Distributed hierarchical deep optimization for federated learning in mobile edge computing

Xiao Zheng, Syed Bilal Hussain Shah, Ali Kashif Bashir,
Raheel Nawaz, Umer Rana

PII: S0140-3664(22)00275-4
DOI: https://doi.org/10.1016/j.comcom.2022.07.028
Reference: COMCOM 7219

To appear in: *Computer Communications*

Received date : 30 March 2022
Revised date : 13 May 2022
Accepted date : 20 July 2022

Please cite this article as: X. Zheng, S.B.H. Shah, A.K. Bashir et al., Distributed hierarchical deep optimization for federated learning in mobile edge computing, *Computer Communications* (2022), doi: https://doi.org/10.1016/j.comcom.2022.07.028.

Distributed Hierarchical Deep Optimization for Federated Learning in Mobile Edge Computing

Xiao Zheng[a], Syed Bilal Hussain Shah [b,*], Ali Kashif Bashir[c], Raheel Nawaz[d], Umer Rana[e]

[a]*School of Computer Science and Technology, ShanDong University of Technology,Zibo and 255000, China*
[b]*School of Computing and Mathematics Manchester Metropolitan University, United Kingdom*
[c]*Department of Computing and Mathematics, Manchester Metropolitan University, United Kingdom*
[d]*Department of Computing and Mathematics, Manchester Metropolitan University, United Kingdom*
[e]*School of Computer Science and Informatics, Cardiff University ,Cardiff, United Kingdom*

Abstract

Deep learning has recently attracted great attention in many application fields, especially for big data analysis in the field of edge computing. Federated learning, as a promising machine learning technology, applies training data on distributed edge nodes to design shared learning systems to protect data privacy. Due to the system update in federated learning is at the expense of parameter exchange between edge nodes, it is extremely bandwidth consuming. A novel distributed hierarchical tensor depth optimization algorithm is proposed, which compresses the model parameters from the high-dimensional tensor space to a union of low-dimensional subspaces to reduce bandwidth consumption and storage demands of federated learning. In addition, an update method based on hierarchical tensor back propagation is developed by directly calculating the gradient of low-dimensional parameters to reduce the memory requirement and improve the training efficiency

[*]Corresponding author

Email addresses: `xiao_zheng0910@163.com` (Xiao Zheng), `S.Shah@mmu.ac.uk` (Syed Bilal Hussain Shah), `Dr.alikashif.b@ieee.org` (Ali Kashif Bashir), `R.Nawaz@mmu.ac.uk` (Raheel Nawaz), `ranaof@cardiff.ac.uk` (Umer Rana)

caused by edge node training. Finally, a large number of simulation experiments were performed to evaluate performance based on classical data sets with different local data distributions. Experimental results show that the proposed algorithm reduces the burden of communication bandwidth and the energy consumption of edge nodes.

Keywords: Federated learning, Back-propagation, Hierarchical tensor, Edge computing.

1. Introduction

With the widespread rise of the Internet of Things, the amount of data generated by connected devices in IoT is enormous. Therefore, it is not practical to upload large amounts of data directly to the remote cloud for further processing and analysis, which will cause severe network congestion and intolerable transmission delays[1]. With the rapid development of edge computing, various edge devices such as those capable of storing, processing and analyzing IoT data are deployed, in addition to learning and analyzing tasks in collaboration with remote cloud. Meanwhile, deep learning has also been successfully applied to industrial big data analysis. Many models have been developed to learn inherent features from large amounts of raw data. For example, the deep computing model based on tensor data representation and hierarchical feature learning is extended to the tensor space[2, 3]. However, some significant challenges exist in distributed edge node training depth computing models, especially in IoT environments. First, uploading local data from edge nodes to remote clouds is not secure due to increasing awareness and demands for data security and user privacy[4, 5].This is true for most industries, and it is generally not permitted to share data between different companies, or even between different parts of the same company. Secondly, due to resource constraints, it is not practical to conduct model learning on these low-end nodes because the parameters of deep computing models are very large, and training such models usually requires expensive hardware resources such as large storage and memory units. In order to solve the problem of data security, a "federated learning" technology is proposed to solve the problem [6].Federated learning can not only learn efficient sharing model, but also effectively break the data privacy barrier between different users. In IoT applications, for example, many factories typically deploy similar assembly lines with many IoT nodes. However, the amount of data collected from just one company is not enough to learn a valid model. In addition, companies generally do not share their data because of data security concerns. The industry alliance aims to extract more knowledge from multiple companies without exposing raw data to others[7]. Federated learning can be used to help organizations train with each other in more efficient models to accomplish such tasks.

When federated learning needs to exchange parameters between edge node and cloud server, the problem of training bottleneck will occur due to large parameter model and high communication cost. Edge devices, in particular, have problems with computing, storage capacity, and poor connectivity, which can make things worse. In order to reduce the cost of communication, several existing methods have been designed by reducing the frequency of communication [8–10] or using recompression technology[11–13]. The communication frequency is reduced by reducing the number of local updates or improving the convergence speed of the training algorithm. Its weight compression techniques include gradient sparsity and quantization, which indicates that a small part of parameters are selected for updating and quantization in each update.

A distributed hierarchical tensor depth optimization algorithm (DHT-DOA) based on federated learning is proposed. The proposed algorithm uses hierarchical tensors decomposition (HTD) to achieve low-rank approximation of weight tensors, thus achieving the purpose of reducing the communication bandwidth between edge nodes and servers, and reducing the storage requirements of edge nodes. The main contributions of this paper are as follows:

1) A new DHT-DOA based on federated learning under edge computing framework model is proposed. The model parameters are compressed from high-dimensional tensor space to a group of low-dimensional subspaces so as to reduce the bandwidth consumption and storage requirements of edge node distributed training.

2) An updated algorithm based on gradient descent hierarchical back-propagation is designed to train the edge node model. It can reduce the memory requirement of edge node training by directly computing the gradient of low dimensional parameters.

3) In this paper, real datasets are used to evaluate the performance of the proposed algorithm. Experimental results show that the proposed algorithm can not only achieve better compression efficiency, but also reduce the total energy consumption of edge nodes.

The rest of this paper is structured as follows: The second section gives relevant introduction. The third section introduces the relevant background, concepts and preparatory knowledge. The fourth section introduces the system model and problems. Section 5 introduces and analyzes the proposed algorithm, and Section 6 provides performance evaluation. Finally, section 7 summarizes this article.

2. Related Works

In recent years, with the continuous rise of industrial information, federal learning has provided unlimited prospects in secure Internet of Things. A real IoT environment with federated learning is designed for human-machine collaboration scenarios[7]. A privacy-protected data sharing strategy combining federated learning with blockchain is designed to improve IoT efficiency sharing and security issues. In addition, federated learning can be spread to other areas such as network attack detection for iot devices, edge computing caching and offloading, and autonomous driving safety prediction in vehicle networks[14]. When sensitive information such as driver's physical condition and action cannot be uploaded to the cloud, for example, federated learning can be used to predict vehicle collision[15]. Wei.et al. [16] can be investigated for a general understanding of federated learning and its specific application background in edge networks.

From other perspectives, some measures are at the expense of reducing communication costs by using weight compression techniques such as gradient thinning and quantization. Gradient sparsity is based on the pre-set gradient threshold or constant sparsity rate. Only a small number of parameters are selected to update and then used for distributed training[11]. Gradient index lowers data transmission by converting gradients into low-precision values[13]. Wiedemann [17] et al. designed a compression structure to better transmission potency and realize more precision, which integrates with gradient sparsity, binarized sparse weight and position index of sparse weight to carry out Columbus coding. Lin et al. studied a strategy named Depth Gradient Compression (DGC), which uses gradient sparsity to compress communication bandwidth, and furthermore uses impetus rectification and local gradient prunning to prevent precision loss caused by gradient sparsity[11]. However, the above strategies all use compression and decompression measures; Its system schema is compressed before being passed to the server, retrieved from the server, and then decompressed.

Compared with the above work, the difference of this paper is that it aims to loss the bandwidth for federated learning by applying the low-rank representation of weight tensors. As a particularly effective method, tensor decomposition can use low-rank representation to proximate the weight of the system, and reduce the parameters greatly so that the classification accuracy will not decrease too much, but also greatly speed up its training speed [18]. In addition, a tensor series set fuzzy deep convolution system is designed, in which the system parameters are compressed by applying the tensor series set decomposition technique [19]. However, at present, the application of tensor decomposition to the compression of depth computing system cannot produce successful compression rate. And most of the previous research work only concentrates on an alone low-end device, but does not exist in the federated learning system structure.

3. Concept description and Preliminaries

Define 1: Matrixization is the expansion (or flattening) operation of a matrix (or tensor)[20]. A tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$ with d different modes is adopted.Suppose these modes are divided into two disjoint subsets: $p = \{p_1, ..., p_k\}$ and $q = \{q_1, ..., q_{d-k}\}$. Meanwhile,these modes are trivialized by dividing the two groups into row and column indexes, The result is that the system used for training still has parameters equivalent to that of the primary system on the nodes, thus reducing its training efficiency. In addition, it is the extra communication required to spread the location message of those sparse gradients.

$$\boldsymbol{A}_t \in \mathbb{R}^{(I_{p_1} \ldots I_{p_k}) \times (I_{q_1} \ldots I_{q_{d-k}})} \tag{1}$$

and it has properties: $(\boldsymbol{A}_t)_{(i_{p_1},...,i_{p_k}),(i_{q_1},...,i_{q_{d-k}})} \equiv \mathcal{A}_{i_1,...,i_d}$, the set $\{1, ..., I_1\} \times ... \times \{1, ..., I_d\}$, $\forall i_1, ..., i_d$.

Define 2: The dimesion tree DT_I is a binary tree, and each node $n \in DT_I$ consists of the subset of the mode $\Phi = \{1, ..., d\}$ and the depth $h = [log_2 d] = \min\{i \in N, i \geq log_2 d\}$, If the root node is an Φ, then each leaf node is a single node and each inner node is a disjoint union of its two children, respectively. To simplify the notation, the set of leaf nodes and internal nodes is defined by $L(DT_I)$ and $I(DT_I)$ respectively.

Define 3: Define $(\delta_t)_{t \in DT_I}$ be a rank probability distribution for the DT_I[21]. The hierarchical rank $(\delta_t)_{t \in DT_I}$ of the tensor $\mathcal{A} \in \mathbb{R}^I$ is denoted as

$$\delta_t \equiv rank(\boldsymbol{A}_{(t)}), \forall t \in DT_I \tag{2}$$

Define 4: Define (δ_t) be the rank probability distribution with $t \in DT_I$ being the node. The matrix $\boldsymbol{M}_t \in \mathbb{R}^{I_t \times r_t}$ and the tuple $(\boldsymbol{M}_t)_{t \in DT_I}$ of frames are named t-frame and frame tree, of which $r_t := rank(\boldsymbol{A}_t) \leq \delta_t$. Every internal node t has two successors $S(t) = \{t_l, t_r\}$. The tensor $\mathcal{T}_t \in \mathbb{R}^{r_t \times r_{t_1} \times r_{t_2}}$, where is the coefficient for the expression of $\boldsymbol{M}_t$ by $\boldsymbol{M}_{t_l}$ and $\boldsymbol{M}_{t_r}$.

$$\boldsymbol{M}_t = (\boldsymbol{M}_{t_l} \otimes \boldsymbol{M}_{t_r})\boldsymbol{T}_t \qquad \text{with}(\boldsymbol{T}_t)_{i,j,k} \equiv (\mathcal{T}_t)_{i,j,k} \tag{3}$$

is the transfer tensor.

A. Hierarchical Tensor Decomposition(HTD)

HTD uses a matrixized hierarchy to decompose higher-order tensors into a series of matrices or lower-order tensors. HTD correspond to dimension trees whose nodes are described by subsets of tensor mode. The design of HTD is based on t-frames and transfer tensors to obtain the node cluster t of dimension tree, among which t-frames $\boldsymbol{M}_t$ is obtained by recursive application (3). The result is that the tensor $\mathcal{A}$ can be decomposed into a sequence of t-frames and transfer tensors. The HTD case of an eighth-order tensor is shown in **Fig.1**. For example, t-frames $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are derived from r_t left singular vectors of $\mathcal{A}_1$ and $\mathcal{A}_2$. In addition, $\boldsymbol{M}_{(1,2)}$ can also be derived from the same operation of mode $\mathcal{A}_{(1,2)}$. Therefore, the transfer tensor $\mathcal{T}_{(1,2)}$ can be obtained according to (3).

4. Problem definition and formulation

This paper adopts a system model of federated learning in edge computing architecture, as shown in **Fig.2**. The model consists of iot nodes, wireless access points(WAP) and a cloud server. IoT nodes can be considered edge nodes, deployed in different locations. IoT nodes connect to cloud servers through WAPs such as routers and gateways. This model is usually used in federated learning models. Based on this, this paper studies the collaborative

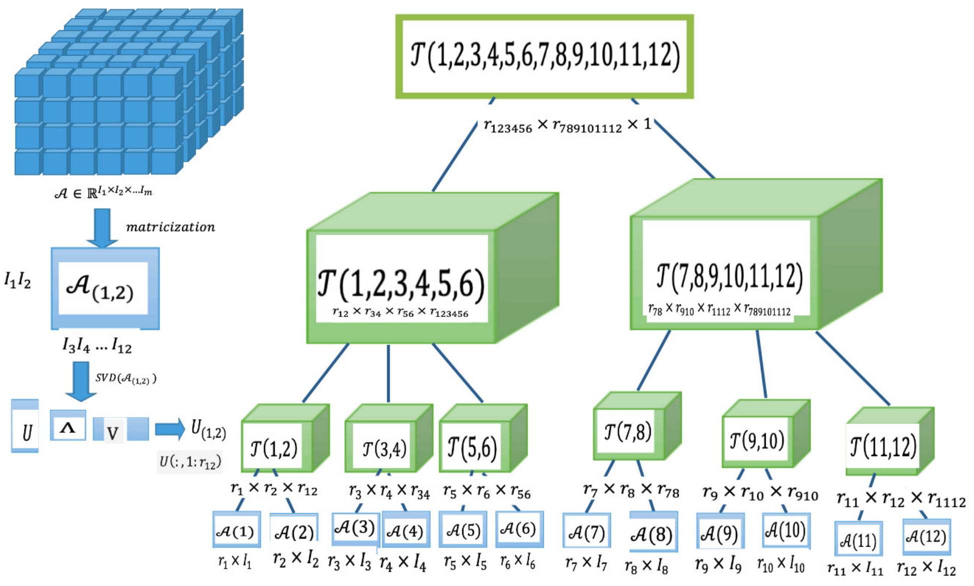

$\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots I_m}$
matricization
$I_1 I_2$
$\mathcal{A}_{(1,2)}$
$I_3 I_4 \ldots I_{12}$
$SVD(\mathcal{A}_{(1,2)})$
U
Λ
V
$U_{(1,2)}$
$U(:,1:r_{12})$
$\mathcal{T}(1,2,3,4,5,6,7,8,9,10,11,12)$
$r_{123456} \times r_{789101112} \times 1$
$\mathcal{T}(1,2,3,4,5,6)$
$r_{12} \times r_{34} \times r_{56} \times r_{123456}$
$\mathcal{T}(7,8,9,10,11,12)$
$r_{78} \times r_{910} \times r_{1112} \times r_{789101112}$
$\mathcal{T}(1,2)$
$\mathcal{T}(3,4)$
$\mathcal{T}(5,6)$
$\mathcal{T}(7,8)$
$\mathcal{T}(9,10)$
$\mathcal{T}(11,12)$
$r_1 \times r_2 \times r_{12}$
$r_3 \times r_4 \times r_{34}$
$r_5 \times r_6 \times r_{56}$
$r_7 \times r_8 \times r_{78}$
$r_9 \times r_{10} \times r_{910}$
$r_{11} \times r_{12} \times r_{1112}$
$\mathcal{A}(1)$ $r_1 \times I_1$
$\mathcal{A}(2)$ $r_2 \times I_2$
$\mathcal{A}(3)$ $r_3 \times I_3$
$\mathcal{A}(4)$ $r_4 \times I_4$
$\mathcal{A}(5)$ $r_5 \times I_5$
$\mathcal{A}(6)$ $r_6 \times I_6$
$\mathcal{A}(7)$ $r_7 \times I_7$
$\mathcal{A}(8)$ $r_8 \times I_8$
$\mathcal{A}(9)$ $r_9 \times I_9$
$\mathcal{A}(10)$ $r_{10} \times I_{10}$
$\mathcal{A}(11)$ $r_{11} \times I_{11}$
$\mathcal{A}(12)$ $r_{12} \times I_{12}$

Figure 1: Example HTD diagram of a twelve-order tensor.

learning sharing model using federated learning instead of transferring raw data from multiple edge nodes to the cloud server.

Define M edge nodes $\{E_1, ..., E_M\}$, and use their respective local data sets $\{D_1, ..., D_M\}$ to train the learning model. The process of federated learning is described below. Each node first iterates through its local dataset to learn the local model using stochastic gradient descent several times, and then sends it to the cloud. The cloud collects all models from multiple nodes and performs a federated averaging algorithm to arrive at a global model.Therefore, the model is sent back to the node to update its local model. The communication process ends.In each round t, the model is updated by the following equation as follows:

$$\theta_i^t = \theta_i^{t-1} - \gamma g_i^t ; \theta^t = \sum_{i=1}^{N} \frac{n_i}{n} \theta_i^t \tag{4}$$

where θ_i^t and g_i^t are the updated local model and the gradient using back propagation at the edge node E_i after round t, respectively. θ^t is the global

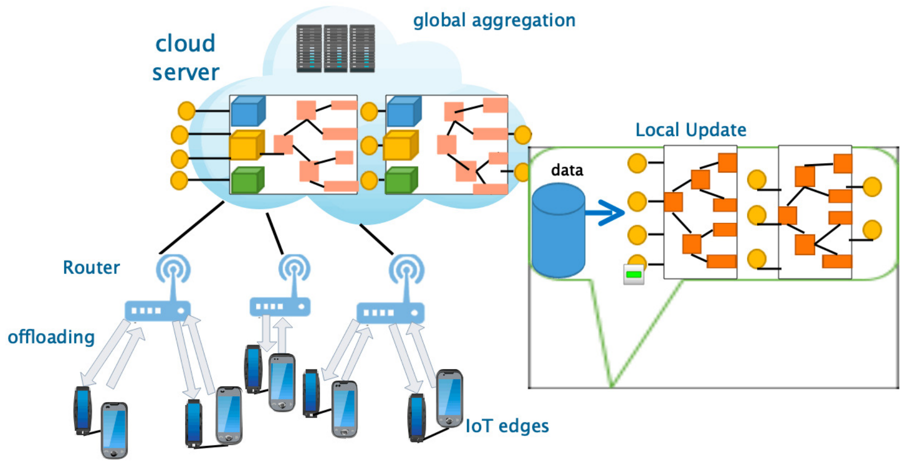


Figure 2: System model.

model after the cloud performs federated averaging after round t, γ is the learning rate, n_i is the data of node E_i, and n is the total number of data. However, updating the model requires a large amount of high-bandwidth energy consumption due to the large number of parameters involved in the model θ_i^t at the node and the need for communication exchanges with the cloud at each round. This paper aims to study a distributed deep computing model, which uses structural update to map local model θ_i^t to lower dimensional space via HTD to reduce communication bandwidth and energy consumption of edge nodes.

5. Algorithm Design and Analysis

A. Distributed Deep Convolution Optimization Algorithm Based on HTD

In this paper, a distributed deep convolution algorithm based on HTD in edge computing framework is proposed. The designed distributed learning process involves two processes: local update and global aggregation.For local update, each edge node uses gradient descent algorithm and its local training data set to train the learning model independently. For global aggregation, the cloud aggregates the weights from different edge nodes obtained by local update, and then returns the updated weights to edge nodes for the next iteration update process.

Instead of sending all data sets to the cloud for processing, edge nodes can cooperate to share and learn a common model.In local update processing, deep convolutional neural network (CNN) based on tensors is used as the computing model, because it has been proved to achieve better performance than traditional CNN. The output of the above three types of layers can be expressed as follows:

$$f(\mathcal{W}^{(1)} * \mathcal{A} + b^{(1)}) = \mathcal{H}(\mathcal{W}^{(1)}; b^{(1)}) \tag{5}$$

$$f(\mathcal{W}^{(2)} \cdot \mathcal{A} + b^{(2)}) = \mathcal{P}(\mathcal{W}^{(2)}; b^{(2)}) \tag{6}$$

$$f(\mathcal{W}^{(3)} \odot \mathcal{A} + b^{(3)}) = \mathcal{C}(\mathcal{W}^{(3)}; b^{(3)}) \tag{7}$$

where $\theta \equiv \left\{\mathcal{W}^(j); b^{(j)} | j \in \{1, 2, 3\}\right\}$ represent the model parameters. $*$ and $\odot$is tensor convolution and multi-point product operation, $f(\cdot)$ is the activation function, $\mathcal{W}$ and b indicate the corresponding weight and bias tensors of the input tensors $\mathcal{A} \in \mathbb{R}^{M_1 \times M_2 \times \ldots \times M_d}$. $\mathcal{H}(\cdot)$, $\mathcal{P}(\cdot)$ and $\mathcal{C}(\cdot)$ represent the feature extractors of the classifier at the convolutional layer, pooling layer and full connection layer, respectively. Uploading the above parameters to the cloud during local updates consumes bandwidth.HT mode is used to represent the weight of learning model of each edge node in order to reduce communication cost. In particular, the weight tensor $\mathcal{W} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$ is decomposed by HTD into d factor matrices $\boldsymbol{M}_t \in \mathbb{R}^(I_t \times p_t)$ and the $d-1$ third-order tensor $\mathcal{T}_t \in \mathbb{R}^{p_t \times p_{t_l} \times p_{t_r}}$. Therefore, the weight is expressed by the following compact mode:

$$\mathcal{W} = \otimes_{t \in \mathcal{I}(DT_\mathcal{I})} (\boldsymbol{M}_{t_l} \otimes \boldsymbol{M}_{t_r}) \boldsymbol{T}_t \text{with} (\boldsymbol{T}_t)_{(i,j,k)} \equiv (\mathcal{T}_t)_{(i,j,k)} \tag{8}$$

where $t \in \mathcal{I}(DT_\mathcal{I})$ belongs to node t of dimension tree $\mathcal{I}(DT_\mathcal{I})$, $\boldsymbol{M}_{t_l}$, $\boldsymbol{M}_{t_r}$ and $\mathcal{T}_t$ are t-frame and transfer tensors of each internal node of dimension tree in tensor mode respectively.

For the learning model, the typical loss function of each data sample at edge node E_i is defined on the training data as follows:

$$J_i(\theta) = \frac{1}{2}\sum_{n=1}^{m_i}(l_\theta(x_i^n) - y_i^n)^2 \tag{9}$$

where $l_\theta(\cdot)$ is the output of the deep convolution computing model at each edge node, and x_i^n, y_i^n comes from the training set $\{(x_i^1, y_i^1), (x_i^2, y_i^2), ..., ((x_i^{m_i}, y_i^{m_i}))\}$ sample node E_i.

The proposed distributed deep convolution model algorithm is shown in Algorithm 1. Firstly, the cloud starts the learning process with the initial model parameter θ_1, and then randomly selects I nodes out of M edge nodes for parameter update. This process is similar to dropout regularization to avoid overfitting. In particular, the weight tensor is represented by a dimension tree whose corresponding hyperparameter is $\theta \equiv \left\{M_t^j, T_t^j, b^i, \mathcal{W}^(2) | i \in \{1,2,3\}, j \in \{1,3\}, t \in \mathcal{I}(DT_\mathcal{I})\right\}$ transformed from $\theta \equiv \left\{\mathcal{W}^{(j)}; b^{(j)} | j \in \{1,2,3\}\right\}$.

Hereafter, the forward propagation algorithm in Algorithm 2 is used to train edge nodes and perform local update. The shared model performs HT back-propagation algorithm to obtain parameter updates (e.g., lines 24-32 in Algorithm 1) based on the gradient descent algorithm. Specifically, in the back propagation algorithm, the error term of the output layer (e.g., line 26 in Algorithm 1) is computed first, and then the error term of the hidden layer (e.g., line 29 in algorithm 1) is computed. In the similar operation of the output layer, the error term of the hidden layer is firstly computed according to the propagation gradient of the following layer (e.g., lines 5, 12 and 17 in Algorithm 3). The gradient of the hyperparameter is computed based on the error term of each layer (as shown in lines 6-9, 13-14 and 18-21 in Algorithm 3).

The main purpose of Algorithm 3 in the backpropagation process is based on the HT back-propagation algorithm described in Algorithm 4. To better explain the algorithm, this paper sets up a cell in the dimension tree, which consists of a parent node and two child nodes. Then introduce Kronecker's gradient. Two matrices $\boldsymbol{A}_1 \in \mathbb{R}^{x_1 \times x_2}$ and $\boldsymbol{A}_2 \in \mathbb{R}^{x_3 \times x_4}$ are defined, and the Kronecker gradient between $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ is computed as follows: $\boldsymbol{X} = \frac{\partial(A_1 \otimes A_2)}{\partial A_{s \in \{1,2\}}} = \frac{\partial B}{\partial A_{s \in \{1,2\}}} = \sum_{i=1}^{x_1 x_3} \sum_{j=1}^{x_2 x_4} \frac{\partial B(i,:)}{\partial A_s(:,j)}$.

In Algorithm 4, the parent node is firstly computed as the gradient of the root node, which includes the transfer tensor of the parent node and the

Algorithm 1 Algorithm 1: Distributed Hierarchical Deep Convolution Model Algorithm

```
1: Cloud computing: **Input:** the number of edge nodes $M$, the maximum
   number of communication rounds $t_{max}$, rank $R$, learning ratio $\alpha$, thresh-
   old $\rho$
   **Input:** $\theta^*$
   initialize $\theta_0$
   training edge nodes $N$=max$(M \cdot \alpha, 1)$
2: for $t = 1 : t_{max}$ do
3:   training nodes set $\Omega$=(random set of $N$ nodes);
4:   for user $i \in \Omega$ in parallel do
5:     $\theta_i^t$=nodeupdate($\theta_i^{t-1}, R, \rho$;
6:     $\theta^t = \sum_{i=1}^{N} \frac{n_i}{n} \theta_i^t$;
7:     Edge computing:
8:     nodeupdate($\theta_i, R, \rho$)
9:     **Input:** rank $R$, threshold $\rho$, the maximum number of local iterations
       $r_{max}$, $\theta_i$;
10:    **Output:** $\theta_i^*$;
11:    $\mathcal{T}$=(batch decomposition of local data sets $D_i$), $y^1 = x_i^1$
12:    for $r = 1 : r_{max}$ do
13:      for each bath $b \in \mathcal{T}$ do
14:        for $tr = 1 : n_{tr}$ do
15:          $z^{tr+1} = Edge_F P(tr, \theta_i, y^{tr}) y^{(tr+1)} = f(z^{(tr+1)})$;
16:          if $J(\theta_i) > \rho$ then
17:            if $tr = n_{tr}$ then
18:              $sigma^(tr) = (f(z^{tr}) - f(z^{tr+1}))f'(z^{tr})$;
19:            end if
20:          end if
21:          for $tr = n_{tr} - 1 : -1 : 1$ do
22:            $\nabla\theta_i, \sigma^{tr} = Edge_F P(tr, \theta_i, z^{tr}, \sigma^{tr+1})$;
23:            $\theta_i^* = \theta_i - \gamma\nabla\theta_i$;
24:          end for
25:        end for
26:      end for
27:    end for
28:  end for
29: end for
```

Algorithm 2 Algorithm 2: Forward Propagation Algorithm at Edge Nodes

```
Edge_FP($tr, \theta, z^{tr}$ );
Input: $tr, \theta, z^{tr}$;
Output: $z^{tr+1}$;
if Layer(tr)=convolutional lay then
    $\mathcal{W}^1 = \otimes_{t\in\mathcal{I}(DT_\mathcal{I})}(M_{t_l} \otimes M_{t_r})T_t$;
    $z^{tr+1} = \mathcal{W}^1 \times z^{tr} + b^1$;
    if Layer(tr)=pooling layer then
        $z^{tr+1} = \mathcal{W}^2 \times pooling(z^(tr)) + b^2$;
        if Layer(tr)=fully layer-connected layer then
            $\mathcal{W}^3 = \otimes_{t\in\mathcal{I}(DT_\mathcal{I})}(M_{t_l} \otimes M_{t_r})T_t$;
            $z^{tr+1} = \mathcal{W}^3 \odot z^{tr} + b^3$
        end if
    end if
end if
```

t-frames of the two child nodes (as shown in Algorithm 4, lines 8-10). Next, compute the gradient of the other parent node that is the inner node. Since the child node belongs to the parent node of the last layer, the process can be divided into two processes. In Algorithm 4, the gradient is calculated when the parent node belongs to the left or right child node of the last level cell (as shown in algorithm 4, lines 19-21). The edge nodes then update the local model in parallel and aggregate it in the cloud.

In the global aggregation process, the cloud updates the model parameters by calculating the average weight from multiple edge nodes. After that, the cloud sends the average weight back to the edge nodes and starts new iterations until the model converges or reaches the maximum number of communication rounds.

B. Performance Analysis of the Proposed Algorithm

First, HTD is used to analyze the compression ratio index of model parameters. Obviously, the higher the compression rate of model parameters, the less bandwidth required to transmit parameters from edge nodes to cloud servers. We use HTD to decompose tensor weights of tensor convolution layer and full connection layer.

For the convenience of analysis, this paper assumes that $\mathcal{A} \in \mathcal{R}^{I_1\times I_2\times\ldots\times I_D}$ represents the weight tensor to be transmitted in each training iteration on each node, where D, I and R are the order, the maximum dimension, and the

Algorithm 3 Algorithm 3: Back Propagation Algorithm at Edge Nodes

```
1: Edge_BP($tr, \theta, z^{tr}, \sigma^{tr+1}$ );
2: **Input:** $tr, \theta, z^{tr}, \sigma^{tr+1}$;
3: **Output:** $\sigma^{tr}, \nabla\theta$;
4: **if** Layer(tr)=convolutional lay **then**
5:     $\sigma^{tr} = (\mathcal{W}^1)^T \times \sigma^{tr+1} \cdot f'(z^{tr+1})$;
6:     $\nabla\mathcal{W}^1 = \sum \sigma^{tr+1} \otimes (f(z^{tr}))^T$;
7:     $(\nabla T_{t\in\mathcal{I}(T_\mathcal{I})}, \nabla M_{t\in\mathcal{L}(DT_\mathcal{I})}) = HT\_BP(\nabla\mathcal{W}^1)$;
8:     $\nabla\theta = \left\{\nabla b, \nabla T_{t\in\mathcal{I}(T_\mathcal{I})}, \nabla M_{t\in\mathcal{L}(T_\mathcal{I})}\right\}$, with$\lambda b = \sigma^{tr+1}$;
9:     **if** Layer(tr)=pooling layer **then**
10:        $\sigma^{tr} = (\mathcal{W}^2 \times \sigma^{tr+1} \cdot f'(z^{tr+1})$;
11:        $\nabla\mathcal{W}^2 = \sigma^{tr+1} \otimes$ pooling$(f(z^{tr}))$;
12:        $\nabla b = \sigma^{tr+1}$
13:        **if** Layer(tr)=fully layer-connected layer **then**
14:            $\sigma^{tr} = (\mathcal{W}^3)^T \odot \sigma^{tr+1} \cdot f'(z^{tr+1})$;
15:            $\nabla\mathcal{W}^3 = \sigma^{tr+1} \odot (f(z^{tr}))^T$;
16:            $(\nabla T_{t\in\mathcal{I}(T_\mathcal{I})}, \nabla M_{t\in\mathcal{L}(DT_\mathcal{I})}) = HT\_BP(\nabla\mathcal{W}^3)$;
17:            $\nabla\theta = \left\{\nabla b, \nabla T_{t\in\mathcal{I}(DT_\mathcal{I})}, \nabla M_{t\in\mathcal{L}(DT_\mathcal{I})}\right\}$, with$\lambda b = \sigma^{tr+1}$;
18:        **end if**
19:    **end if**
20: **end if**
```

maximum hierarchicak rank of the tensor $\mathcal{A}$. Next, by HTF, the tensor will be decomposed into the D-factor matrix $M_t \in R^{I_t \times r_t}$ and the $D-1$ third-order tensor $\mathcal{T}_t \in R^{r_t \times r_{t_l} \times r_{t_r}}$, resulting in the storage complexity of $O(DIR)$ and $O((D-1)R^3)$, respectively. Therefore, the magnitude of the corresponding weight tensor is reduced to $O(DIR + (D-1)R^3)$ after decomposition. Since the initial size of the weight tensor $\mathcal{A}$ is $O(I^D)$, the compression ratio becomes $O(\frac{I^D}{DIR+(D-1)R^3})$ with HTF. The compression rate using HTN is about $O(I/R)$ times that of TTN compared to TTN with data size $O(DIR^2)$. Therefore, the proposed algorithm can reduce bandwidth consumption and storage requirements. Next, analyze the energy consumption of the model, which consists of the communication and computing energy of each edge node. This paper adopts the communication and computing models adopted in [7], which are usually used to analyze the energy consumption of edge nodes. It is assumed that l_i bits of the parameter are transmitted from the edge node E_i to the cloud with transmitted power p_i, and that there is no

Algorithm 4 Algorithm 4: Hierarchical Tensor Back-Propagation Algorithm

```
1: Input: $tr, \theta, z^{tr}, \sigma^{tr+1}$;
2: Output: $\nabla M_{t\in\mathcal{L}(DT_{\mathcal{I}})}, \nabla T_{t\in\mathcal{I}(DT_{\mathcal{I}})}$
           $h = [log_2 d] = min\{i \in N, i \geq log_2 d\}$;
3: for layer=0: h do
4:   if layer=0 then
5:     $\nabla T_t^{tr} = \nabla\mathcal{W} \cdot (M_{t_l} \odot M_{t_r})$;
6:     $\nabla M_{t_l}^{tr} = \nabla\mathcal{W} \cdot X_{(M_{t_l} \odot M_{t_r})} T_t$;
7:     $\nabla M_{t_r}^{tr} = \nabla\mathcal{W} \cdot X_{(M_{t_l} \odot M_{t_r})} T_t$;
8:   else
9:     for $t = 1 : 2^{layer}$ do
10:      if $t \in t_l^{tr-1}$ then
11:        $\nabla T_t^{tr} = \nabla M_{t_l}^{tr-1} \cdot (M_{t_l} \odot M_{t_r})$;
12:        $\nabla M_{t_l}^{tr} = \nabla M_{t_l}^{tr-1} \cdot X_{(M_{t_l} \odot M_{t_r})} T_t$;
13:        $\nabla M_{t_r}^{tr} = \nabla M_{t_r}^{tr-1} \cdot X_{(M_{t_l} \odot M_{t_r})} T_t$;
14:      end if
15:    end for
16:    if $t \in t_r^{tr-1}$ then
17:      $\nabla T_t^{tr} = \nabla M_{t_r}^{tr-1} \cdot (M_{t_l} \odot M_{t_r})$;
18:      $\nabla M_{t_l}^{tr} = \nabla M_{t_r}^{tr-1} \cdot X_{(M_{t_l} \odot M_{t_r})} T_t$;
19:      $\nabla M_{t_r}^{tr} = \nabla M_{t_r}^{tr-1} \cdot X_{(M_{t_l} \odot M_{t_r})} T_t$;
20:    end if
21:  end if
22: end for
```

path loss at each model update, Therefore, the transmission rate obtained is given as follows[22]:

$$r_i = Blog_2(1 + \frac{p_i c_i}{\vartheta^2}) \tag{10}$$

where B is the bandwidth,$c_i \sim exp(1)$ is the channel power gain and ϑ^2 denotes the additive white Gaussian nosie power. It is assumed that c_i is constant during transmission of l_i bit parameters. Therefore, the duration T_i of the parameters transmitted to l_i bits between the edge node E_i and the cloud is:

$$T_i = l_i / r_i \tag{11}$$

The corresponding communication energy of l_i bit transmitted on edge node E_i during duration T_i is:

$$E_i^{com} = p_i T_i = (p_i l_i)/r_i \tag{12}$$

It is still assumed that $\mathcal{A} \in \mathcal{R}^{I_1 \times I_2 \times \ldots \times I_D}$ is the weight tensor, with b bits for each weight. Thus, the communication energy related to the dimension I, order D and rank R of $\mathcal{A}$ can be obtained as follows:

$$E_i^{com} = \frac{O(p_i bDIR + p_i b(D-1)R^3)}{r_i} \tag{13}$$

It can be concluded that the larger the rank R is, the larger the size of the model is, and the more communication energy is consumed.

For the convenience of analysis, this paper considers that the computation energy consumption of a local iteration of edge node is E_i. Edge node is set to compute the effective capacitance chipset coefficient beta λ_i [23] , time block set to T_i, for all the sample training $\{(x_i^n, y_i^n)\}$, the size of D_i (bit). In addition, set the CPU frequency to f_i and the number of one-cpu cycles for executing the sample to l_i. Therefore, the number of CPU cycles to run a local iteration on the edge node E_i is $l_i|D_i|$.It can be concluded that the E_i energy consumption of edge nodes required to compute a local iteration is as follows:

$$E_i^{comp} = \sum_{i=1}^{l_i|D_i|} \lambda_i f_i^2 = \lambda_i l_i |D_i| f_i^2 = \lambda_i T_i f_i^3 \tag{14}$$

where $\sum_{i=1}^{l_i|D_i|} \frac{1}{f_i} = T_i$.

The following sections will evaluate the energy consumption of the proposed algorithm based on the above analysis.

6. Experimental Settings and performance

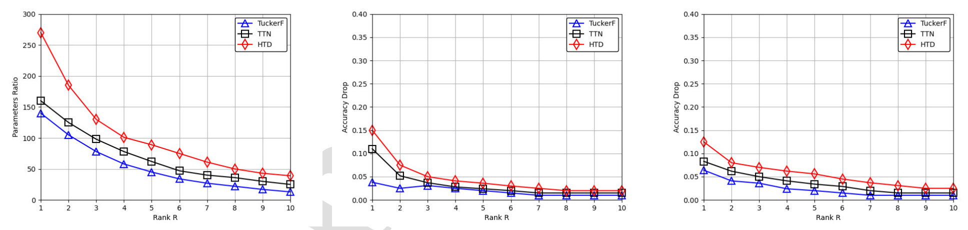
6.1. Experimental environment Setting

In terms of performance evaluation, this paper adopts the classical data set STL-10 for testing [24], which contains 15000 labeled images of 10 categories. The whole data set is divided into a training set containing 5000 images of 10 categories and a test set containing 10,000 images of 10 categories. During all the simulations, each training image was adjusted to 64×64 pixels with RGB channels represented by third-order tensors.

In this paper, two different cases of data distribution on edge nodes are taken as reference. In case I, the data samples are first divided into ten categories, and each edge node occupies the entire category, which represents independent isomorphic distribution (IID). In case II, data samples of the same category are assigned to each node, which represents a non-IID distribution. These two cases are IID=1 and IID=0 respectively. When using HTF for model compression, the maximum level of the weight tensor ranges from 1 to 10. Adam was selected as the optimizer, and the first-order moment estimation was $\beta_1 = 0.9$, and the second-order moment estimation was $\beta_2 = 0.999$. By default, IID=1, $\eta_1 = 0.01$, while IID=0, $\eta_2 = 0.001$.All simulation experiments were conducted on tensorFlow framework computers with Intel CPU I7-9700, 16GB memory and a single NVIDIA Tesla V100 GPU, which used synchronization to simulate multiple edge node scenarios.

6.2. Performance analysis

Performance requirements in terms of reduced accuracy, parameter compression ratio (PCR), and power consumption were first assessed. The lower accuracy refers to the lower classification accuracy of the proposed model due to parameter compression compared with the original model. The smaller the drop, the better the performance of the method. PCR represents the compression ratio of model parameters before and after compression. For comparative performance fairness, the proposed model does not use any additional model compression techniques such as parameter pruning, sharing, and knowledge distillation. CNN based on tensors is used to construct the proposed model. The proposed model contains three tensor convolution blocks, each block is composed of two tensor convolution layers, a pooling layer and two fully connected layers. In particular, there are 128 convolution kernels in the first three convolution layers, and 256 convolution kernels in the last three convolution layers, whose size is 3×3. As for the fully connected layer, it has 512 and 20 output units respectively. Fully connected networks also employ a dropout strategy to prevent the neuron units from being set to a probability of dropout of 52% to avoid over-fitting. ϵ is defined as the number of local update steps at the edge nodes between two global aggregates, and the number of communication rounds represents an aggregation performed by its system. In this paper, T is also set as the total number of iterations carried out by each edge node, and α is set as the percentage of all edge nodes participating in each round of model update. 1) Performance comparison algorithm with different tensor decomposition: The accuracy decline and performance comparison of PCR with different rank R were evaluated. **Fig.3** shows the change from 1 to 10 of the accuracy decline of different tensor decomposition algorithms compared with R. In this simulation, the parameters $\alpha = 0.8$ and $\epsilon = 25$ were selected. It is obvious from **Fig. 3(a)** that PCR with HT is superior to the other two decomposition algorithms in all rank R. However, it can also be seen from **Fig. 3 (b)** and (c) that this improvement comes at the cost of accuracy. For example, when rank R is 6, the accuracy decline of the above algorithm is similar: IID=1, when rank R is not greater than 0.01, IID=0, when rank R is not greater than 0.03; The corresponding PCR was about 60 for HTD, 40 for TTN and 35 for TuckerF.



(a) (b) (c)

Figure 3: Performance of different tensor decomposition algorithms: HTD, TTN, Tucker decomposition. (a) PCR VS R. (b) Accuracy VS R in IID=1. (c) Accuracy VS R in IID = 0.

2) Performance evaluation with noise properties: It is assumed that when the model communicates between the edge node and the server, its model weight is affected by gaussian noise pollution. **Fig. 4** shows the comparison of the accuracy decline of the proposed algorithms for two different data distributions with different SNR. It can be seen from **Fig. 4** that the accuracy

decline of the two data distributions decreases with the increase of SNR.

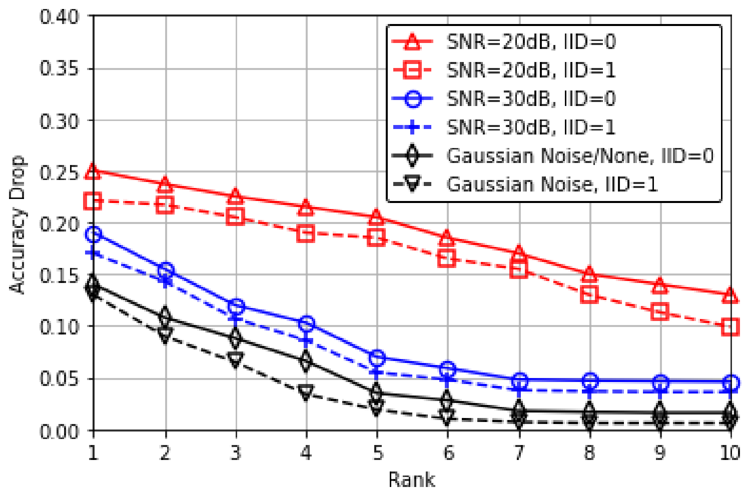


Figure 4: Comparison of performance with (without) noise

3) Comparison of influence of polymerization frequency performance: Evaluate performance comparisons of different aggregation frequencies with different data distributions. **Fig. 5** shows the decreasing accuracy of the proposed model when the polymerization frequency varies from 1 to 150, where $M = 80$ and $\alpha = 0.8$ and the rank is $R = 10$ and $R = 5$, respectively. It can be seen that at IID=1, the decrease in accuracy decreases as the aggregation frequency ϵ decreases until it tends to level. This indicates that high frequency model aggregation is not required when data distribution is balanced. This is because edge nodes can learn the features of the whole data set by themselves, rather than communicating frequently with other edge nodes (in this case, the data set is evenly and randomly distributed among each edge node). Compared with the case where IID $= 0$, the accuracy decreased first and then increased with the aggregation frequency. This indicates that there exists an optimal value of aggregation frequency in the case of deviation and imbalance of data distribution. This also means that in the case of IID $= 0$, edge nodes can effectively learn the features of other data sets only by communicating with the cloud.

4) Comparison of energy consumption and performance of grade R: Consider here that 50 edge nodes will participate in the federated learning process. This paper evaluates the energy consumption performance of each

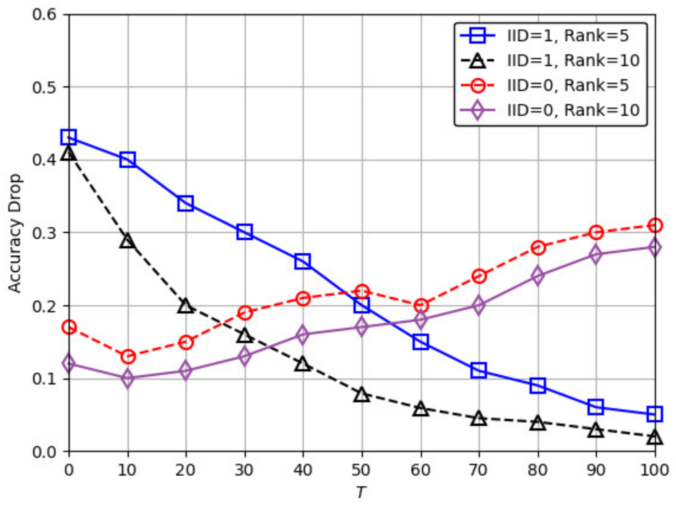


Figure 5: Comparison of aggregation frequency and accuracy under different conditions.

edge node for parameter communication and local computation. The following environment configuration is used in the simulation: $B = 3$MHz, $\vartheta^2 = -80$dBm, $P_i = 45$ dBm, $l_i = 10^{-3.20}$, $f_i = 2.5$ GHz, $\lambda_i = 10^{-29}$. As can be seen from **Fig.6** and **Fig.7**, the higher rank R is, the more communication energy and computing energy are consumed. This is mainly because the size of model parameters increases with the increase of rank R. It is obvious that the energy consumption of communication is much greater than that of calculation. Because of its higher compression efficiency, the proposed model can achieve lower communication energy consumption than other tensor decomposition algorithms. It also shows that the computational energy consumption of the proposed algorithm is similar to that of other methods, which indicates that the proposed algorithm can reduce energy consumption without introducing additional computational complexity.

7. Conclusion

In this paper, a distributed deep computing model with HTD for federated learning is proposed. For two different data distributions, the proposed algorithm can effectively compress model parameters by HTF algorithm at each model update. Specially, the decline in accuracy for IID distributions is

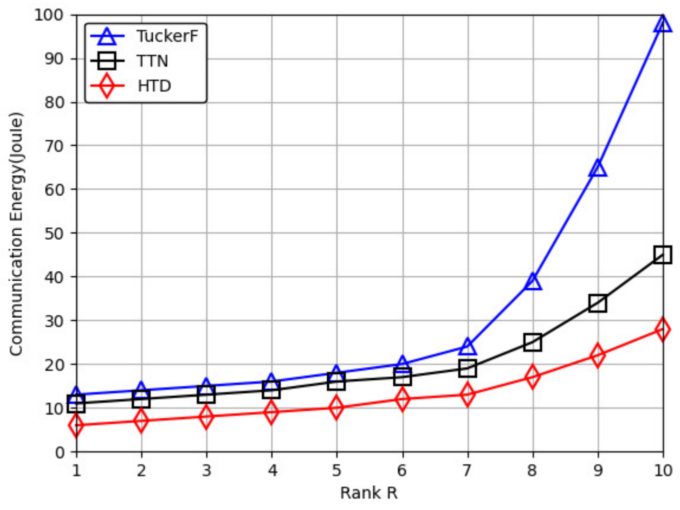


Figure 6: Comparison of communication energy consumption and rank R under different tensor decomposition algorithms.

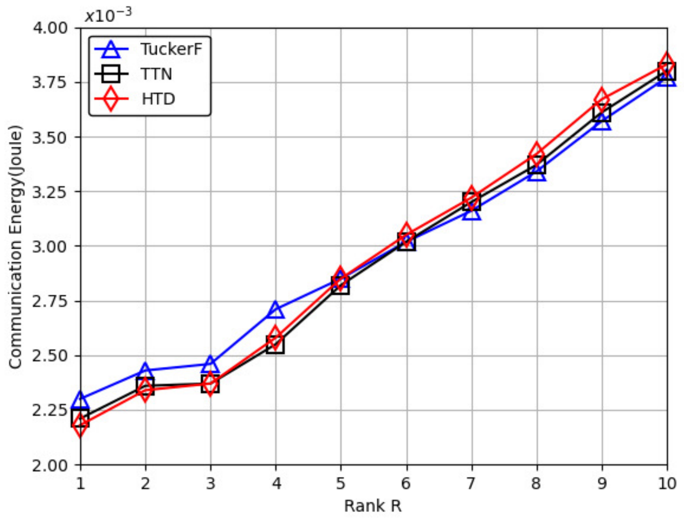


Figure 7: Comparison of calculated energy consumption and rank R under different tensor decomposition algorithms.

usually lower than for non-IID distributions in the same PCR case. Moreover, compared with the compression and decompression method, it is found that the proposed algorithm not only achieves small precision reduction in most PCR cases, but also improves the training efficiency. In addition, it is shown that compared with other tensor decomposition algorithms, the proposed algorithm can save more energy, especially in communication energy. The effect of aggregation frequency on different data distribution performance of edge nodes is also studied. Finding the appropriate frequency of aggregation is important for different data distributions.

Although the proposed algorithm has some innovative advantages, there are still many challenges in practical application. First, federated learning relies on the server for local model aggregation, which makes it difficult to expand. To solve this problem, a fully distributed learning method based on consensus algorithm can be adopted[7]. In addition, federated learning has some limitations such as slow and unstable communication, and different resources in heterogeneous devices. For example, when heterogeneous devices participate in training, they may have different resources and involve different computing capabilities, bandwidth limits, and so on. At the same time, unstable wireless communication channels can cause problems with frequent disconnections to servers.These limitations require serious consideration in practice, which can be addressed by system-level design and some promising communication techniques, which we will also consider in future work.

References

[1] C. C. Lin, J. W. Yang, Cost-efficient deployment of fog computing systems at logistics centers in industry 4.0, IEEE Transactions on Industrial Informatics (2018) 1–1.

[2] B. Zhao, X. Li, X. Lu, Tth-rnn: Tensor-train hierarchical recurrent neural network for video summarization, IEEE Transactions on Industrial Electronics PP (2020) 1–1.

[3] Q. Zhang, L. T. Yang, Z. Chen, P. Li, Incremental deep computation model for wireless big data feature learning, IEEE Transactions on Big Data (2020) 1–1.

[4] Z. Li, H. Hu, H. Hu, B. Huang, V. Chang, Security and energy-aware

collaborative task offloading in d2d communication, Future Generation Computer Systems 118 (2021).

[5] A. Alwarafy, K. A. Al-Thelaya, M. Abdallah, J. Schneider, M. Hamdi, A survey on security and privacy issues in edge computing-assisted internet of things, IEEE Internet of Things Journal (2020).

[6] P. Zhou, Q. Lin, D. Loghin, B. C. Ooi, H. Yu, Communication-efficient decentralized machine learning over heterogeneous networks (2020).

[7] L. Kong, X. Y. Liu, H. Sheng, P. Zeng, G. Chen, Federated tensor mining for secure industrial internet of things, IEEE Transactions on Industrial Informatics PP (2019) 1–1.

[8] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, K. Chan, Adaptive federated learning in resource constrained edge computing systems, IEEE Journal on Selected Areas in Communications (2019) 1–1.

[9] L. Huang, Y. Yin, Z. Fu, S. Zhang, D. Liu, Loadaboost: Loss-based adaboost federated machine learning with reduced computational complexity on iid and non-iid intensive care data, PLoS ONE 15 (2020) e0230706.

[10] G. K. Gudur, S. K. Perepu, Zero-shot federated learning with new classes for audio classification (2021).

[11] Y. Lin, S. Han, H. Mao, Y. Wang, W. J. Dally, Deep gradient compression: Reducing the communication bandwidth for distributed training (2017).

[12] A. F. Aji, K. Heafield, Sparse communication for distributed gradient descent, in: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, 2017.

[13] J. Bernstein, Y. X. Wang, K. Azizzadenesheli, A. Anandkumar, signsgd: compressed optimisation for non-convex problems, International Conference on Machine Learning (2018).

[14] S. Samarakoon, M. Bennis, W. Saad, M. Debbah, Federated learning for ultra-reliable low-latency v2v communications, arXiv e-prints (2018).

[15] C. Chen, H. Xiang, T. Qiu, W. Cong, Z. Yang, V. Chang, A rear-end collision prediction scheme based on deep learning in the internet of vehicles, Journal of Parallel and Distributed Computing 117 (2017) 192–204.

[16] Y. Wei, N. C. Luong, D. T. Hoang, Y. Jiao, C. Miao, Federated learning in mobile edge networks: A comprehensive survey, IEEE Communications Surveys Tutorials PP (2020) 1–1.

[17] F. Sattler, S. Wiedemann, K. Müller, W. Samek, Sparse binary compression: Towards distributed deep learning with minimal communication, in: 2019 International Joint Conference on Neural Networks (IJCNN), 2019.

[18] An incremental deep convolutional computation model for feature learning on industrial big data, IEEE Transactions on Industrial Informatics 15 (2019) 1341–1349.

[19] A. Aa, A. Yz, B. Mz, Exploiting dynamic spatio-temporal correlations for citywide traffic flow prediction using attention based neural networks, Information Sciences 577 (2021) 852–870.

[20] D. Kressner, C. Tobler, Algorithm 941: htucker—a matlab toolbox for tensors in hierarchical tucker format, Acm Transactions on Mathematical Software 40 (2014) 1–22.

[21] Grasedyck, Lars, Hierarchical singular value decomposition of tensors, SIAM Journal on Matrix Analysis and Applications 31 (2010) 2029–2054.

[22] X. Cao, F. Wang, J. Xu, R. Zhang, S. Cui, Joint computation and communication cooperation for energy-efficient mobile edge computing, IEEE Internet of Things Journal (2018).

[23] T. D. Burd, R. W. Brodersen, Processor design for portable systems, Journal of Vlsi Signal Processing Systems for Signal Image & Video Technology 13 (1996) 203–221.

[24] A. Coates, A. Ng, H. Lee, An analysis of single-layer networks in unsupervised feature learning, in: Proceedings of the fourteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, 2011, pp. 215–223.

Author Statement

Author 1 (First Author):

Conceptualization, Methodology, Software, Investigation, Formal Analysis, Writing - Original Draft;

Author 2 (Corresponding Author):

Data Curation, Supervision, Resources, Writing-Review & Editing;

Author 3:

Visualization, Investigation, Formal Analysis;

Author 4:

Resources, Supervision;

Author 5:

Software, Validation

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: