud centre, which will lead a large communication burden

The Work is Supported by Major Science and Technology Projects in Anhui Province (202003a05020009).

L. Wang et al. (Eds.): WASA 2022, LNCS 13473, pp. 495–506, 2022.
https://doi.org/10.1007/978-3-031-19211-1_41

and threat the data privacy. To solve these problems, some scholars are trying to combine edge computing and Federated Learning (FL) to train AI models [3].

Edge computing refers to the enabling technologies allowing computation to be performed at the edge of the network and is much efficient to process the data at the edge of the network [4]. For example, in [5] and [6], authors use computing resources from end devices and edge servers to process data. FL is a distributed machine learning approach suitable for edge computing [7]. In FL, participants collaborate with each other to train a shared DNN model together by using their own local data without sharing them [8,9]. In detail, each participant first trains a shared DNN model by using its local private data, and then uploads the model parameters to the parameter server for aggregation to obtain a global model. This process can be iterated several times until the trained model achieves the desired accuracy. FL has two different types of iterations, synchronous and asynchronous. The synchronous one means that model aggregation occurs after all participants complete local computation [10].

There are already some researches on combining edge computing with FL. In [11], a multi-layer federated learning protocol called HybridFL is designed for the Mobile Edge Computing (MEC) architecture, HybridFL improves the FL training process significantly in terms of shortening the federated round length, speeding up the global model's convergence and reducing end device energy consumption. [12] proposes a new FL-based client selection optimization to balance the trade-off between energy consumption of the edge clients and the learning accuracy of FL. [13] introduces a novel Hierarchical Federated Edge Learning (HFEL) framework, further formulate a joint computation and communication resource allocation and edge association problem for device users under HFEL framework to achieve global cost minimization.

In addition, model splitting can also accelerate the training process while protecting data privacy [14]. By using model splitting technique, A DNN model can be split inside between two successive layers into two partitions and then be deployed on different locations without losing accuracy [15]. [16] uses model splitting in FL to protect data privacy by placing the first layer of the DNN model on the end device, so there is no need to transmit sample data.

Previous work has made some contributions in combining edge computing and FL. However, they mostly focus on improvements to FL framework or aggregation protocols, and few researches use model splitting in federated learning to reduce latency. They do not take full advantage of the computing resources of end devices and edge servers in edge computing. In addition, they also ignore the time cost and resource waste in synchronous FL caused by the difference in computing resources of end devices. In this paper, we make full use of computing resources in edge computing environment through model splitting and task scheduling, which reduces the time cost of synchronous federated learning. First, we build a mathematical model for federated learning under end-to-edge collaborative edge computing. Edge computing scenario is complex and the mathematical model is difficult to solve directly because of the many variables. We first reduce the size of the solution space by filtering the split points, and Federated Learning

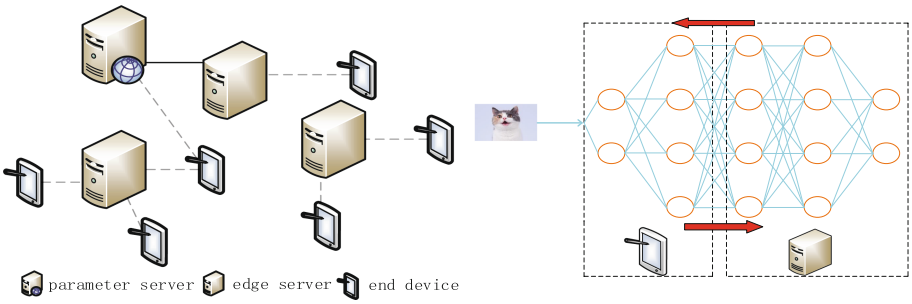


Fig. 1. System model.

Offloading Acceleration (FLOA) algorithm is designed based on matching theory to solve for the mathematical model.

The rest of this paper is organized as follows: In Sect. 2, we introduce system model and define our problem. In Sect. 3, we introduce the matching theory and the details of FLOA algorithm respectively. In Sect. 4, we give the simulation results and analyze them. In Sect. 5, we summarize this paper.

2 System Model and Problem Formulation

2.1 System Model

The system model is shown in Fig. 1. Suppose there are n end devices and m edge servers deployed in the scheduling network. Denote $e_i (i \in \{1 \dots n\}, e_i \in E)$ as one end device and $s_j (j \in \{1 \dots m\}, s_j \in S)$ as one edge server. Suppose all devices and servers are heterogeneous with different computing capabilities and suppose servers have better computing capabilities comparing with devices. Suppose there is a common DNN model needed to be trained in a distributed training manner by the whole network. Since we use model splitting technique, which means the DNN model will be split inside between two successive layers and then be trained separately on devices and servers. We also have a separate parameter server PS for updating parameters in the whole training process, devices and edge servers will send local training improvement to PS to obtain the global DNN model. The end devices are connected to edge servers and PS by wireless, the edge servers and PS are connected by wired.

Suppose we use the Stochastic Gradient Descent (SGD) algorithm to train the DNN model. Suppose the DNN model has v layers. Each device has its own data set, and the data set can be divided into many batches. For the training process, each data batch will undergo one forward propagation and one backward propagation. We call this process as an iteration. For a whole training process, each batch may be trained for several iterations and there are many iterations. But since we use the synchronous iteration method for training, which means iterations on different devices do not affect each other. So in our model, we only

consider one batch in each device for one iteration. Define an iteration for one batch on e_i as a task m_i. All tasks are computed in a serial manner on end devices and edge servers. For one task, an appropriate model splitting point will be selected according to the calculation amount, the size of parameter data and the size of intermediate results of each layer. Then the first half of the model will be trained on the end device, and the second half of the model will be trained on an edge server. For each task, the edge server will be selected according to the real-time status to undertake the training of the second half of the model. The total training time under synchronous training is the sum of all iterative training time. So we can minimize the total training time by minimizing the training time of each iteration.

Optimization objective: Minimize the training time of one iteration.

2.2 Problem Formulation

The time for completing task m_i can be expressed as

$$t_i = t_i^e + t_i^{trans} + t_i^s + t_i^w + t_i^{up}, \tag{1}$$

where t_i^e and t_i^s represent the training time for m_i on end device and edge server, respectively, t_i^w represents the waiting time for executing m_i on the edge server, t_i^{trans} represents the data transfer time between end device and edge server, and t_i^{up} represents the time to update the parameters on PS.

Define a binary variable x_i^r to indicate whether m_i is split at the r-th layer of DNN, we have

$$x_i^r = \begin{cases} 1 & :m_i \text{ is split at the } r\text{-th layer of the DNN;} \\ 0 & :\text{otherwise.} \end{cases} \tag{2}$$

Note that $\sum_{r=1}^{v} x_i^r = 1$, which means m_i should select one and only one model splitting point.

For the first item t_i^e, it consists of the forward propagation time $t_i^{e,f}$ and the backward propagation time $t_i^{e,b}$ on e_i. We have

$$t_i^e = t_i^{e,f} + t_i^{e,b} = \sum_{r=1}^{v} (x_i^r \cdot \sum_{w=1}^{r} (L_{e_i,w}^f + L_{e_i,w}^b)), \tag{3}$$

where $L_{e_i,w}^f$ and $L_{e_i,w}^b$ represent that for task m_i, the forward and the backward propagation time for the w-th layer when processing one batch on e_i.

For the second item t_i^{trans}, similar to t_i^e, it consists of the forward transmission time $t_i^{trans,f}$ and the backward transmission time $t_i^{trans,b}$ on e_i. We have

$$t_i^{trans} = t_i^{trans,f} + t_i^{trans,b} = 2 \cdot \sum_{r=1}^{v} x_i^r \cdot \frac{G_r}{B_{e,s}}, \tag{4}$$

where G_r represents the output size of the r-th layer in the forward phase, which is equal to the output data of the $(r+1)$-th layer in the backward phase, and $B_{e,s}$

indicates the bandwidth between any device and any edge server (we suppose the bandwidths between any device and any edge server are the same).

For the third item t_i^s, denote a binary scheduling variable y_i^j to indicate whether m_i is executed on the server s_j. We have

$$y_i^j = \begin{cases} 1 & :m_i \text{ is executed on the server } s_j; \\ 0 & :\text{otherwise.} \end{cases} \tag{5}$$

Note that $\sum_{j=1}^{m} y_i^j = 1$, which means that a task can only select one edge server for offloading. Then the third item t_i^s can be expressed as

$$t_i^s = t_i^{s,f} + t_i^{s,b} = \sum_{j=1}^{m} (y_i^j \cdot \sum_{r=1}^{v} (x_i^r \cdot \sum_{w=r+1}^{v} (L_{s_j,w}^f + L_{s_j,w}^b))), \tag{6}$$

where $t_i^{s,f}$ and $t_i^{s,b}$ represent forward and backward propagation time on edge server s_j. $L_{s_j,w}^f$ and $L_{s_j,w}^b$ represent the m_i's forward and the backward processing one batch time for layer w on s_j.

For the fourth item t_w, we first denote AT_i as the arrival time of task m_i to the edge server. We have

$$AT_i = t_i^{e,f} + t_i^{trans,f}, \tag{7}$$

where $t_i^{e,f}$ is the forward propagation time on the end device e_i, and $t_i^{trans,f}$ is the time that the intermediate result is transferred from the end device to the edge server.

When task m_i arrives, there may already have some tasks from other devices on the server forming a waiting queue. Therefore, the task should wait for the completion of these previous tasks. Then the fourth item t_i^w can be expressed as

$$t_i^w = max\{0, I_i - AT_i\}, \tag{8}$$

where I_i represents the completion time of the task which precedes task m_i on the same task queue. I_i can be expressed as

$$I_i = \sum_{j=1}^{m} y_i^j \cdot (\sum_{i'=1}^{n} (y_{i'}^j \cdot Z_i(m_{i'}) \cdot (AT_{i'} + t_{i'}^w + t_{i'}^s))), \tag{9}$$

where $Z_i(m_{i'})$ is a binary scheduling variable to indicate whether task $m_{i'}$ precedes task $m_{i,\tau}$. We have

$$Z_i(m_{i'}) = \begin{cases} 1 & :m_{i'} \text{ arrives one bit earlier than task } m_i; \\ 0 & :\text{otherwise.} \end{cases}$$

For the last item t_i^{up}, we have

$$t_i^{up} = \sum_{r=1}^{v} x_i^r \cdot (\sum_{w=1}^{r} \frac{d_w}{B_{e,ps}}), \tag{10}$$

where d_w is the parameter data size of the w-th layer, and $B_{e,ps}$ is the bandwidth between PS and any device (we suppose the bandwidths between any device and PS are the same).

Our optimization goal is to minimize the training time of one iteration. So we have

$$\begin{aligned} &\min(\max_{i\in\{1,2,\ldots,n\}} t_i). \\ &(1)\ (2)\ (3)\ (4)\ (5)\ (6)\ (8)\ (10) \\ &\sum_{r=1}^{v} x_i^r = 1; \\ &\sum_{j=1}^{m} y_i^j = 1. \end{aligned} \tag{11}$$

The analysis of the problem shows that there are two types of 0–1 variable in this optimization problem: x_i^r and y_i^j, while the others are constants. As each end device has its own decision variables, the number of variables is large. The two decision variables of one device in (6) are multiplied together, so this optimization problem is nonlinear. In summary, we learn that the optimization problem is very complex and difficult to solve directly.

3 Algorithm

In the last section, we give the whole problem model and find it is hard to be solved directly. In this section, we will try to solve it and give our algorithm. For solving it, we need to find some way to reduce the size of the solution space. If we use the model splitting technique, the DNN can be split at any layer. However, the characteristics of different layers, such as the size of the computation and the parameter data, vary greatly. So not all layers are suitable for splitting. In our algorithm, we will first reduce the number of split points between each device and each server with the help of the Partition Points Selection(PPS) algorithm in [17]. Then based on matching theory, we design our whole algorithm of selecting a split point and an offloaded server for each end device. We name our algorithm as the Federated Learning Offloading Acceleration (FLOA) algorithm.

3.1 Many-to-One Matching with Externalities

In this sub-section, we will give some definitions. These definitions will be useful for the FLOA algorithm description. The related two-sided matching problem is to assign agents of one set to agents of other disjoint set [18]. From Sect. 2 we know that one end device can only select one edge server, while one edge server can be assigned to multiple devices. So the server selection is many-to-one matching, which can be defined as follows.

Definition 1. *Suppose e is one end device from E, and suppose s is one edge server from S. Define a many-to-one matching function μ on $E \cup S$, such that*

(1) $|\mu(e)| = 1$ for every device $e \in E$ and $|\mu(s)| \leq n$ for every server $s \in S$;
(2) $e \in \mu(s)$ if and only if $s = \mu(e)$.

Based on Definition 1, we define $U_{e_i}(\mu)$ as the utility function for e_i on μ, and define $U_s(\mu)$ as the utility function for s on μ. The utility function can be used for measuring the matching effect. Then we have

$$U_{e_i}(\mu) = \frac{1}{t_i(\mu)}, \tag{12}$$

where $t_i(\mu)$ is the task training time of e_i under the matching state μ.

For $U_s(\mu)$, we have

$$U_s(\mu) = \frac{1}{\max\limits_{i \in \{1,\ldots,n\}} t_i(\mu)}, \tag{13}$$

Based on (12) and (13), we know that the matching effect is decided by the task training time $t_i(\mu)$. However, in our environment, training time for different tasks have high relationship, which means devices and edge servers care about more than their own matching. So traditional pairwise stable matching may not exist [19]. We continue to leverage the concept of two-sided exchange stability [20] on the following definition.

Definition 2. *Define $\mu_e^{e'} = \{\mu \setminus \{(e,s),(e',s')\}\} \cup \{(e,s'),(e',s)\}$ as a swap matching, where $\mu(e) = s$, $\mu(e') = s'$, and $e \neq e'$.*

A swap matching can enable device e and e' to swap their matched servers with each other, and remain the matching of other devices and servers unchanged. Notice that when e' is 0, it means that the edge server matched by device e is changed to s', i.e. $\mu_e^0 = \{\mu \setminus \{(e,s)\}\} \cup \{(e,s')\}$, where $\mu(e) = s$ and $s \neq s'$.

Definition 3. *Given a matching function μ and a pair of devices (e,e'), if there exists $\mu(e) = s$ and $\mu(e') = s'$, and satisfies: (1) $\forall x \in \{e,e',s,s'\}$, $U_x(\mu_e^{e'}) \geq U_x(\mu)$; (2) $\exists x \in \{e,e',s,s'\}$, $U_x(\mu_e^{e'}) > U_x(\mu)$; then we call (e,e') as a swap-blocking pair under μ.*

Definition 4. *A matching μ is said to be two-sided exchange stable if and only if there is no swap-blocking pairs in μ.*

Definition 4 indicates that a matching is two-sided exchange stable if all devices fail to increase their own or the matching edge server's utility after changing the matching edge server.

3.2 FLOA Algorithm

Now we discuss the FLOA algorithm. The FLOA pseudo code can be found in Algorithm 1, and the detail steps are shown in the following.

Algorithm 1: Federated Learning Offloading Acceleration Algorithm

Input: E:Set of device; S:Set of server; $\mathcal{P}$:Set of split points to be selected.

Output: A two-sided exchange stable matching μ;a split point strategy θ.

1 Initialization a matching μ.
2 Initialize a split point strategy θ for all devices based on the set $\mathcal{P}$.
3 **repeat**
4 $\exists e \in E, \mu(e) = s$.
5 Re-select a split point in $\mathcal{P}$ for device e such that $U_e(\mu_e^0)$ is the maximum.
6 **if** *device e meets* $U_e(\mu_e^0) \geq U_e(\mu)$ **then**
7 Device e sends an offload request to server s'.
8 **if** $U_s(\mu_e^0) > U_s(\mu)$ *or* $U_{s'}(\mu_e^0) > U_{s'}(\mu)$ **then**
9 The edge server s' receives the request:$\mu \leftarrow \mu_e^0$
10 change split point strategy θ.
11 **else**
12 The edge server s' rejects the request,
13 the split point strategy θ remains unchanged.
14 $\exists e \in E, e' \in E, \mu(e) = s, \mu(e') = s'$, and $e \neq e'$.
15 Select a split point in $\mathcal{P}$ for device e and e' such that $U_e(\mu_e^{e'})$ and $U'_e(\mu_e^{e'})$ is the maximum.
16 **if** *device e and* e' *meet* $U_e(\mu_e^{e'}) \geq U_e(\mu)$ *and* $U_{e'}(\mu_e^{e'}) \geq U_{e'}(\mu)$ **then**
17 Device e sends an offload request to server s', e' sends an offload request to s.
18 **if** $U_s(\mu_e^{e'}) > U_s(\mu)$ *or* $U_{s'}(\mu_e^{e'}) > U_{s'}(\mu)$ **then**
19 The edge servers s and s' both receive the request:$\mu \leftarrow \mu_e^{e'}$
20 change split point strategy θ.
21 **else**
22 The edge servers s and s' both reject the request,
23 the split point strategy θ remains unchanged.
24 **until** *Matching* μ *meets Definition 4*;

step 1 Initialize a matching μ and a split point strategy θ based on the selected split point set $\mathcal{P}$, which is obtained from the PPS algorithm (line 1–2).
step 2 Perform μ_e^0 for device e and re-select a split point in $\mathcal{P}$ such that $U_e(\mu_e^0)$ is maximized under all available split points (line 4–5).
step 3 If $U_e(\mu_e^0)$ is greater than $U_e(\mu)$, which is the utility of device e when matching the original matching server s, then let e send an offload request to s' (line 6–7).

step 4 If $U_s(\mu_e^0)$ or $U_{s'}(\mu_e^0)$ increases after e changes the matched server, then let s' accept the offload request and update the matching μ and the split point policy θ, otherwise μ and θ remain unchanged. (line 6–13).

step 5 Perform $\mu_e^{e'}$ for device e and e', re-select a split point in $\mathcal{P}$ such that $U_e(\mu_e^{e'})$ and $U_{e'}(\mu_e^{e'})$ is maximized under all available split points (line 14–15).

step 6 If both e and e' have greater utility $U_e(\mu_e^{e'})$ and $U_{e'}(\mu_e^{e'})$ after swapping the matched servers than that before, then e and e' send offload requests to s' and s, respectively (line 16–17).

step 7 If the utility of s or s', $U_s(\mu_e^{e'})$ and $U_{s'}(\mu_e^{e'})$ increases after the swap, then s and s' accept the offload request and update μ and θ, otherwise μ and θ remain unchanged (line 18–23).

step 8 Repeat above steps until Definition 4 is satisfied.

4 Simulation and Experiment

In this section, we demonstrate the validity of the previous sections of the work through simulation experiments. The DNN model to be trained is VGG16, and the training data is a set of RGB images of size $224 \times 224 \times 3$. All model training tasks are performed using the pytorch. The computing power of the end devices is simulated with the following 5 CPUs: AMD Ryzen 7 4800H, AMD Ryzen 9 3900X, i5-6200U, i5-11400H, and i7 11700F. And the computing power of the edge servers is simulated with the following GPU: NVIDIA GeForce RTX 2060(Note book).

First, we use CPUs and GPU to train the DNN model, and get the forward and backward propagation time for each layer of the DNN model on each end device and each edge server, i.e. the value of $L^f_{e_i,w}$, $L^b_{e_i,w}$, $L^f_{s_j,w}$ and $L^b_{s_j,w}$. Then we get the size of parameter data and intermediate data for each layer of the DNN model, i.e. the value of G_r and d_w. We assume that there are several end devices, 5 edge servers and one parameter server in the scheduling network, and that the value of bandwidth $B_{e,ps}$ is $6MBps$.

In Subsect. 4.1, according to PPS algorithm, we compare the total training time of the task when different layers are used as split point, and get the set of split points $\mathcal{P}$. In Subsect. 4.2, simulations are carried out with different numbers of end devices and different bandwidths between end devices and edge servers, respectively, to validate the proposed system model and algorithm.

4.1 Split Point Selection Experiment

First, we train the DNN model using a set of images of size $224 \times 224 \times 3$, and obtain the forward and backward propagation time for each layer of VGG16 on different end devices and different edge servers. Figure 2 shows the training time for each layer of VGG16 on one of the devices and one of the servers. We then study the structure of VGG16 to obtain the amount of data for the parameters and intermediate data for each layer of VGG16.

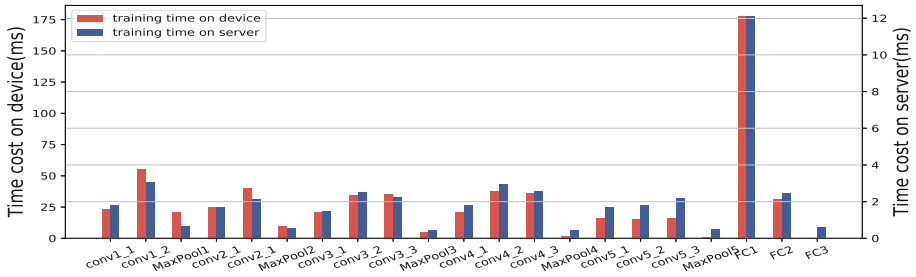


Fig. 2. The training time for each layer of VGG16 on device and server

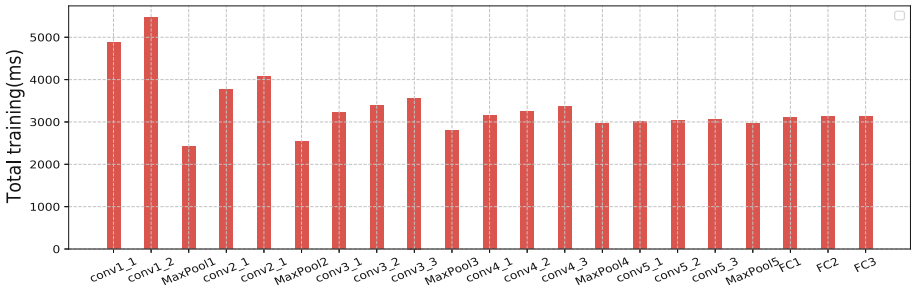


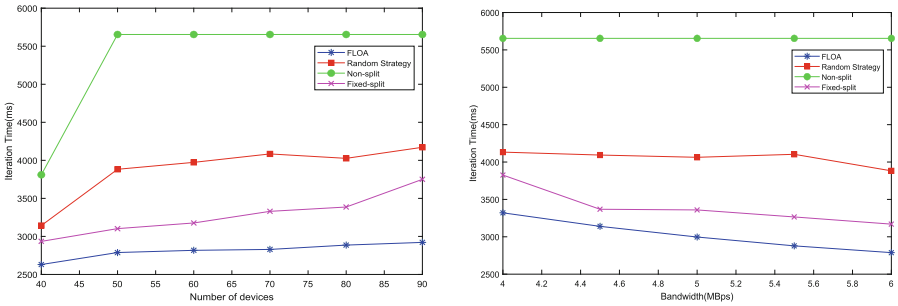
Fig. 3. The total training time under different split points of VGG16

We also set up a range of different bandwidths between the end devices and the edge servers. We then calculate the total training time ignoring waiting time for the task at different bandwidths with different layers as split point. Figure 3 shows that time when the bandwidth between the end device and the edge server is 6 Mbps. We obtain that the third, sixth and tenth layers are the three split points with the smallest total training time.

4.2 Simulation Results

In order to compare the effect of the algorithm with different number of devices and different bandwidths, first we set the bandwidth between the end device and the edge server to 6 Mbps, and the number of end devices from 40 to 90. Then we set bandwidth from 4 Mbps to 6 Mbps and fix the number of devices at 50. In both cases above, we calculate the time for a single iteration in different cases: Non-split, Fixed-split, Random strategy, and FLOA. The result is shown in Fig. 4.

In Fig. 4, FLOA indicates that we use the FLOA algorithm for training task scheduling. Non-split indicates that no model splitting is used, i.e. all training tasks are performed locally on the device. Fixed-split indicates that the split point is fixed and the training task randomly selects a edge server for offloading. Random strategy means randomly selecting a split point and a edge server for splitting and offloading.



(a) Different number of end devices (b) Different bandwidths

Fig. 4. Simulation result. (a) One iteration time with different number of devices. (b) One iteration time at different bandwidths.

As shown in the Fig. 4 (a), as can be seen from Non-split, the iteration time is always limited by the worst-performing device as the number of devices increases. FLOA obviously works better. Compared to Non-split, the one iteration time of FLOA is reduced by 47%. Compared to Fixed-split, the one iteration time is reduced by 14%. And compared to Random strategy, the one iteration time is reduced by 28%. In the Fig. 4 (b), FLOA is still obviously better than other solutions at different bandwidths.

5 Conclusion

In this paper, we use model splitting and task scheduling to reduce the overall training time for synchronous federated learning (FL) by reducing the time for one iteration. First we build mathematical models for synchronous federated learning in edge computing scenario. Then we analyse the mathematical model and design the corresponding solutions: using PPS algorithm to reduce the size of solution space and then proposing FLOA algorithm based on the two-sided matching theory to obtain the DNN splitting and offloading scheme. Experimental and simulation results show that using model splitting in synchronous FL, dynamically selecting split points and assigning training tasks can effectively reduce the time spent in synchronous FL.

References

1. Liang, Y., Cai, Z., Yu, J., Han, Q., Li, Y.: Deep learning based inference of private information using embedded sensors in smart devices. IEEE Network **32**(4), 8–14 (2018)
2. Cai, Z., Xiong, Z., Xu, H., Wang, P., Pan, Y.: Generative adversarial networks: a survey toward private and secure applications. ACM Comput. Surv. **54**(6), 1–38 (2021)
3. Ren, J., Yu, G., Ding, G.: Accelerating DNN training in wireless federated edge learning systems. IEEE J. Sel. Areas Commun. **39**(1), 219–232 (2021)

4. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
5. Cai, Z., Shi, T.: Distributed query processing in the edge-assisted IoT data monitoring system. IEEE Internet Things J. **8**(16), 12679–12693 (2021)
6. Zhu, T., Shi, T., Li, J., Cai, Z., Zhou, X.: Task scheduling in deadline-aware mobile edge computing systems. IEEE Internet Things J. **6**(3), 4854–4866 (2019)
7. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Singh, A., Zhu, J. (eds.) Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 54, pp. 1273–1282. PMLR, 20–22 April 2017
8. Pang, J., Huang, Y., Xie, Z., Han, Q., Cai, Z.: Realizing the heterogeneity: a self-organized federated learning framework for IoT. IEEE Internet Things J. **8**(5), 3088–3098 (2021)
9. Xiong, Z., Cai, Z., Takabi, D., Li, W.: Privacy threat and defense for federated learning with non-I.I.D. data in AIoT. IEEE Trans. Ind. Inform. **18**(2), 1310–1321 (2022)
10. Lim, W.Y.B., et al.: Federated learning in mobile edge networks: a comprehensive survey. IEEE Commun. Surv. Tutorials **22**(3), 2031–2063 (2020)
11. Wu, W., He, L., Lin, W., Mao, R.: Accelerating federated learning over reliability-agnostic clients in mobile edge computing systems. IEEE Trans. Parallel Distrib. Syst. **32**(7), 1539–1551 (2021)
12. Zheng, J., Li, K., Tovar, E., Guizani, M.: Federated learning for energy-balanced client selection in mobile edge computing. In: 2021 International Wireless Communications and Mobile Computing (IWCMC), pp. 1942–1947 (2021)
13. Luo, S., Chen, X., Wu, Q., Zhou, Z., Yu, S.: HFEL: joint edge association and resource allocation for cost-efficient hierarchical federated edge learning. IEEE Trans. Wireless Commun. **19**(10), 6535–6548 (2020)
14. Wang, X., Han, Y., Leung, V.C.M., Niyato, D., Yan, X., Chen, X.: Convergence of edge computing and deep learning: a comprehensive survey. IEEE Commun. Surv. Tutorials **22**(2), 869–904 (2020)
15. Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K., Zhang, J.: Edge intelligence: paving the last mile of artificial intelligence with edge computing. Proc. IEEE **107**(8), 1738–1762 (2019)
16. Qu, X., Hu, Q., Wang, S.: Privacy-preserving model training architecture for intelligent edge computing. Comput. Commun. **162**, 94–101 (2020)
17. Shi, L., Xu, Z., Shi, Y., Fan, Y., Ding, X., Sun, Y.: A DNN inference acceleration algorithm in heterogeneous edge computing: joint task allocation and model partition. In: Gao, H., Wang, X., Iqbal, M., Yin, Y., Yin, J., Gu, N. (eds.) CollaborateCom 2020. LNICST, vol. 349, pp. 237–254. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67537-0_15
18. Liu, Z., Wang, K., Zhou, M.T., Shao, Z., Yang, Y.: Distributed task scheduling in heterogeneous fog networks: a matching with externalities method. In: 2020 International Conference on Computing, Networking and Communications (ICNC), pp. 620–625 (2020)
19. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Mon. **120**(5), 386–391 (2013)
20. Bodine-Baron, E., Lee, C., Chong, A., Hassibi, B., Wierman, A.: Peer effects and stability in matching markets. In: Persiano, G. (ed.) SAGT 2011. LNCS, vol. 6982, pp. 117–129. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24829-0_12