IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 8, 2020, accepted February 25, 2020, date of publication March 3, 2020, date of current version March 19, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2978082

Privacy-Preserving Asynchronous Federated Learning Mechanism for Edge Network Computing

XIAOFENG LU[1], YUYING LIAO[1], PIETRO LIO[2], AND PAN HUI[3], (Fellow, IEEE)
[1]School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China
[2]Computer Laboratory, University of Cambridge, Cambridge CB2 1TN, U.K.
[3]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong

Corresponding author: Xiaofeng Lu (luxf@bupt.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61472046, in part by the Ant Financial through the Ant Financial Science Funds for Security Research, and in part by the Beijing Association for Science and Technology Seed Fund.

ABSTRACT In the traditional cloud architecture, data needs to be uploaded to the cloud for processing, bringing delays in transmission and response. Edge network emerges as the times require. Data processing on the edge nodes can reduce the delay of data transmission and improve the response speed. In recent years, the need for artificial intelligence of edge network has been proposed. However, the data of a single, individual edge node is limited and does not satisfy the conditions of machine learning. Therefore, performing edge network machine learning under the premise of data confidentiality became a research hotspot. This paper proposes a Privacy-Preserving Asynchronous Federated Learning Mechanism for Edge Network Computing (PAFLM), which can allow multiple edge nodes to achieve more efficient federated learning without sharing their private data. Compared with the traditional distributed learning, the proposed method compresses the communications between nodes and parameter server during the training process without affecting the accuracy. Moreover, it allows the node to join or quit in any process of learning, which can be suitable to the scene with highly mobile edge devices.

INDEX TERMS Federated learning, edge computing, privacy preservation, asynchronous distributed network, gradient compression.

I. INTRODUCTION

Since its introduction, deep learning has gradually changed the way we live, learn, and work. It has made great breakthroughs in speech, image, as well as text recognition [1], language translation, and other area. Traditional deep learning requires a large amount of data to be collected for learning. Although the ascendancy of deep learning is undeniable, the involved training data may have serious privacy issues. First, millions of photos and videos are collected centrally, and this data is kept by large companies forever. Users can neither delete the data nor control how it is used. Second, images and videos are likely to contain a lot of sensitive information [2], such as faces, license plates, computer screens, and people's conversations. In addition, the internet giants monopolize this "big data" and enjoy huge economic benefits behind it.

The associate editor coordinating the review of this manuscript and approving it for publication was Laurence T. Yang.

It is known that with an increase in the number of training data sets and the data diversification, the neural network models behave better. Data held by one single organization (such as a specific medical clinic) may be similar and not diversified enough, so that models based on such data sets may eventually result in over-fitting or less scalability. In this case, the limitations of privacy and confidentiality significantly affect performance of the neural network model.

On the other hand, the current global Internet of Things has entered the third era of development. In 2018, the number of global IoT connections was about eight billion [3]. These IoT devices generate a large amount of data every day. If the data generated by the Internet of Things is transmitted to the cloud-computing center for processing, it will cause transmission congestion and data processing delays. It is necessary to shift computing tasks from cloud center to the network edge. Based on this, fog computing and edge computing have been proposed to compensate for some of the shortcomings of cloud computing.

Fog computing [4], [5] is an architecture that distributes computation, communication, control, and storage closer to the user side. Fog servers can be connected to the cloud so as to leverage the rich functions and application tools of the cloud, which is conducive to data protection [6]. Therefore, ''the fog is a cloud close to the ground''. Fog computing is not meant to substitute cloud computing, but to complement it in order to ease bandwidth burden and reduce transmission latency [7].

Edge computing refers to a close-to-data-source platform that integrates network, computing, storage, and other core capabilities, providing near-end services [8]. Compared with the data-centralized cloud computing method, edge computing processes data at the edge of the network. Because of its near-end characteristics, edge computing can reduce network bandwidth load, shorten response time, and improve battery life while ensuring data security and privacy.

Fog computing and edge computing both process data close to the data collectors, moving computing load from servers to decentralized computing nodes to achieve workload balance [9], [10]. In many ways, they are very similar, but there are still slight differences between them. Fog computing introduces a ''fog layer'' between the cloud and the clients. The fog layer is composed of many fog servers, which are located close to the edge of the network. Each fog server is a highly virtualized computing system, similar to a lightweight cloud server. On the other hand, edge computing executes computing tasks at the edge of the network. Computing nodes are usually IoT devices with certain computing and storage capabilities. Except for the difference between fog computing and edge computing, they both are complementary to cloud computing. The effective combination of the three can bring forward solutions for various complex needs in reality.

With the advent of 5G, edge computing and federated learning have attracted widespread attention. McMahan and Ramage *et al.* [11] gave a general description of Federated Learning. Bonawitz *et al.* [12] continued their research and explored more possibilities. Federated learning means ''joint learning'', where multiple devices work together to train learning models collaboratively. Traditional federated learning is a decentralized learning framework, in which most of the computation (like model training) is performed on the node side. Nodes learn locally on each device and gradually optimize the learning models through interaction with the central server. Fig. 1 shows the simplified federated learning framework. Throughout the federated learning process, privacy data does not leave the data owner, and it does not need to be shared with other nodes, solving data security and other issues.

Because of the decentralized nature of federated learning, it can make better use of the computing power of IoT devices. Fig. 2 presents a sample application scenario, in which data is stored locally on the cameras so that the cameras themselves can learn from the data. After learning, the cameras have the ability to make simple decisions, such as adjusting real-time traffic by controlling signal lights. Compared to uploading the data to a central server for processing, this method can respond faster and is suitable for real-time scenarios.

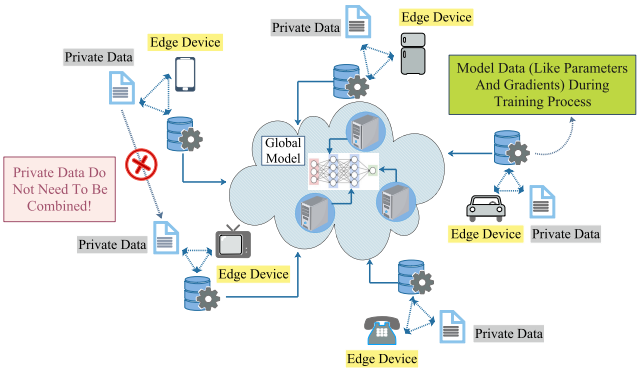


FIGURE 1. Federated learning system architecture.

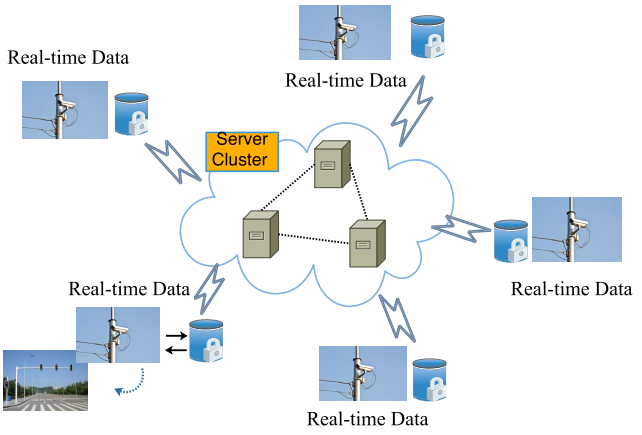


FIGURE 2. One of the possible application scenario of federated learning.

In this paper, we propose a Privacy-Preserving Asynchronous Federated Learning Mechanism for Edge Network Computing (PAFLM) that can satisfy the reality of learning from multi-party data without sharing their own private information. Compared with traditional distributed learning, PAFLM ensures learners' freedom and preservation of privacy without affecting accuracy. Each node independently trains on its own using a local dataset. Our method helps to improve the accuracy of the participants' local models because the limited data owned by either party can easily result in a trapping into the local optimum. Using models learned by other participants to optimize parameters of local model can effectively help each participant escape local preferences and enable them to explore other values, resulting in more accurate models.

Our contributions are as follows:

(1) We designed a federated learning system that is more suitable for collaborative learning of discrete nodes in edge networks and is different from the existing distributed learning system, so that nodes can learn from the data without sharing private information.

(2) We improve and design a gradient compression algorithm based on previous work. When the gradient communications are compressed to 8.77% of the original communication times, the accuracy of the test set is only reduced by 0.03%. The gradient data indirectly reflects the information of training samples, and the attacker can deduct the sample data from the massive effective gradient. Therefore, reducing gradient communications can effectively lower the likelihood of privacy breaches.

(3) We further explored the possibility of asynchronous federated learning in order to better adapt to the unrestricted characteristics of edge nodes, and as a result, designed, proposed, and tested a dual-weights correction method to solve the performance degradation caused by asynchronous learning.

The rest of this paper is organized as follows. Section I is the brief introduction of our work. The state-of-art solutions of federated learning are introduced in section II. In section III, we demonstrate the basic structure of PAFLM, which consists of two parts: self-adaptive threshold gradient compression and asynchronous federated learning with dual-weights correction. In section IV, we further illustrate the self-adaptive threshold gradient compression method, and discuss its examination-free mechanism. In section V, we propose a dual-weights correction to solve asynchronous federated learning problems. The experiment of our proposed methods is shown in section VI, and section VII presents the conclusion.

II. RELATED WORK

A. FEDERATED LEARNING

Traditional federated learning is a decentralized learning method. It allows learners to store the training data privately on their mobile devices and learn a shared model by aggregating locally-computed updates. In the federated learning framework, sensitive sample data does not need to transfer to a cloud center. Edge nodes execute computing tasks separately and independently, guaranteeing data integrity. With the development of technology, federated learning has gradually evolved into different forms. A combination of federated learning and blockchain is an interesting exploration. Blockchain has emerged as a chronological, decentralized, provenance-preserving, and immutable ledger technology. It is an effective solution to replace a vulnerable central server in an unsecure environment. Blockchain integrated with federated learning can effectively solve the security issues involving the central server [13]–[17].

However, we are more concerned with the traditional federated learning framework rather than the development of its other forms. Most of the studies about federated learning are focused on synchronous training algorithms, such as the federated average algorithm proposed by McMahan *et al.* [11]. Privacy protection in federated learning, such as security aggregation [18], requires synchronization operations at the device level, so it essentially belongs to the category of synchronous training. In addition, researchers began to implement the federated learning system on vehicle-to-vehicle communications [19], medical applications [20], and in other areas.

B. GRADIENT COMMUNICATION WORKLOAD

Because federated learning requires a large number of learning nodes, the huge network communication bandwidth required by these nodes cannot be ignored. Researchers have proposed many ways to overcome communication bottlenecks, where gradient quantization and sparseization are two popular areas of research.

Gradient quantization converts gradients to a low-precision value to reduce the communication bandwidth. 1-bit Stochastic Gradient Descent (SGD) [21] reduces the size of gradient transmission data and achieves 10x acceleration in traditional speech applications. TernGrad uses ternary gradients to accelerate distributed deep learning in data parallelism [22]. Both of these tasks demonstrate the convergence of quantitative training. However, TernGrad has only been tested on CNN, and QSGD has only calculated the accuracy of RNN loss.

Gradient sparseness compresses gradient communication times to mitigate network pressure. The first thing that comes to mind is regularly skipping some of the gradient interactions [23]. Dryden *et al.* [24] selected a fixed ratio of positive and negative gradients. Chen *et al.* [25] proposed and proved mathematically a method that automatically adjusts the compression ratio according to the local gradient activity. This method achieves 200 times compression for the fully-connected layer, 40 times compression of the convolution layer, and the reduced top-1 accuracy that is negligible in the ImageNet data set. However, ignoring the information of real-time gradients and compressing communication only according to the fixed interval or ratio can adversely affect the training process. At the same time, Strom *et al.* [26] used thresholds to achieve gradient sparseness—only sending gradients that are greater than a predefined constant threshold. Gradient Dropping [27] uses a single absolute threshold to sparse the gradient matrix. Gradient Dropping saves 99% of gradient swaps while causing a 0.3% loss of BLEU in machine translation tasks.

However, the performance of the above work greatly relies on choosing the correct threshold, which is not an easy task. A ''perfectly'' fixed threshold can solve the problem of one specific scenario, but for another problem scenario it may be a disaster. Moreover, the threshold selection requires a lot of time to debug, which is unrealistic for scenarios requiring high-speed response. At this time, the Lazily Aggregated Gradient (LAG) [28] appears to be more fascinating. LAG adaptively calculates gradient and skips partial gradient communications to reduce communication bandwidth and mitigate server pressure. The basic principle is to detect slowly-changing gradient and compress it. LAG is very valuable, but it requires the optimization problem to be convex and Lipschitz smooth.

Gradient compression in PAFLM belongs to the category of gradient communication sparseization. But unlike the scheme mentioned above, there is no need to explore an optimal gradient threshold in PAFLM because the optimal thresholds vary from problem to problem. In addition, defining gradient thresholds reduces the scalability of the asynchronous federated learning framework.

If the learning object is spread to edge nodes, synchronous training becomes difficult to implement and does not meet the needs of reality. Although a few researchers have adopted the asynchronous learning method, they all use it as a regularization method [29], lacking detailed systematic research. And in most studies, during the federated learning, training status of participating nodes is almost the same. There is a lack of attention to the fact that the learning progress is quite different, even when the learning process is completely staggered. In this paper, we study this type of asynchronous learning problem and mitigate the loss of precision caused by asynchronization.

III. PRIVACY-PRESERVING ASYNCHRONOUS FEDERATED LEARNING MECHANISM FOR EDGE NETWORK COMPUTING

Federated Learning is an emerging artificial intelligence technology. It is designed to carry out efficient machine learning among multiple participants or computing nodes while ensuring the security of private personal data. The machine learning algorithms used in federated learning are not limited to neural network algorithms, but also include other algorithms, such as random forests.

The traditional federated learning mechanism consists of a parameter server and multiple edge nodes. The parameter server is responsible for collecting gradients uploaded by each participating node, updating the parameters of the model according to the optimization algorithm and maintaining the global parameters. The participating nodes learn from their sensitive data independently and locally. After each epoch, nodes upload gradients to the parameter server, and the server summarizes and updates the global parameters. Then nodes download the updated parameters from the parameter server, overwriting their local model parameters and proceeding to the next iteration. During the whole learning process, nodes only communicate with the parameter server. Learning nodes cannot obtain any information about the remaining nodes, except the global parameters that are jointly maintained, which guarantees the confidentiality of the private data.

Fig. 3 shows the Privacy-Preserving Asynchronous Federated Learning Mechanism for Edge Network Computing, which is based on the traditional federated learning framework. Our work contains two layers: parameter server layer and edge node layer. The self-adaptive threshold gradient compression module is at the edge node layer, and the asynchronous federated learning module spans both layers.

Self-Adaptive Threshold Gradient Compression Federated learning oriented for edge network requires interaction with multiple edge nodes for real time training data,

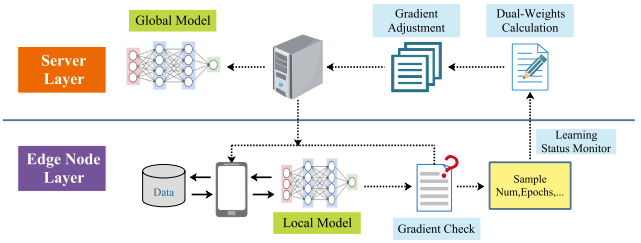


FIGURE 3. Privacy-preserving asynchronous federated learning mechanism for edge network computing.

resulting in high communication costs, which greatly limit the scalability of the federated learning system. In this case, communication latency is the bottleneck of the performance of the whole learning framework. It has been found that 99.9% of the gradient exchanges in distributed SGD are redundant. In this paper, we present a gradient sparseization method that compresses the interaction between nodes and parameter server, effectively reducing communication bandwidth in the learning process. In the experiment, when the gradient communications are compressed to 8.77%, the accuracy of the test set is only reduced by 0.03%.

Asynchronous Federated Learning with Dual-Weights Correction Because edge nodes are highly unrestrained, it is impractical to force all nodes to train at the same time. We explore the different situations of asynchronous learning and propose the dual-weights correction for asynchronous learning with edge nodes. Asynchronous federated learning aims to provide a freer learning environment for edge nodes and to reduce the loss of precision caused by extremely unrestrained learning.

Self-adaptive threshold gradient compression module is divided into two sub-modules: self-adaptive threshold computation and gradient communication compression, which are respectively responsible for calculating the threshold according to the latest parameter change and using the threshold to compress the redundant gradient communications. Asynchronous federated learning is divided into four sub-modules: parameter update, gradient adjustment, dual-weights calculation, and learning status monitor. The monitor sub-module is responsible for monitoring the learning state of the node, such as current learning round and the number of samples. The dual-weights computation sub-module calculates the corresponding sample weights and parameter weights according to its learning information. The gradient adjustment sub-module corrects the gradient uploaded by the node according to the dual-weights. Moreover, the corrected gradient is used for the global parameter update in the parameter update sub-module.

The self-adaptive threshold gradient compression module and the asynchronous federated learning module are not independent of each other. The self-adaptive threshold calculation needs to obtain the updated parameters and compare them with the historical parameters. The compressed gradient communications affect the learning status of the node, which in turn affects the dual-weights calculation.

IV. SELF-ADAPTIVE THRESHOLD GRADIENT COMPRESSION

Gradient compression is used to compress the gradient communications between nodes and parameter server. In other words, it compresses the number of calls that one single node communicates to the parameter server. As described above, the information contained in the gradient is largely redundant. When the number of edge nodes increases, the communication cost required for these redundant communications becomes enormous. Moreover, the parameter server is under a large communication pressure, and skipping part of the information interaction can reduce the communication load. In addition, the gradient data indirectly reflects the sample information secretly owned by nodes. Attackers can deduce sample data of target nodes by combining the trained global model with the collected effective gradient information. Gradient compression reduces gradient communications, which not only improves the efficiency of federated learning, lightens network load, but also reduces the chance of sample privacy leakage.

Previous work [23]–[27], whether simply introducing the communication compression ratio or compressing gradient communications based on the fixed threshold, contains many shortcomings. Because gradient change is varied for different learning processes, simply selecting nodes for compression according to the compression ratio tends to ignore the gradient with a large amount of information, which affects the global model training. At the same time, using fixed thresholds can easily result in over-compression, causing model convergence difficulties in the latter part of the training.

In PAFLM, the nodes automatically adapt to the changes of each round of the model training process, and calculate the appropriate threshold to compress the gradient communications. Only the qualified nodes can communicate with the parameter server in the corresponding round. Otherwise, the gradient is accumulated locally until the next round. The final gradient will accumulate enough information to upload to the parameter server. Regardless of whether the node is qualified for communication in a certain round, gradient checks are performed at the end of each learning iteration; that is, the gradient checks participate in the entire learning process. We propose the self-test expression in gradient compression and prove it with a mathematical method.

A. MATHEMATICAL DERIVATION

Table 1 explains the symbolic representations involved in the formulas below.

In the gradient descent algorithm (GD), there is a parameter server that needs to communicate with M learning nodes to complete the update of the model parameters. In the k_{th} iteration, the parameter server broadcasts the current model $\boldsymbol{\theta}^{k-1}$ to all learners; each learning node $m \in M$ computes $\nabla_{m-1}(\boldsymbol{\theta}^k)$ and uploads it to the parameter server; then parameter server receives the gradients from all learning nodes. Parameter server updates the model parameters by iterating

TABLE 1. The meaning of symbol in the following formulas.

Symbol	Meaning
$\boldsymbol{\theta}^k$	the k_{th} round parameter of the parameter server
$\nabla_m(\boldsymbol{\theta}^k)$	the k_{th} round gradient calculated by node m based on the k_{th} round parameter
∇_M^k	the sum of the k_{th} round gradient of all nodes in the set M
M	the set contains all edge nodes
m	$card(M)$, the total number of all elements in the set M
S	total number of samples owned by all nodes
S_i	number of samples owned by node i
β_S^i	the sample weight of current epoch of node i
β_P^i	the parameter weight of current epoch of node i

the gradient descent algorithm.

$$\boldsymbol{\theta}^k = \boldsymbol{\theta}^{k-1} - \alpha \nabla_M^{k-1}. \tag{1}$$

where α is the learning rate, and ∇_M^{k-1} is the aggregation gradient, which represents a round of variation of the model.

We refers to the work of Chen *et al.* [28] about compressing the gradient interaction to reduce the bandwidth burden of communications. The main idea of the algorithm is to, in a certain round, ignore the "Lazy" nodes and only communicate with the "Hard work" nodes. These ignored nodes are represented as M_L, and the nodes communicating with the server are M_H, that is, $M = M_L + M_H$. Nodes that are ignored still need to accumulate the gradient locally, and then finally, gradient will increase large enough to participate in the communications. Therefore ∇_M^{k-1} can be updated to:

$$\nabla_M^{k-1} = \nabla_{M_L}^{k-1} + \nabla_{M_H}^{k-1}. \tag{2}$$

Let the set M_L satisfy the following formula, where m_L, m respectively are the total number of all elements in the set M_L and M:

$$\frac{\left\|\nabla_{M_L}^{k-1}\right\|^2}{m_L} \leq \frac{\left\|\nabla_M^{k-1}\right\|^2}{m}. \tag{3}$$

Substituting formula (1) into the above formula we can obtain:

$$\left\|\nabla_{M_L}^{k-1}\right\|^2 \leq \frac{m_L}{\alpha^2 m}\left\|\boldsymbol{\theta}^k - \boldsymbol{\theta}^{k-1}\right\|^2. \tag{4}$$

where $\left\|\nabla_{M_L}^{k-1}\right\|^2 = \left\|\sum_{m\in M_L} \nabla_m(\boldsymbol{\theta}^k)\right\|^2$. According to the *Inequality of Arithmetic and Geometric Means*, $\left\|\nabla_{M_L}^{k-1}\right\|^2$ satisfies the following formula:

$$\left\|\nabla_{M_L}^{k-1}\right\|^2 \leq m_L \sum_{m\in M_L}\left\|\nabla_m(\boldsymbol{\theta}^k)\right\|^2. \tag{5}$$

The formula (3) must satisfy if the node $m \in M_L$ meets the following conditions :

$$\left\|\nabla_m(\boldsymbol{\theta}^k)\right\|^2 \leq \frac{1}{\alpha^2 m_L m}\left\|\boldsymbol{\theta}^k - \boldsymbol{\theta}^{k-1}\right\|^2. \tag{6}$$

Since the total number of set M_L cannot be obtained in advance, in order to simplify the problem, we introduce a

proportional coefficient β to express the total number of nodes of the set M_L, that is, $m_L = \beta m$. From formula (6) we can obtain:

$$\left\|\nabla_m(\boldsymbol{\theta}^k)\right\|^2 \leq \frac{1}{\alpha^2\beta m^2}\left\|\boldsymbol{\theta}^k - \boldsymbol{\theta}^{k-1}\right\|^2. \tag{7}$$

However, the acquisition of $\boldsymbol{\theta}^k - \boldsymbol{\theta}^{k-1}$ is difficult, but since the parameter changes tend to be smooth during the learning process, $\boldsymbol{\theta}^k - \boldsymbol{\theta}^{k-1}$ is approximated as:

$$\boldsymbol{\theta}^k - \boldsymbol{\theta}^{k-1} \approx \sum_{d=1}^{D} \xi_d(\boldsymbol{\theta}^{k-d} - \boldsymbol{\theta}^{k-1-d}). \tag{8}$$

where ξ_d and D are constant coefficients, simply we choose $\xi_d = 1/D$.

Substituting equation (8) into equation (7) gives us:

$$\left\|\nabla_m(\boldsymbol{\theta}^k)\right\|^2 \leq \frac{1}{\alpha^2\beta m^2}\left\|\sum_{d=1}^{D} \xi_d(\boldsymbol{\theta}^{k-d} - \boldsymbol{\theta}^{k-1-d})\right\|^2. \tag{9}$$

Formula (9) is the self-testing expression for the gradient check of federated learning, that is, the node conducts the self-test operation after the end of every round of learning. It should be noted that if the gradient does **not satisfy** formula (9), the corresponding node will communicate with the parameter server. Otherwise, the current round communication will be skipped, and learning nodes cumulative gradient locally, continuing to execute the next round of learning.

B. GRADIENT COMPRESSION WITH EXAMINATION-FREE MECHANISM

As described above, gradient compression compresses redundant gradient communications in federated learning. The edge learning nodes calculate the current gradient after one epoch of training and then decide whether to upload parameters to server according to the self-test expression. Nodes that satisfy the condition communicate with the server, upload the gradient, and receive the updated parameter. Otherwise, nodes accumulate the gradient locally and begin the next epoch. This is a typical example of sacrificing the local computation time of nodes to reduce global communication time. In many neural network learnings, the number of epochs is large. Therefore, the calculation time brought by the self-testing is undoubtedly not negligible.

In order to alleviate the amount of local computation, we add an "examination-free" mechanism based on the gradient compression described above. Let the set be $\gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_{TotalEpochs}\}$, where the random variable $\gamma_k \in \gamma$ indicates the possibility of nodes that can skip the gradient check and directly communicate with the parameter server after the k_{th} epoch. Only when $\gamma_k \geq \gamma_T$, the node is eligible to skip the self-testing process, where γ_T is a predefined probability threshold.

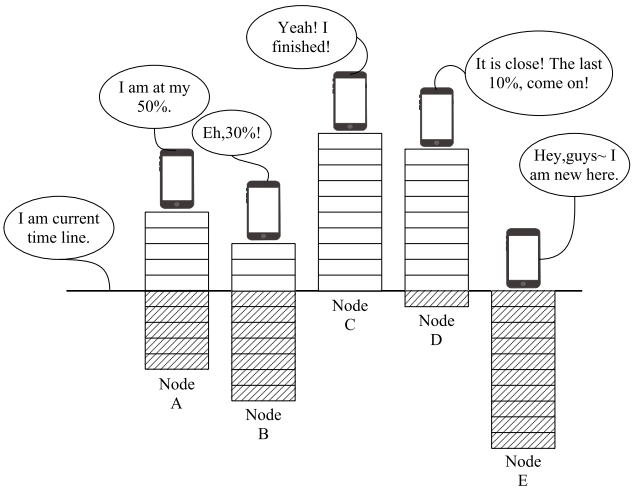


FIGURE 4. Asynchronous federated learning (bright part means the finished learning tasks, while the dark part means tasks that need to be carried on).

V. ASYNCHRONOUS FEDERATED LEARNING FOR EDGE NETWORK

As mentioned above, PAFLM is oriented toward the unrestricted edge nodes. There are many factors that can cause asynchronous problems, such as different time to join federated learning and different computing power (varied computing time required for the same training task), gradient compression, learning interruption caused by various external factors, and so on. We propose dual-weights correction to solve asynchronous federated learning problems.

A. PROBLEMS IN ASYNCHRONOUS FEDERATED LEARNING

In asynchronous federated learning, nodes have uneven learning samples and different learning status. As shown in Fig. 4, the straight line in the figure represents the current time. The total number of learning rounds of all nodes is the same. The part that intersects with timeline indicates the current epoch. Each node is at different stages of learning, such as node A is at the 50% of the learning process. If the total number of epochs is 1000, node A is at its 501_{th} round. It obviously is unreasonable for the nodes with large differences to update the global parameters equally. Therefore, we introduce dual-weights correction to solve the problem of unbalanced learning status in asynchronous learning.

B. ASYNCHRONOUS FEDERATED LEARNING WITH DUAL-WEIGHTS CORRECTION

The dual-weights of PAFLM are divided into two parts: sample weights and parameter weights. The sample weight is determined by the proportion of the node samples to the total samples of all learning nodes, and the parameter weight is affected by the time difference between one node downloading parameters and it uploading corresponding gradients.

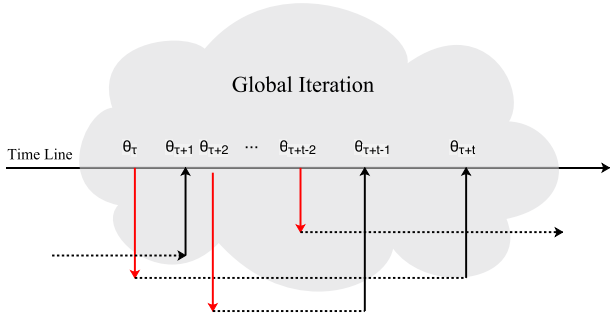


FIGURE 5. Asynchronous federated learning and parameters staleness.

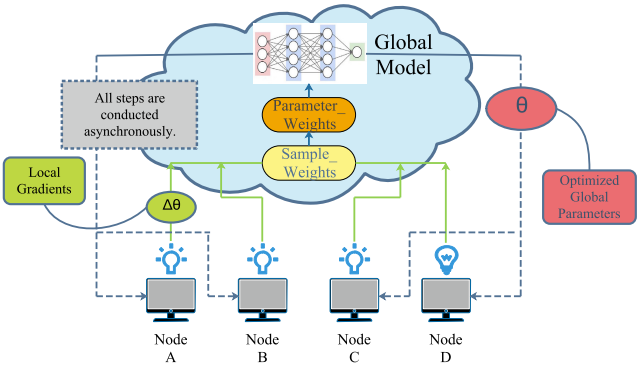


FIGURE 6. Detail of asynchronous federated learning in PAFLM.

Definition 1: The sample weight represents the proportion of the sample owned by one node to the total number of all learning nodes.

Let n learning nodes be $N = \{N_1, N_2, \ldots, N_n\}$, and the number of samples owned by node N_i be represented by S_i. The sample weight of one particular node can be computed from its sample number and the total sample number. The calculation is as follow:

$$\beta_S^i = \frac{S_i}{S}. \tag{10}$$

where β_S^i represents the sample weight of node i,and $S = \sum_{j=0}^{n} S_j$ is the total sample number of n nodes.

Definition 2: The parameter weight indicates the time difference between one node downloading parameters and uploading corresponding gradients.

From the perspective of the parameter server, the entire federated learning is the iteration of the parameter optimization update. Nodes continually send requests to the parameter server for downloading the latest parameters, and upload the latest gradient to update the parameters. The simplified process is shown in Fig. 5. Each node has upload operations of other nodes interspersed between the time when its latest parameter is downloaded and the corresponding gradient is uploaded. In the parameter optimization described in Fig. 5, the gradients have certain ''staleness'', and the definition of ''staleness'' is as follows:

$$\mu_{staleness} = I_{upload} - I_{download}. \tag{11}$$

As the example in Fig. 5 shows, the staleness is $\mu_{staleness} = t + \tau - t = \tau$. The parameter staleness reflects the computing power of the node in some ways. In order to guarantee that nodes with larger staleness have smaller parameter weights and that their weight attenuation process is relatively flat, we select the exponential function with the base number less than 1 as the attenuation function of the parameter weights:

$$\beta_P^i = \alpha^{(\frac{1}{n}\mu_{staleness}^i)-1}. \tag{12}$$

where $\mu_{staleness}^i$ represents the staleness of node i, and α is the base of the exponential function that determines the speed of attenuation. When the exponential part is greater than zero, we want the parameter weight to decrease as the exponential value increases, so the selection interval of α is reduced to [0, 1]. As we all known, in the interval [0, 1], the larger the α, the gentler is the decline of the calculation result. So, in this paper, we simply choose $\alpha = 0.9$.

The dual-weights correction formula is as follows, where $\boldsymbol{\theta}$ is the original model parameter and $\boldsymbol{\theta}'$ is the modified model parameter:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} * \beta_S^i * \beta_P^i. \tag{13}$$

As shown in Fig. 6, in the asynchronous federated learning, the gradient submitted by the edge nodes is subject to dual-weights correction on the parameter server before optimizing the global model. The corrected gradients update the global parameters according to a specific optimization algorithm. After one round of optimization is finished, nodes download the latest parameters and overwrite their local parameters to prepare the next learning iteration.

VI. EXPERIMENT

A. EXPERIMENTAL CONFIGURATION

In order to simulate the actual scene, we set up the following experimental environment: a GPU server with a strong calculation capacity played the role of a parameter server, being responsible for most of the calculation work. Many other computers simulated the individual learning nodes in the edge network, and each of them independently conducted federated learning. The communications between the parameter server and the nodes were based on the Thrift framework. In it, each computer stores its data locally (in our experiment, the data of each node took 0.2% of the total data), and trains the neural network model based on their own private data. Our research objects were multiple edge nodes with different computing powers, resulting in different data processing times. Therefore, we tried to simulate these devices by adding pause intervals between every adjacent epoch. In order to better simulate the real-world scenarios, the federated learning systems built on these computers were controlled by an independent management computer.

The experimental environment is shown in Fig. 7, where **Order** represents the start order of N nodes, and **Interval** represents the interval time between two epochs. In the initialization phase, the management node generated a shuffled

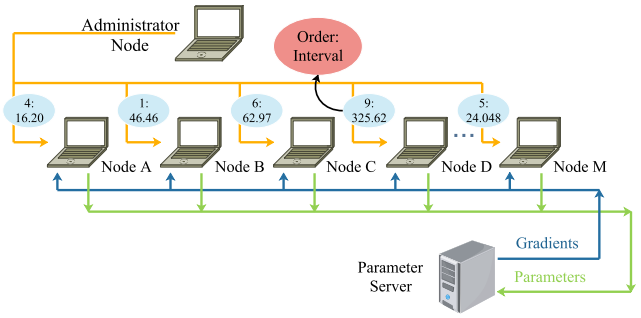


FIGURE 7. Experimental configuration example.

sequence **Order** according to the total number of nodes (such as 4,1,6, 9,...,5 in Fig. 7), and correspondingly generated a random sequence **Interval** (as 16.20, 46.46, 62.96, 325.62, ..., 24.048(s) in the Fig. 7). After the initialization, the management node started the designated node according to the **Order: Interval** sequence pair.

B. EXPERIMENTAL INDEX

The experimental indexes used for evaluation were Accuracy (Acc), Compression Ratio (CR), and Compression Balance Index (CBI).

Acc reflects the performance of the model classification, which is defined as follows:

$$Acc = \frac{\text{The number of correctly classified samples}}{\text{The number of total samples}} \times 100\%. \tag{14}$$

CR reflects the degree of the gradient compression. The smaller is the CR value, the higher is the degree of compression. The definition of CR is as follows:

$$CR = \frac{\text{Communication times after compression}}{\text{Communication times before compression}} \times 100\%. \tag{15}$$

As CR gradually decreases, the Acc is also goes down. Weighing the two indicators and making the best decisions becomes a problem. Therefore, we introduce the Compression Balance Index (CBI) to represent the comprehensive performance of gradient compression, which is defined as follows:

$$CBI = a_1 * ACC + a_2 * (1 - CR). \tag{16}$$

a_1, a_2 are two adjustable parameters for balancing the priority of Acc and CR, $a_1 + a_2 = 1$, $a_1 > 0$, $a_2 > 0$. If in the real situation, the priority of Acc is higher than CR, it can be set that $a_1 > a_2$; otherwise, $a_1 < a_2$. If the priorities of the two indexes are the same, then $a_1 = a_2$. The higher the CBI value, the better is the comprehensive performance of gradient compression.

TABLE 2. Degree of compression and accuracy under different β.

β	Average Communication Times After Compression	Accuracy on Train Set	Accuracy on Test Set	Compression Ratio
0.1	19	93.68%	91.98%	6.33%
0.2	214	95.4%	92.4%	71.3%
0.3	270	95.3%	92.5%	90%
0.4	283	95.34%	92.52%	94.3%
0.5	290	95.46%	92.52%	96.7%
0.6	292.5	95.44%	92.44%	97.5%
0.7	296.5	95.45%	92.4%	98.83%
0.8	296.5	95.31%	92.35%	98.83%
0.9	297	95.41%	92.42%	99%
1	300	95.41%	92.41%	100%

C. SELF-ADAPTIVE THRESHOLD GRADIENT COMPRESSION EXPERIMENT

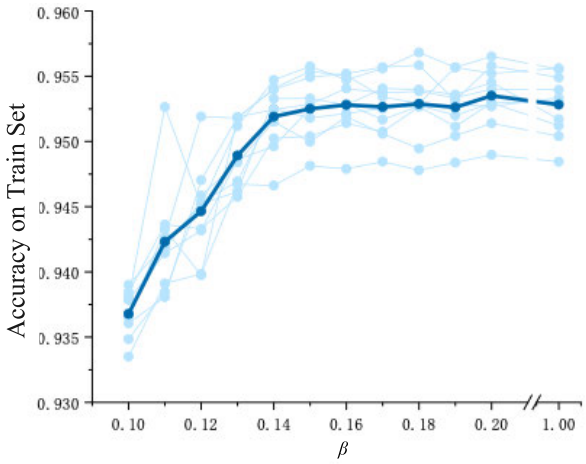
We evaluated the gradient compression performance of PAFLM using the MNIST data set, which is a handwritten data set with 60,000 train samples and 10,000 test samples. We normalized sample images to a 32×32 format so that the handwriting number was at the center of the image.

In this part of the experiment, three training nodes were included. The model structure of the training refers to the Tutorial in the Tensorflow official website, and we did not adjust it. The model structure was a three-layer MLP model, and the number of neurons in each layer was 256, 256, and 10, respectively. The data set was divided into three equal parts, and the learning nodes randomly obtained one of them. In each experiment, the nodes reacquired new data. We experimented with different hyper-parameters β and averaged the final results.

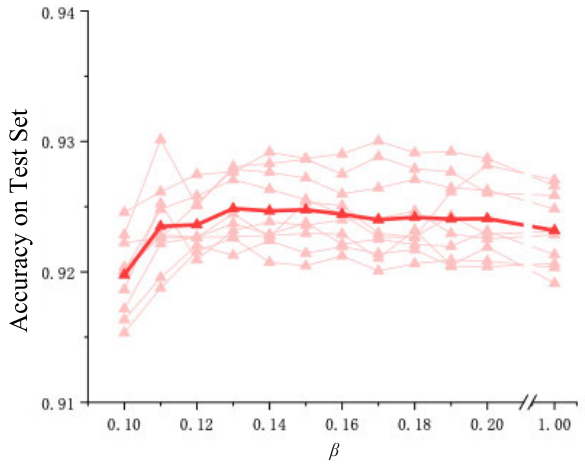
Supplementary Note: Our work focused on the federated learning framework, and we did not make too many adjustments to the model structure or the optimization algorithms. Therefore, in comparing the results of the training set and the test set, it can be seen that the model had a slight over-fitting phenomenon. Similarly, experimental indices, such as Acc were only used to compare the performance of each method, and did not evaluate the pros and cons of the model. PAFLM did not limit the type of the learning model. The model structure could be adjusted to actual problems, and to solve various learning problems such as over-fitting.

1) Gradient Compression Experiment Analysis – Acc and CR: Table 2 shows the effect of different β values on the compression ratio and accuracy. Obviously, in the interval of $\beta = [0.1, 0.2]$, the compression rate increases significantly, While in the interval $\beta = [0.2, 1]$, the compression rate raises within a smaller range. So in Fig.8, we highlight the interval $\beta = [0.1, 0.2]$.

As mentioned above, β affects the compression ratio. The smaller the β value, the smaller the compression ratio value, that is, the higher is the level of gradient communications. Normally, as the gradient communications are gradually compressed, the accuracy is also reduced. Fig. 8 shows the results of the comparison experiment with 10 different



(a) Different β values and model accuracy on train set (Thin lines indicate the accuracy on the train set, and the bold lines indicates the corresponding average accuracy.)



(b) Different β values and model accuracy on Test set (Thin lines indicate the accuracy on the test set, and the bold lines indicates the corresponding average accuracy.)

FIGURE 8. Self-adaptive threshold gradient compression.

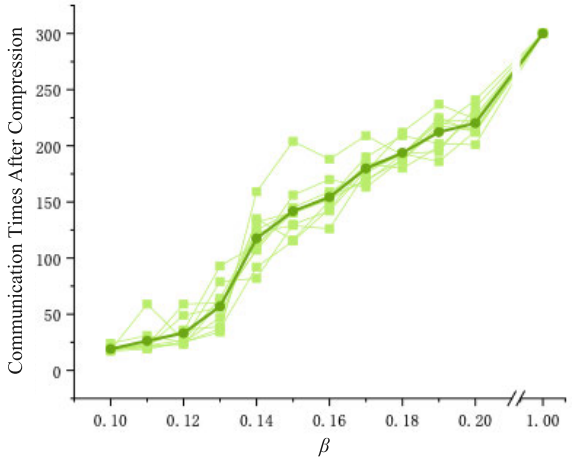


FIGURE 9. Different β and communication times.

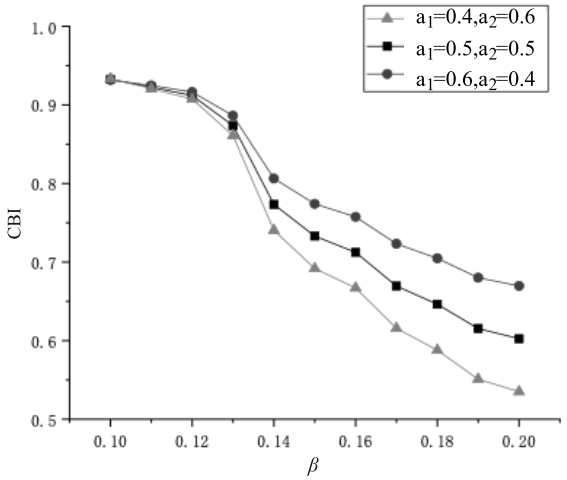


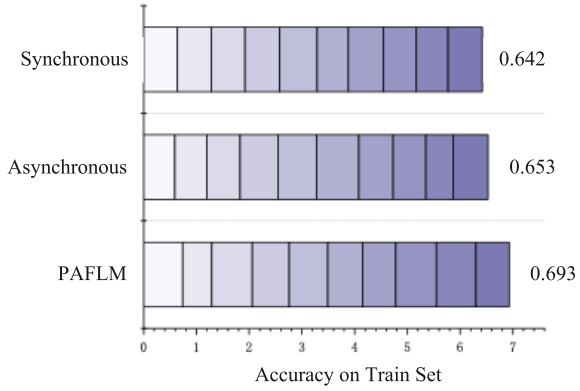
FIGURE 10. CBI of different β values in gradient compression.

hyper-parameter sets (sample number and pause interval). Light-color thin lines in Fig. 8 (a) and Fig. 8 (b) are respectively the specific data fluctuations on the test set and the train set in ten experiments. The dark bold lines in Fig. 8 (a) and Fig. 8 (b) are the test set and train set averages in these ten experiments.

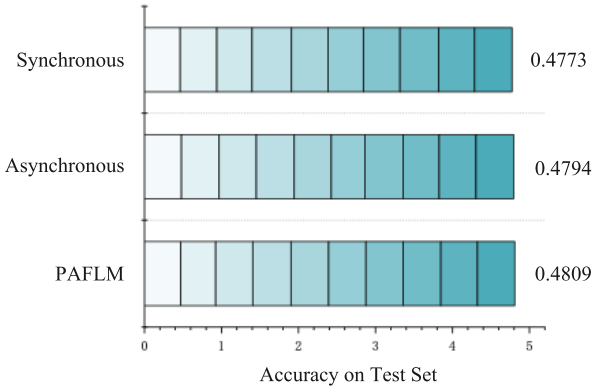
In Fig. 8, although the 10 sets of experimental data fluctuate because of individual differences, the overall accuracy shows that as the compression ratio increases, the accuracy of model also increases, which can also be seen from the corresponding dark bold lines. Interestingly, as can be seen in Fig. 8(a), in the interval $\beta = [0.14, 1]$ (corresponding to the interval $\beta = [0.13, 1]$ in Fig. 8 (b)), gradient compression has little effect on accuracy, and the results fluctuate. This is because the selected learning model has reached maximum learning. Obviously, redundant gradient communications have a lower gain in a model that has achieved saturation learning, and gradient compression helps to improve the performance of federated learning. In Fig. 9, as the β increases, the CR also increases, that is, the number of gradient communications increases.

2) Gradient Compression Experiment Analysis – CBI: In order to select the optimal β value, we calculated CBI in the interval $\beta = [0.1, 0.2]$, and the results are shown in Fig.10. It can be found that the optimum β in our experiment is $\beta = 0.1$.

We compared the performance of gradient compression in PAFLM with LAG algorithm [28]. Table 3 shows the performance comparison between threshold adaptive gradient compression algorithm in PAFLM and the LAG algorithm int three different aspects (Acc, CR, and CBI, respectively). It can be seen that CR of LAG is lower than that of PAFLM with $\beta = 0.1$ and $\beta = 0.11$. But Acc of LAG is lower than PAFLM in both the train set and the test set. Furthermore, to compare the performance of the two methods, we took three different combinations of a_1, a_2 to calculate CBI. It can be seen from Table 3 that except for $a_1 = 0.4$, $a_2 = 0.6$,



(a) Asynchronous experiment performance on train set.



(b) Asynchronous experiment performance on test set.

FIGURE 11. Asynchronous federated learning.

TABLE 3. Performance comparison between different β values of PAFLM and LAG.

Compression Algorithm	PAFLM		LAG
β	0.1	0.11	
Acc(Train set)	93.68%	94.23%	89.9%
Acc(Test set)	91.98%	92.35%	89.13%
CR	6.33%	8.77%	5.11%
CBI($a_1 = 0.4, a_2 = 0.6$)	0.9333	0.9206	0.9274
CBI($a_1 = 0.5, a_2 = 0.5$)	0.9325	0.9226	0.922
CBI($a_1 = 0.6, a_2 = 0.4$)	0.9316	0.9247	0.9167

PAFLM performed better than LAG. What is more, CBI of PAFLM with $\beta = 0.1$ is larger than LAG in three cases. In PAFLM, the value of β can be adjusted according to the actual needs.

D. ASYNCHRONOUS FEDERATED LEARNING

In order to highlight the experiment results, the experimental data of the asynchronous learning part was replaced by the Cifar10 data set. The Cifar10 data set consists of 60,000 32×32 color images of 10 classes, each of which contains 6000 images. There are 50,000 training images and 10,000 test images. We divided the training set into 500 parts. Before each experiment, the nodes randomly selected one of those as local learning data. In order to simulate asynchronous federated learning, we increased the number of learning nodes to 10. In addition, we added the pause intervals to the learning process based on Fig. 7, that is, each node paused for a period of time after the end of every epoch. The management node generated the entire stall time series. Similarly, we used the convolutional neural network code of the Tutorial on the Tensorflow official website, the model structure of which is a five-layer convolutional network model. Since asynchronous federated learning focuses on edge devices with high mobility that cannot train network models with as much stability as a server and cannot make them long-lasting, to simulate this situation, we set the number of training rounds to 500. The reduction of epochs has led to a decrease in accuracy. This problem can be solved by algorithm optimization, model tuning or by using other methods that will not be discussed too much here. The hyper-parameter values set in this experiment is only for reference, and the actual settings still need to be adjusted according to the specific problems.

We randomly selected ten different sets of hyper-parameters (subset index, pause interval) for three comparison experiments, namely no pause during training (corresponding to "synchronous" in Fig. 11), pause in training ("asynchronous"), and pause during training with dual-weights correction ("PAFLM (Dual-Weights)"). **Within the scope of consulted literature, we had not found other researchers that have improved the algorithm of asynchronous federated learning, so we could not conduct comparative experiment with more methods.**

Fig. 11 is a stacked bar graph of the average accuracy of ten devices under five different hyper-parameter sets.Comparing the two subgraphs, it can be seen that the accuracy of each experiment is fluctuating. This is because the experimental node has a small amount of data, and each subset is largely different. However, the overall trend shows that asynchronous learning has better performance than synchronous learning. Asynchronous learning is similar to a regularization process that can to a certain extent prevent the over-fitting of learning. Moreover, PAFLM shows better performance than both asynchronous learning and synchronous learning.

VII. SUMMARY

In this paper, we propose a Privacy-Preserving Asynchronous Federated Learning Mechanism (PAFLM) for Edge Network Computing to satisfy the realistic needs of learning multi-party data without sharing private information. PAFLM gives learners more freedom and privacy protection without compromising accuracy of training. Participants learn from their own sensitive data sets locally. After a round of training, the participants check whether the current round meets the conditions for communicating with the parameter server according to the self-test condition. If it is satisfied, learning nodes upload the gradient to the parameter server. All nodes perform the above steps asynchronously without waiting for

the remaining nodes or synchronizing the learning process. Throughout the federated learning, the nodes only communicate with the parameter server, not having obtained any information about the other nodes except for the global parameters that are jointly maintained.

We demonstrate two aspects of PAFLM: self-adaptive threshold gradient compression and asynchronous federated learning, and then conduct related experiments. In self-adaptive threshold gradient compression, there are many deficiencies due to simply introducing communication compression ratio or compressing gradient communications based on the fixed threshold. The self-adaptive threshold gradient compression algorithm can automatically adapt to the change of gradient in each model-training process and calculate the appropriate threshold to compress the gradient communications. Since the gradient data indirectly reflects the information of the training samples, attackers can deduce the sample data from the effective gradient information. Therefore, reducing gradient communications can effectively reduce the possibility of privacy leakage.

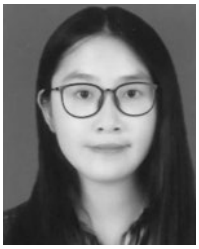
Multiple problems, such as uneven learning samples and different learning progress, arise in asynchronous federated learning due to high mobility of edge nodes. Obviously, it is unreasonable to expect the nodes with large differences to update the global parameters equally. Therefore, we introduce dual-weights correction to solve the problem of unbalanced learning status in asynchronous federated learning.

As a relatively new research content, asynchronous learning still has much room for discussion. As mentioned above, asynchronization is caused by many factors, and different causes require different solutions. In future work, we will discuss different attenuation functions and look for a better attenuation function to fit the attenuation requirements.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," presented at the IEEE Int. Conf. Comput. Vis. (ICCV), Dec. 2015.
[2] D. Shultz, "When your voice betrays you," *Science*, vol. 347, no. 6221, p. 494, Jan. 2015, doi: 10.1126/science.347.6221.494.
[3] T. Jian, "Analysis on the application of Internet of Things in Chengdu radio and TV network," *Telecom World*, vol. 26, no. 5, pp. 35–36, 2019.
[4] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, Dec. 2016.
[5] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 416–464, 1st Quart., 2018.
[6] T. D. Dang and D. Hoang, "A data protection model for fog computing," presented at the 2nd Int. Conf. Fog Mobile Edge Comput. (FMEC), May 2017.
[7] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the suitability of fog computing in the context of Internet of Things," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 46–59, Jan. 2018.
[8] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[9] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing towards balanced delay and power consumption," *IEEE Internet Things J.*, to be published.
[10] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the Internet of Things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, Aug. 2016.
[11] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," *Google Res. Blog*, vol. 3, Apr. 2017. [Online]. Available: https://www.googblogs.com/federated-learning-collaborative-machine-learning-without-centralized-training-data/
[12] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," 2019, *arXiv:1902.01046*. [Online]. Available: http://arxiv.org/abs/1902.01046
[13] Y. J. Kim and C. S. Hong, "Blockchain-based node-aware dynamic weighting methods for improving federated learning performance," presented at the APNOMS 20th Asia–Pacific Netw. Oper. Manage. Symp., Sep. 2019.
[14] D. Conway-Jones, T. Tuor, S. Wang, and K. K. Leung, "Demonstration of federated learning in a resource-constrained networked environment," presented at the SMARTCOMP IEEE Int. Conf. Smart Comput., Jun. 2019.
[15] U. Majeed and C. S. Hong, "FLchain: Federated learning via MEC-enabled blockchain network," presented at the APNOMS 20th Asia–Pacific Netw. Oper. Manage. Symp., Sep. 2019.
[16] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang, "Blockchain and federated learning for privacy-preserved data sharing in industrial IoT," *IEEE Trans. Ind. Inform.*, vol. 16, pp. 4177–4186, Jun. 2020.
[17] J. Weng, J. Weng, J. Zhang, M. Li, Y. Zhang, and W. Luo, "DeepChain: Auditable and privacy-preserving deep learning with blockchain-based incentive," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2019.2952332.
[18] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. Mcmahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," presented at the ACM SIGSAC Conf. Comput. Commun. Secur., 2017.
[19] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Federated learning for ultra-reliable low-latency V2V communications," presented at the GLOBECOM IEEE Global Commun. Conf., Dec. 2018.
[20] T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi, "Federated learning of predictive models from federated electronic health records," *Int. J. Med. Informat.*, vol. 112, no. 1, pp. 59–67, Apr. 2018.
[21] F. Seideetal, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs," presented at the INTERSPEECH 15th Annu. Conf. Int. Speech Commun. Assoc., 2014.
[22] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," presented at the NIPS Adv. Neural Inf. Process. Syst., 2017.
[23] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.
[24] N. Dryden, T. Moon, S. A. Jacobs, and B. V. Essen, "Communication quantization for data-parallel training of deep neural networks," presented at the MLHPC 2nd Workshop Mach. Learn. HPC Environ., Nov. 2016.
[25] C. Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan, "Adacomp: Adaptive residual gradient compression for data-parallel distributed training," presented at the 32nd AAAI Conf. Artif. Intell., 2018.
[26] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," presented at the INTERSPEECH 16th Annu. Conf. Int. Speech Commun. Assoc., 2015.
[27] A. Fikri Aji and K. Heafield, "Sparse communication for distributed gradient descent," 2017, *arXiv:1704.05021*. [Online]. Available: http://arxiv.org/abs/1704.05021
[28] T. Chen, G. Giannakis, T. Sun, and W. Yin, "LAG: Lazily aggregated gradient for communication-efficient distributed learning," presented at the NIPS Adv. Neural Inf. Process. Syst., 2018.
[29] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," presented at the 22nd ACM SIGSAC Conf. Comput. Commun. Secur., Sep. 2015.

XIAOFENG LU received the Ph.D. degree from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2010. During his Ph.D., he held visiting scholar positions at the Computer Laboratory, University of Cambridge, U.K. He is currently an Associate Professor with the School of Cyberspace Security, Beijing University of Post and Telecommunications. His main research interests include cyberspace security, information security, and artificial Intelligence.



YUYING LIAO received the B.S. degree in information security from the Nanjing University of Posts and Telecommunications, China, in 2017. She is currently pursuing the M.S. degree with the School of Cyberspace Security, Beijing University of Posts and Telecommunications, China. Her current research interests include federated learning, edge computing, privacy preservation, and AI security.

PIETRO LIO is currently a Professor with the Computer Laboratory, University of Cambridge, U.K., and also a Fellow and Director of Studies at Fitzwilliam College, University of Cambridge. He is currently modeling biological processes on networks, modeling stem cells, as well as developing transcription and phylogenetic applications on a grid environment. He is also interested in bio-inspired design of wireless networks and epidemiological networks.

PAN HUI (Fellow, IEEE) received the bachelor's and M.Phil. degrees from The University of Hong Kong, and the Ph.D. degree from Computer Laboratory, University of Cambridge. During his Ph.D., he was also affiliated with Intel Research Cambridge. He is currently a Professor of computer science and engineering with The Hong Kong University of Science and Technology. He is also a Distinguished Scientist with Deutsche Telekom Laboratories (TLabs), Berlin. His research interests include delay tolerant networking, mobile networking and systems, planet-scale mobility measurement, social networks, and the application of complex network science in communication system design.