

Developing normalization schemes for data isolated distributed deep learning

Yujue Zhou^{1,2}  | Ligang He¹ | Shuang-Hua Yang²

¹Department of Computer Science, University of Warwick, Coventry, UK

²Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China

Correspondence

Shuang-Hua Yang, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China.
Email: yangsh@sustech.edu.cn

Funding information

National Natural Science Foundation of China, Grant/Award Numbers: 61873119, 61911530247; Science, Technology and Innovation Commission of Shenzhen Municipality, Grant/Award Number: KQJSCX20180322151418232

Abstract

Distributed deep learning is an important and indispensable direction in the field of deep learning research. Earlier research has proposed many algorithms or techniques on accelerating distributed neural network training. This study discusses a new distributed training scenario, namely data isolated distributed deep learning. Specifically, each node has its own local data and cannot be shared for some reasons. However, in order to ensure the generalization of the model, the goal is to train a global model that required learning all the data, not just based on data from a local node. At this time, distributed training with data isolation is needed. An obvious challenge for distributed deep learning in this scenario is that the distribution of training data used by each node could be highly imbalanced because of data isolation. This brings difficulty to the normalization process in neural network training, because the traditional batch normalization (BN) method will fail under this kind of data imbalanced scenario. At this time, distributed training with data isolation is needed. Aiming at such data isolation scenarios, this study proposes a comprehensive data isolation deep learning scheme. Specifically, synchronous stochastic gradient descent algorithm is used for data exchange during training, and provides several normalization approaches to the problem of BN failure caused by data imbalance. Experimental results show the efficiency and accuracy of the proposed data isolated distributed deep learning scheme.

1 | INTRODUCTION

Distributed deep learning has become a popular research and industrial topic in recent decades [1]. The main purpose of previous research on using distributed methods to train neural networks is the pursuit of higher training efficiency, or simply to deal with the problem that the excessive data cannot be put into a machine for training [2]. The distributed learning has many advantages, but, in some cases, the existing methods do not work. Because in traditional distributed learning, the central server can obtain all the training data, the data is evenly shuffled and distributed to each distributed computing nodes during the training process. However, in some scenarios, each node can only have its own data, and the data between the nodes cannot be shared or uploaded to the central node. This is the new scenario studied here. Distributed neural network training is performed in the case of data isolation, that is, data isolated distributed deep learning.

The data isolated distributed scenario is not a completely new concept. It is also involved in federated learning. Federated learning is essentially a distributed learning framework. Although the scenario considered in federated learning is also a situation where data cannot be shared between computing nodes, the scenario considered in federated learning is different from the data isolation scenario discussed in the following aspects. The first is from the scene. Federated learning is originally proposed by Google [3], which is a federated learning system for mobile devices. This starting point determines that the computing nodes in the federated learning system have the characteristics of a large number, limited computing power and unstable communication. This is the main difference between federated learning and distributed learning, and the most fundamental difference between the scenarios it considers and the data isolation scenario we consider. In the data isolation scenario discussed here, each computing node has certain computing capabilities, and the communication between nodes

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2021 The Authors. *IET Cyber-Physical Systems: Theory & Applications* published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.

is relatively stable. The second is from the objective. Because the scenario considered by federated learning is a situation where the nodes are large and unstable, the goal is to solve the node selection strategy and coordinate the imbalance of computing resources among the nodes. The research objective of the distributed data isolation deep learning discussed here is whether the data imbalance between the nodes will affect the traditional distributed training architecture, and to solve the negative impact of the imbalanced data distribution on the normalization operation of the neural network. The third is from the focus. Federated learning focuses more on how to encrypt and decrypt the information that needs to be interacted to protect privacy, and how to coordinate and schedule the data imbalance and computing resource imbalance of each computing node. The focus of our work is to discuss the impact of data imbalance in distributed computing on the normalization method, and how to use the normalization scheme to ensure the accuracy of distributed training neural networks.

The scene of data isolation is very common in the real-life situation. The data isolation may first aim at the protection of privacy. For example, user data cannot be uploaded to the central server for training, but the central server needs to train a model based on the data of each user. This scenario can refer to the data of the hospital [4]. Each hospital has its own patient data. For the purpose of privacy protection, data among various hospitals cannot be shared. However, the data of each hospital may have its own distribution. For example, one hospital that specializes in lung cancer may have more data of lung cancer patients, and another hospital that specializes in treating oral cancer may have more data of oral cancer patients. In order to ensure the generalization of the model, this model is required to learn enough data, not just based on data from a local hospital. This is a neural network training process with data isolated situation. Another consideration for the formation of data isolation may be the limit of the communication capacity of data transmission. When the data is widely distributed, all local data need to be uploaded to the central server for unified training, which requires huge communication overhead and memory. In this case, the use of data isolated network training framework is also a better choice.

Therefore, based on such data isolation scenarios, we aim to propose a neural network training framework. At the same time, because of the data isolation situation, the data between various data sources can be imbalanced, which poses a challenge to the normalization process in the neural network training. Here four solutions are provided to solve the normalization problem during neural network training in data isolation scenarios.

In summary, our contributions lie mainly in the following four aspects:

- We have proposed a neural network training framework for data isolated scenario.
- We have theoretically proved batch normalization (BN) is not suitable for imbalanced data distribution in data isolation distributed learning.

- Due to data imbalance in data isolation scenarios, we have discussed the feasibility of four existing normalization approaches, instance normalization (IN), layer normalization (LN), channel normalization (CN) and group normalization (GN), which were designed for other different purposes. These four methods are not affected by the imbalanced distribution of data, and can be used in our data isolated deep learning.
- We conclude that GN outperforms among these four approaches for the data isolated deep learning, but needs to manually choose the parameter group number.

The following part of this study is structured as follows. The previous work about distributed deep learning is introduced in Section 2. In Section 3, the overall framework of neural network training based on data isolation scenarios is given. In Section 4, we discuss the reason why BN algorithm is not applicable in data isolated distributed training. In Section 5, we present four approaches to the data imbalance normalization problem caused by the data isolation scenario. In Section 6, the experimental part shows the effect of the four normalization approaches. Finally, the conclusion and future work are discussed in Section 7.

2 | RELATED WORK

Deep neural network (DNN) has become an important direction in the field of machine learning research, and also an important tool in many industrial and daily life applications. However, data that need to be processed by DNN is getting larger and larger. In order to train DNN more effectively, distributed/parallel deep learning research has attracted more interests. According to [2], the reason for the parallel neural network is that the neural network model contains millions of parameters and thus, it takes large amount of data to learn these parameters. This is a very time-consuming and computationally intensive process, while the distributed neural network can effectively improve the training efficiency of DNN.

According to [5], the previous research on distributed deep learning can be mainly divided into three categories: data parallelism, model parallelism and hybrid parallelism. Data parallelism is splitting the complete data set to different machines, and each machine uses the allocated data for calculation in each training iteration. There are two main methods of data parallelism, one is synchronous update [6], and the other is asynchronous update [7]. The difference between them is whether to wait for all nodes to complete the calculation when updating the weight. If it needs waiting, it is called as synchronous stochastic gradient descent (SSGD), and if it does not need to wait, it is called as asynchronous stochastic gradient descent (ASGD). Furthermore, in order to solve the delayed gradient problem in ASGD, more research has been carried out [8–10]. Among them, the authors of [10] proposed a delay compensation ASGD method to compensate when the weights are updated, so that the training speed is improved

compared with ordinary ASGD. The method of model parallelism is also called as network parallelism. It splits the neural network into different machines. After all machines get the same batch of data, each machine calculates different parts of the DNN. Previous studies on model parallelism include [11–13]. Hybrid parallelism is the combined application of data parallelism and model parallelism, and also has some successful studies [14–17]. However the focus of this work is how to accelerate distributed training, without considering the data isolation in distributed scenarios.

The generation of the data isolation scenario is mainly due to privacy issues, that is, each node has its own local data and does not allow to exchange data with any other nodes. The purpose of data isolation distributed learning is to train a global model only with the data of each node. Previous research on privacy-preserving machine learning models includes [18–20]. Among them, [18, 19] discussed the training of privacy-preserving distributed regression models. And [20] proposed a new protocol that encrypts the data to be exchanged for privacy protection purposes. This method can be used for linear regression, logistic regression and neural network training. However, the focus of these works is how to encrypt data to protect data privacy. In essence, they all study the exchange of encrypted data or the encrypted intermediate results. They hardly considered distributed training with the deep learning model, and did not consider the problem of data imbalance caused by data isolation.

The previous research about data isolated deep learning has been discussed in a few studies from the aspect of federated learning. For example, [21] discussed secure multiparty computation, differential privacy and homomorphic encryption are three privacy protection mechanisms for federated learning. It focused on the use of some strategies or encryption methods to protect privacy, and did not consider the problem of data imbalance in data isolation scenarios. The author in [22] established a self-balanced federated learning framework called *Astraea*, which was used to mitigate the impact of data imbalance on the accuracy of federated learning. However, it needed to know the distribution of data at each node and the distribution of all data, which may not be available in the scenario where data is isolated. Similar to [22, 23] studied collaborative fairness in federated learning and proposed a collaborative fair federated learning framework. This framework uses reputation to force participants to converge to different models, thereby achieving fairness. The research on the imbalanced data distribution caused by data isolation in federated learning mainly comes from [24, 25]. Among them, [24] studied the method of data expansion that can be used for boundary-aware fusion and boundary expanding to solve the problem of data skew. And [25] proposed a monitoring scheme that can infer the composition of each round of federated learning training data, and design a new loss function to alleviate the impact of imbalance. However, none of them discussed the issue of BN failure caused by data imbalance. At the same time, [26] researched the dynamic modeling of cross-layer soft error rate based on back propagation neural network, and proposed a system availability optimization strategy.

All the studies mentioned above have not paid attention to how to achieve normalization in the case of data isolation. Normalization is very important to guarantee the accuracy of neural network. We discuss the problem of imbalanced distribution of data among various nodes caused by data isolation scenarios. Such imbalanced data will lead to the failure of BN in distributed neural network training. And we will give the solution to this problem, thus giving a new data isolation deep learning scheme.

3 | FRAMEWORK OF DATA ISOLATED DISTRIBUTED DEEP LEARNING

The framework of distributed neural network is based on the SSGD algorithm. The essence of data isolated deep learning is a distributed neural network training process. In order to understand the proposed framework better, the structure of the neural network and SSGD algorithm will be introduced in detail in the following section.

3.1 | Neural network structure

The complete neural network structure includes an input layer, one or more hidden layers and an output layer. The input data goes through the layer-by-layer connected network and gets the final output result. In the simplest DNN structure, the connection between each upper and lower layer will generally pass through the fully connected layer, the normalization layer and the activation layer, as shown in Figure 1. In a more complex network with CNN structure or RNN structure, the fully connected layer is generally replaced by the convolutional layer or an RNN connection layer, though it is essentially a combination of the linear connection layer and the activation layer. For the sake of simplicity, the following discussion uses a simple DNN structure as an example, but the derivation from the DNN structure to the CNN or RNN structure can be easily achieved.

As mentioned above, the transfer between the two layers of neurons in the neural network generally goes through three processes: the linear connection layer, the normalization layer and the activation layer. In the forward propagation of the neural network, the formulas of these three layers are as follows.

The linear connection layer is described as:

$$X' = W \cdot X + B = \sum_i (w_i \cdot x + b_i), \quad (1)$$

where X is the input of the linear connection layer, X' is the output of the linear connection layer and W and B are the weights and biases of the linear connection layer.

The normalization layer is described as:

$$X'' = \text{NORM}(X'), \quad (2)$$

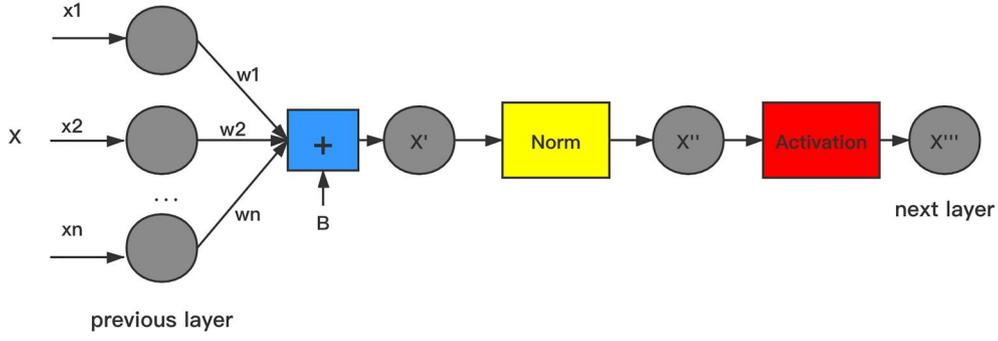


FIGURE 1 Neural network with normalization layer

where X' is the input of the normalization layer, X'' is the output of the normalization and $NORM()$ represents various normalization methods.

The activation layer is presented as:

$$X''' = \sigma(X''), \quad (3)$$

where X'' is the input of the activation layer, X''' is the output of the activation and $\sigma()$ represents various activation functions. The most common activation functions are Relu, Sigmoid, Tanh etc.

In the backward propagation of the training, the parameters which need to be updated are in the linear connection layer and the normalization layer, while the activation layer does not have learnable parameters. The normalization layer will be discussed in detail in the next two sections. The parameter update formula given below is only for the parameters in the linear connection layer.

$$\theta := \theta - \eta \frac{\partial J}{\partial \theta}, \quad (4)$$

Such a parameter update formula is the stochastic gradient descent algorithm [27], where θ denotes the parameters that need to be learnt and updated, and specifically includes the linearly connected weights (W) and the bias (B) referred as weights. Here J is the cost function, generally the most common ones are the quadratic cost function and cross-entropy cost. η denotes the learning rate, which is an adjustable parameter in the neural network optimizer. The most important part in the formula is $\frac{\partial J}{\partial \theta}$, which is called gradient (G). The transmission and integration of the gradients is the core of distributed deep learning, which will be specifically explained in the next section.

3.2 | Synchronous stochastic gradient descent

SSGD [28] is the most common method for distributed neural network training. In the structure of SSGD, there are $n + 1$ processes, one of which is called master, and the other

processes are called workers. The master's job is to receive the information sent from each worker, and then broadcasts it after processing. The worker's job is to use its own local data for local calculations. The algorithm of SSGD is shown in Algorithm 1.

Algorithm 1: SSGD

```

Workeri collects its own data ( i=1,2,...,n, n is number of
workers); Master and Workers initialize the network;
for each epoch do
  Workers load data into batches;
  for each iteration do
    1) Master broadcast latest weights( $W_t$ ) to all
    Workers;
    2) Workers do Forward Propagation:
       calculate loss( $L_i$ );
    3) Workers do Backward Propagation:
       calculate gradient( $G_i$ );
    4) Workers send its own gradient( $G_i$ ) to Master;
    5) Master calculates average gradient:
        $G_{ave} = \frac{1}{n} \sum_{i=0}^n G_i$ 
    6) Master updates weights using average gradient:
        $W_{t+1} = W_t - \eta G_{ave}$ 
  end
end

```

First, each worker organizes its local data to prepare for training. Then the master and workers initialize the network structure. When training, in each epoch, the worker first loads its own local data divided into batches. Then in each batch of training, the master first sends its latest weights to all workers. All workers replace the weights of the local models with the received weights. Next, each worker starts the local calculation, first doing the forward propagation to calculate the loss, and then using the loss to perform the backward propagation to calculate the gradients. After completing the calculation, the worker sends the local gradients to the master and then waits. When the master receives the gradients from all workers, it calculates the average of all gradients and uses the average gradients to update the weights. This process repeats until the model training reaches the end condition.

Next, we provide a simple proof that the distributed SSGD algorithm is essentially the same as all data placed on a central server for training. Without considering BN, SSGD is not affected by the imbalance distribution of data among workers.

For each weight needed to be updated, the average gradient G_{ave} calculated by the master is the average of n workers gradients $\{G_i\}_{i=1,\dots,n}$, G_i is the gradient calculated by the i_{th} worker, one has:

$$G_{ave} = \frac{1}{n} \sum_{i=1}^n G_i, \quad (5)$$

where G_i is the gradient average of all samples in this batch of the i_{th} worker. Assuming that the batch size is m_i , g_{ij} represents the gradient obtained by the j_{th} sample of the i_{th} worker, then G_i can be expressed as:

$$G_i = \frac{1}{m_i} \sum_{j=1}^{m_i} g_{ij}. \quad (6)$$

Then, we can get that:

$$G_{ave} = \frac{1}{n} \sum_{i=1}^n \frac{1}{m_i} \sum_{j=1}^{m_i} g_{ij}. \quad (7)$$

When the batch size of all workers is the same, that is, $m_1 = m_2 = \dots = m_i = \dots = m$, we can show that:

$$G_{ave} = \frac{1}{nm} \sum_{i=1}^n \sum_{j=1}^m g_{ij}. \quad (8)$$

It can be seen that G_{ave} obtained now is actually equivalent to the gradient training with the batch size of nm using a single process. At the same time, this nm -size batch is the integration of the data from all workers. It shows that even if there is data imbalance in a single worker, this batch can ensure that the data is derived from the average sampling of the entire data. When the batch sizes of the workers are inconsistent, G_{ave} can also be calculated by using the weighted average method, which will not affect the conclusion and not be discussed in detail here.

3.3 | Scheme of data isolated deep learning

After the introduction of SSGD, the scheme of data isolated deep learning can be given. Each of the data sources has its own data. When we need to train a global model but cannot collect data from various data sources, we first set up a master node to receive and send the weights and gradients during the training process. Then each local node uses its own local data in the local training, and this process uses the SSGD algorithm to integrate and update the model, and

finally gets the global model. This scheme is shown in Figure 2.

However, in this scheme, the BN layer in the neural network is missing, because the training of the BN layer cannot be simply achieved by the SSGD algorithm. Therefore, we need to solve the problem of normalization for the neural network training for data isolation scenario.

4 | BATCH NORMALIZATION

BN [29] was proposed to solve a problem called covariate shift. In the simple term, the problem is that the input distribution of the activation function changes during the training process, which is prone to the problem of gradient disappearance. Once the gradient disappearance occurs the learning process cannot go any further. Adding a normalization layer to a neural network can effectively improve the training efficiency of the model. Therefore, when training the neural network with data isolated case, we also hope that the network structure can contain the normalization layer.

4.1 | Local batch normalization

In distributed neural network training, if BN is implemented directly, we call it local BN, which means that BN is performed inside each local node. During the local BN training process, similar to the training of the linear connection layer, except that the weights and gradients need to be interacted, other data does not need to be exchanged.

Algorithm 2: Local Batch Normalization

mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β ;

Training:

$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$ //mini-batch mean

$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$ //mini-batch variance

$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$ //normalize

$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)$ //scale and shift

Inference:

$E[x] \leftarrow E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ //global mean

$\text{Var}[x] \leftarrow \frac{m}{m-1} E_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$ //global variance

$y = \frac{\gamma}{\sqrt{\text{Var}[x] + \epsilon}} \cdot x + \left(\beta - \frac{\gamma \cdot E[x]}{\sqrt{\text{Var}[x] + \epsilon}} \right)$

For the algorithm of the local BN, we can refer to Algorithm 2. In the training process, local BN layer goes through four steps: (1) find the mean within this batch, (2) find the variance within this batch, (3) normalize the input and (4) finally scale and shift the input. In the inference/evaluation process, the steps are similar: use the mean and variance of each batch to update the global mean and variance, and then

normalize, scale and shift the input. Similar to distributed linear connection layer training, after each iteration of local BN computing, the gradients of weights are sent by the worker nodes to the master node. Then, the master node calculates the average gradients to update the weights. The weights that need to be updated here include scale weight γ and shift weight β .

The reasons that make the local BN inadequate for using in data isolated training are the calculations of the mean and variance. Because in the data isolation scenario, the data distribution of each data source cannot guarantee uniformity, and even the distribution may vary greatly. In this way, each node may differ greatly in calculating its own mean and variance, or it can be said that the mean and variance calculated by each node are not global. Although at the end of training, we can integrate the inference mean and variance of all nodes, this is still different from the case where the data in each batch is relatively uniform when the data is concentrated on one single server for training. Therefore, from the principle of BN, it can be seen that the local BN algorithm is not suitable for neural network training in data imbalanced distribution scenarios caused by data isolation. At the end of this study, our experimental part also gives the same conclusion.

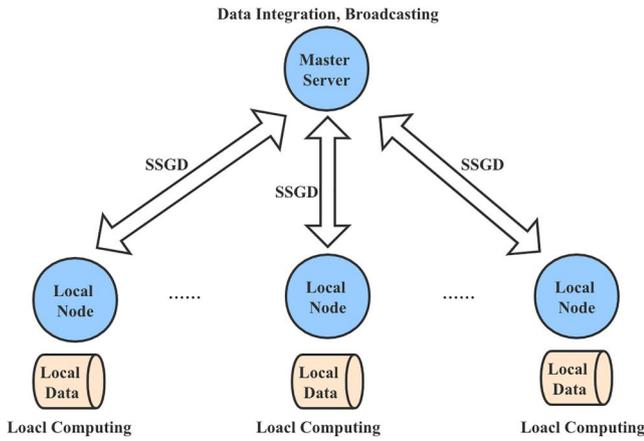


FIGURE 2 Data isolated deep learning without normalization

4.2 | Distributed batch normalization

In order to solve the normalization problem in data isolation distributed training, the most straightforward way is to use distributed BN. The study [30] proposed a model called cross-GPU BN for image processing. Because the memory of each GPU is limited, a single GPU cannot complete large batch size SGD training when the training images are too large. Therefore, this cross-GPU BN was proposed to implement neural network training across GPUs. Inspired by [30], although the purpose is different, the distributed BN in the data isolation scenario can adopt the same principle as the cross-GPU BN. That is, the results of the intermediate calculations in the BN layer are all sent to the master node to be averaged, and the average value is sent back to the local node to continue the calculation.

The above process is shown in Figure 3. The whole process is that in each training of the BN layer, each local node calculates the local batch mean and sends it to the master server. Then the master calculates the average mean and broadcasts it back to the local nodes. Each local node uses the average mean to calculate the local variance and sends it to the master. Next, the master calculates the average variance and broadcasts it back to the local nodes. Finally, the local nodes continue the calculation of the BN layer with the received global mean and variance.

Such a process can be simply proved to be equivalent to the BN layer training where the data is concentrated on a central server, and the detailed proof will not be performed here. Therefore, this distributed BN solution can be considered as an ideal realization of BN in data isolated scenario. However, this algorithm has a fatal flaw, that is, it greatly increases the communication overhead. When the BN layer is ignored and the SSGD algorithm is originally used for distributed training, data exchange is performed only after the completion of a batch training, that is, one iteration performs a round-trip data exchange. However, the distributed BN needs to perform two round-trip data exchanges during each BN layer training. If there are n -layer structures in the neural network including the

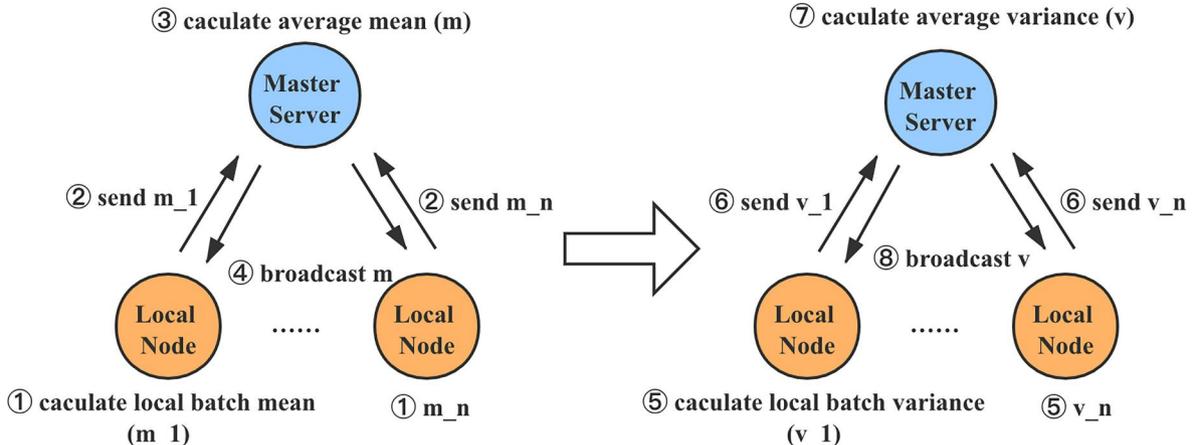


FIGURE 3 Distributed batch normalization

BN layer, the communication load will be $2n + 1$ times the original. At the same time, considering the waiting problem of the synchronization algorithm, this distributed BN algorithm will be unacceptable as it is time-consuming when the model has a large network structure. Therefore, this distributed BN algorithm can only be considered when the network structure is small, the communication speed is guaranteed in various nodes and the computing power between the nodes is relatively equally distributed.

5 | NORMALIZATION INDEPENDENT FROM DATA DISTRIBUTION

Due to the huge additional communication overhead involved in distributed BN, its availability in reality will be very low, if it is not impossible. Therefore, the following will introduce four feasible normalization approaches, which were proposed in different fields. we select them because they are independent from data distribution, and can be used for data isolation deep learning.

In addition to BN, some other normalization methods have been previously proposed for various purposes [31–34]. Among them, [31] proposed a method called IN to achieve better results in the image style transfer task. LN was proposed by [32] to compensate for the defect that the effect of BN depends on batch size. On the basis of [31, 32], a normalization method was proposed by [33] that also does not depend on batch size. It is called GN, which achieves the effect superior to IN and LN in image classification problems. In recent years, a new normalization method called CN [34] has been proposed to solve the problem of gradients vanish in deep single-channel linear convolutional networks. It is worth noting that although the purposes of these four normalization methods are different, none of them has been implemented in the batch size dimension. This shows that IN, LN, GN and CN will not be affected by data distribution, and therefore might be used for distributed training in imbalanced data distribution scenarios.

The existing normalization algorithms were proposed for image processing. Figure 4 is a schematic of five normalization algorithms when processing image data. Specifically, the dimensions of the input are batch size (N), channel (C), height (H) and width (W). Here, batch size denotes the number of samples selected in one iteration training. And height, width and channel represent the size of each input sample in three dimensions. For a 32×64 RGB picture, the values of H , W and C are 32, 64 and 3, respectively. For the input of this dimension, BN is normalized in N, H, W dimensions, IN is normalized in H, W dimension, LN is normalized in C, H, W dimensions, CN is normalized in C dimension and GN is slightly more complicated and is an algorithm between IN and LN. It divides the channel dimensions into groups and does normalization in the group, H and W dimensions.

It can be seen that except for BN, the other four normalization algorithms are not normalized in the batch size

dimension. Not normalizing in the N dimension means that normalization will only be performed within each input sample, and there is no cross-sample normalization. The result of this is that the imbalanced distribution of the data of each node will not cause the normalization to fail. Therefore, the LN, IN, CN and GN can be directly applied to neural network training under the data isolation scenario. It can be proved that their effect is the same as the effect of centralizing data to a central server for training.

Algorithm 3: Pseudocode of Normalization

Function Norm($x, \gamma, \beta, \epsilon = 1e - 5$):

```

 $N, C, H, W = x.shape$ 
 $x = reshape(x, dim = [N, P, C//P, H, W])$  *
 $mean = mean(x, dim = D)$ 
 $var = var(x, dim = D)$ 
 $x = \frac{x - mean}{\sqrt{var + \epsilon}}$ 
 $x = reshape(x, dim = [N, C, H, W])$  *
return  $x \cdot \gamma + \beta$ 

```

End Function

The pseudocodes of these five algorithms are summarized in Algorithm 3. The two lines with asterisks (*) are only required for GN. P is a parameter called group number that needs to be set when applying GN. D denotes the dimension to calculate the mean and variance. When the dimensions of x are (N, C, H, W) , $D = 0$ means that the mean and variance are calculated only on the first dimension of x , which is N dimension. When Norm = BN, $D = [0, 2, 3]$, it means that the mean and variance are calculated on the N, H, W dimensions. When Norm = IN, $D = [2, 3]$, it means that the mean and variance are calculated on the H, W dimensions. When Norm = LN, $D = [1, 2, 3]$, it means that the mean and variance are calculated on the C, H, W dimensions. When Norm = CN, $D = [1]$, it means that the mean and variance are calculated on the C dimension. When Norm = GN, $D = [2, 3, 4]$, it means that the mean and variance are calculated on the $C//P, H, W$ dimensions. It can be seen from the pseudocode that the process of GN is first setting a parameter group number (P), then dividing the channel dimensions into small groups and then normalizing on $C//P, H$ and W dimensions. It can be seen here that the drawback of the GN algorithm is that a parameter P needs to be manually set, and there is not much theoretical guidance for this parameter setting. And the different settings of P will have great impact on the experimental results, which will be explained in more detail in the experimental section.

As we have repeatedly emphasized, because the four normalization methods IN, LN, CN and GN are not normalized in the batch size dimension, all of these four approaches can be used as the normalization method for data isolated deep learning to replace the traditional BN method. Next, we will use the experiments to illustrate the feasibility of these four approaches.

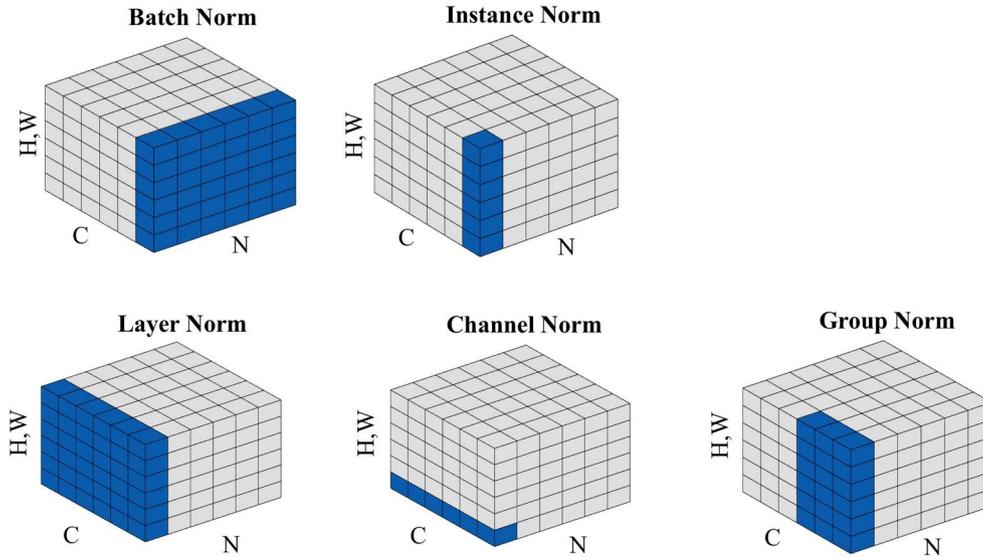


FIGURE 4 Five normalization approaches

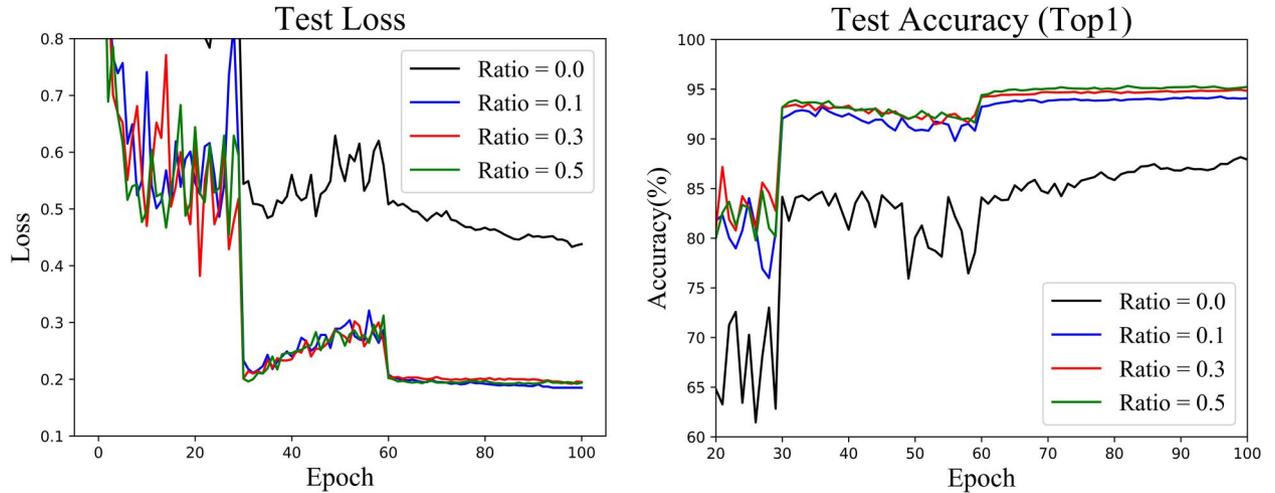


FIGURE 5 Batch normalization results under different imbalanced ratios

6 | EXPERIMENTS

After introducing the four normalization approaches for data isolated deep learning, this section provides experiments to verify our proposed data isolated distributed deep learning framework by using these normalization methods. There are two parts of experiments included. First, we conduct the local BN experiments in the case of imbalanced data training to show that the local BN is not suitable for data imbalance distribution caused by data isolation scenario. Here, the local BN means using each node to do BN according to its own local data, without the data exchange of mean and variance. Next, we give a comparison of the experimental results of local BN, IN, LN, CN and GN, while distributed BN is not implemented because of its huge communication overhead. The results of this part experiment illustrate the feasibility of using IN, LN, CN and GN for data isolation neural network training.

As regards the experimental settings, all of the experiments are executed on the Cifar10 data set [35], the network structure uses resnet18 [36] and the optimizer uses the ordinary SGD. Three processes are used to conduct experiments, one of which is the master node and the other two are worker nodes. The master node is responsible for collecting the gradients calculated by each worker, and updating the weights after calculating the average gradients. The worker nodes are responsible for calculating a new round of gradients with the latest weights returned by the master.

6.1 | Local batch normalization on imbalanced data isolation

In the first experiment, we show that the local BN is not suitable for distributed neural network training under data

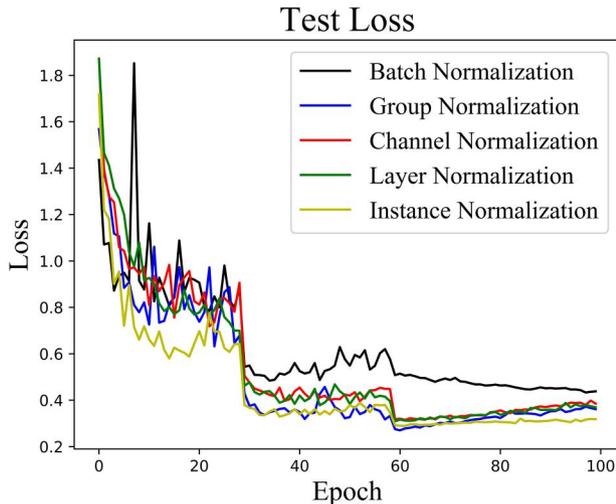
imbalance distribution. Here first, the cifar10 data set is created into imbalanced data. Cifar10 is a data set containing 10 types of data (0–9 types), including 50,000 training data sets and 10,000 test data sets. For the master node, all training sets and test sets are assigned to the master node for evaluating the training results. Then for the two workers nodes, we first divide the entire training set into two subsets $Subset_1$ and $Subset_2$. $Subset_1$ contains all the training data in categories 0–4 with a total of 25,000 images, 5000 in each category; $Subset_2$ contains all the training data in categories 5–9 with a total of 25,000 images, 5000 in each category. Next, we control the data imbalance by setting a parameter Ratio (R). Equations (9) and (10) give the training data set obtained by $worker_1$ and $worker_2$.

$$worker_1Dataset = R \cdot Subset_1 + (1 - R) \cdot Subset_2 \quad (9)$$

$$worker_2Dataset = (1 - R) \cdot Subset_1 + R \cdot Subset_2 \quad (10)$$

Here R ranges from 0% to 50%. It can be seen that when R is 0%, $worker_1$ is allocated to all $Subset_2$ for training, and $worker_2$ is allocated to all $Subset_1$, then the data of the two workers are regarded as completely imbalanced. When R is 50%, both $worker_1$ and $worker_2$ are allocated to 50% of $Subset_1$ and 50% of $Subset_2$. At this time, the data of the two workers are considered to be completely balanced.

Figure 5 and Table 1 show the experimental results under different R settings. Figure 5 shows the changes in loss and top 1 accuracy on the test set. Table 1 shows top 1 accuracy and top 5 accuracy. It can be seen that as the imbalance of the data becomes higher (R becomes smaller), the model trained with the local BN performs worse on the test set. This proves that the local BN is no longer applicable for deep learning in data imbalance scenarios caused by data isolation as explained in Section 4.1. Therefore, the following experiments use different normalization methods to prove that IN, LN, CN and GN are the feasible approaches to the normalization problem in neural network training for data isolation.



6.2 | Comparison of normalization methods

In this part of the experiment, the results of the five normalization methods (local BN, IN, LN, CN and GN) for distributed neural network training under data imbalance are given. First of all, the previous Section 5 has explained that the imbalanced data does not affect the four normalization methods of IN, LN, CN and GN. Therefore, the following experiments are conducted under the setting of parameter $R = 0\%$, that is, the data is completely imbalanced.

Figure 6 and Table 2 show the performance of the five normalization methods local BN, GN, LN, IN and CN on the test set. Here the parameter group number (P) in GN is set to 32, according to the optimal parameter setting proposed [3]. It can be seen that GN, LN, IN and CN have achieved better results than the local BN. Under this parameter setting, the

TABLE 1 Batch normalization results under different imbalanced ratios

Ratio (R)	Top 1 accuracy (%)	Top 5 accuracy (%)
0.0	88.15	99.87
0.1	99.80	94.93
0.3	94.25	99.93
0.5	95.32	99.92

TABLE 2 Results of different normalization methods for imbalanced data isolation

Norm method	Top 1 accuracy (%)	Top 5 accuracy (%)
BN	88.15	99.80
IN	91.78	99.71
LN	90.88	99.74
CN	90.61	99.74
GN	92.32	99.83

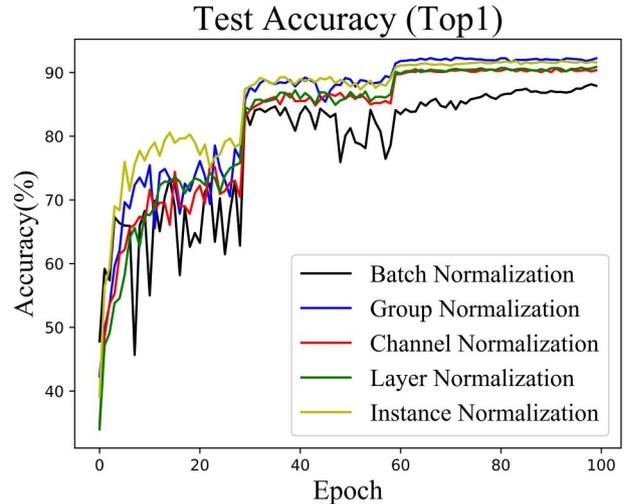


FIGURE 6 Results of different normalization methods for imbalanced data isolation

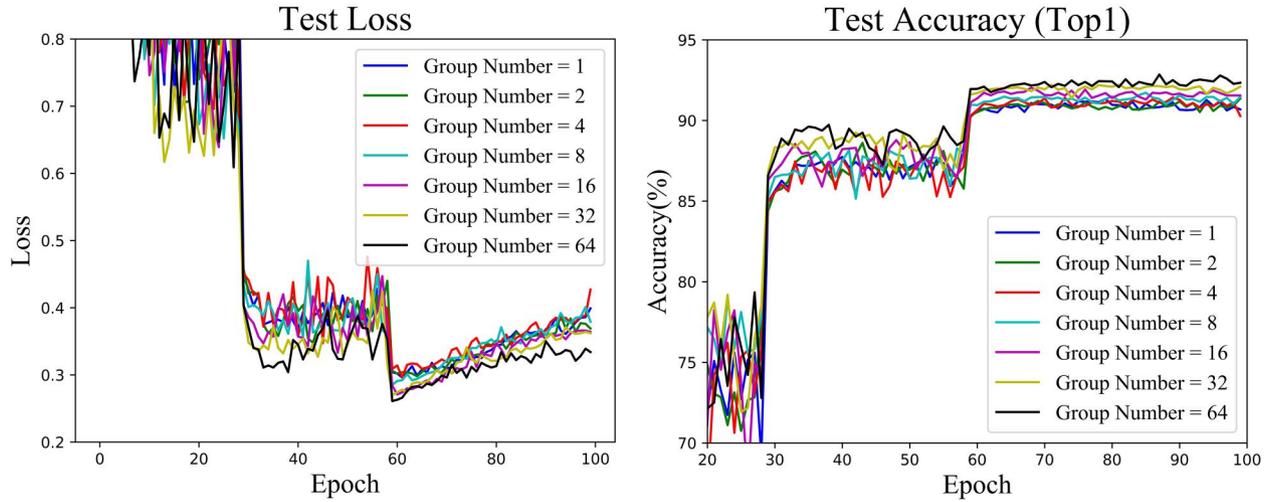


FIGURE 7 Group normalization results under different parameter group numbers

TABLE 3 Group normalization results under different parameter group numbers

Group number	Top 1 accuracy (%)	Top 5 accuracy (%)
1	91.33	99.71
2	91.36	99.76
4	91.42	99.68
8	91.72	99.76
16	92.04	99.77
32	92.22	99.76
64	92.85	99.84

model trained with the GN method performs best. The experiment illustrates that the GN, LN, IN and CN methods are all the feasible normalization approaches for deep learning of data imbalance caused by data isolation scenario.

Next, the experiment of GN with different parameter group number (P) settings is implemented. Figure 7 and Table 3 show the performance of the experiment on the test set. It can be seen that the model using the GN method has a large difference in performance under different group number settings. Therefore, although GN performs best on the cifar10 data set under optimal parameter settings, GN still has the drawback of manual tuning. Therefore, in the actual data isolation neural network training process, we can choose different normalization solutions proposed to complete the deep learning according to the actual situation.

7 | CONCLUSION AND FUTURE WORK

This study addresses the problem of neural network training in data isolation scenarios, in which the data of each node cannot be shared. When a global model that needs learning from all data is required, we need to use data isolated distributed training. The obvious challenge in this case is the imbalanced

distribution of data at each node. This will cause BN algorithm in the neural network no longer applicable. Based on the above problems and challenges, this study first gives the framework of data isolated distributed training based on SSGD algorithm. For the normalization problem of imbalanced data, we prove from both theoretical and experimental aspects that local BN is not suitable for imbalanced data distribution, because it greatly reduces the accuracy of the model. On the other hand, we theoretically illustrate the impracticality of distributed BN because of its huge communication overhead. Further, we adopt four normalization approaches from other fields: IN, LN, CN and GN to replace the BN layer in the neural network. These four algorithms are independent from data distribution, and we explain from theoretical analysis and experiment that they can be implemented directly for imbalanced data distributed training. These four methods achieve higher accuracy than the local BN in the data isolation scenario. Among them, GN achieves the highest accuracy, but manually tuning the group number is defective in the implementation of GN.

For future work, Section 5 gives four normalization approaches that are feasible in data isolated distributed deep learning, but we have not discussed in depth which normalization is most suitable in different cases. This part of the work needs more experimental and theoretical explorations in the future. In addition, all the discussions in this work are based on the SSGD algorithm. Compared with the ASGD algorithm, it is easier to prove the feasibility in theory, although it has to lose the efficiency of training. Therefore, our future work will also focus on the ASGD method and explore its implementation and feasibility in the distributed neural networks training with data isolation.

ACKNOWLEDGEMENTS

This research is supported by the National Natural Science Foundation of China under Grant Nos. 61873119 and 61911530247, and the Science and Technology Innovation Commission of Shenzhen under Grant Nos. KQJSCX 20180322151418232.

ORCID

Yujue Zhou  <https://orcid.org/0000-0003-3990-5408>

REFERENCES

1. Mayer, R., Jacobsen, H.A.: Scalable deep learning on distributed infrastructures. *ACM Comput. Surv.* 53(1), 1–37 (2020)
2. Hegde, V., Usmani, S.: Parallel and distributed deep learning. In: Technical reports, Stanford University, (2016)
3. McMahan, B., et al.: Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics, pp. 1273–1282. PMLR, Fort Lauderdale (2017)
4. Jochems, A., et al.: Distributed learning: developing a predictive model based on data from multiple hospitals without data leaving the hospital - a real life proof of concept. *Radiother. Oncol.* 121(3), 459–467 (2016)
5. Ben-Nun, T., Hoefler, T.: Demystifying parallel and distributed deep learning. *ACM Comput. Surv.* 52(4), 1–43 (2019)
6. Zinkevich, M., et al.: Parallelized stochastic gradient descent. In: Advances in neural information processing systems, pp. 2595–2603 (2010)
7. Dean, J., et al.: Large scale distributed deep networks In: Advances in neural information processing systems, pp. 1223–1231 (2012)
8. Lian, X., et al.: Asynchronous parallel stochastic gradient for nonconvex optimization. In: Advances in neural information processing systems, pp. 2737–2745 (2015)
9. Avron, H., Druinsky, A., Gupta, A.: Revisiting asynchronous linear solvers. *J. Acm.* 62(6), 1–27 (2015)
10. Zheng, S., et al.: Asynchronous stochastic gradient descent with delay compensation. In: International conference on machine learning. PMLR, pp. 4120–4129. International Convention Centre, Sydney, Australia (2017)
11. Muller, U., Gunzinger, A.: Neural net simulation on parallel computers, Proceedings of 1994 IEEE international conference on neural networks (ICNN 94). IEEE. vol. 6, pp. 3961–3966. New York, NY, USA (1994)
12. Ericson, L., Mubvha, R.: On the performance of network parallel training in artificial neural networks. ArXiv, vol. abs/1701.05130, Cornell University, USA (2017)
13. Ngiam, J., et al.: Tiled convolutional neural networks. In: Advances in neural information processing systems, pp. 1279–1287 (2010)
14. Yadan, O., et al.: Multi-GPU training of convnets. CoRR. 1312, 5853. abs/Cornell University, USA (2014)
15. Krizhevsky, A.: One weird trick for parallelizing convolutional neural networks. ArXiv, vol. abs/1404.5997, Cornell University, USA (2014)
16. Ben-Nun, T., et al.: Memory access patterns: the missing piece of the multi-GPU puzzle. In: SC 15: Proceedings of the international conference for high performance computing, networking, storage and analysis, pp. 1–12. IEEE, Fort Lauderdale (2015)
17. Gaunt, A.L., et al.: Ampnet: asynchronous model-parallel training for dynamic neural networks. Cornell University, USA (2018)
18. Sanil, A.P., et al.: In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining, pp. 677–682 Privacy preserving regression modelling via distributed computation (2004)
19. Gascón, A., et al.: Privacy-preserving distributed linear regression on high-dimensional data. In: Proceedings on privacy enhancing technologies. 2017(4), 345–364. New York, NY (2017)
20. Mohassel, P., Zhang, Y.: SecureML: a system for scalable privacy-preserving machine learning. In: 2017 IEEE symposium on security and privacy (SP), pp. 19–38. IEEE, Fort Lauderdale (2017)
21. Yang, Q., et al.: Federated machine learning: concept and applications. *ACM Trans. Intell. Syst. Technol.* 10(2) 1–19 Jan. (2019)
22. Duan, M., et al.: Astraea: self-balancing federated learning for improving classification accuracy of mobile deep learning applications. In: 2019 IEEE 37th international conference on computer design (ICCD), pp. 246–254. IEEE, Fort Lauderdale (2019)
23. Lyu, L., et al.: Collaborative fairness in federated learning. *Federated Learning*, pp. 189–204. Springer, HongKong (2020)
24. Verma, D.C., et al.: Artificial intelligence and machine learning for multi-domain operations applications. International society for optics and photonics. Approaches to address the data skew problem in federated learning. vol. 11006, pp. 110061I. Bellingham, Washington USA (2019)
25. Wang, L., et al.: Towards class imbalance in federated learning. arXiv preprint arXiv:2008.06217 (2020)
26. Li, L., et al.: Learning-based modeling and optimization for real-time system availability. *IEEE Trans. Comput.* 70(4), 581–594 (2020)
27. Bottou, L.: Stochastic gradient learning in neural networks. Proceedings of neuro-nimes. Nîmes, France 91(8), 12 (1991)
28. Das, D., et al.: Distributed deep learning using synchronous stochastic gradient descent. ArXiv, vol. abs/1602.06709, Cornell University, USA (2016)
29. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning, pp. 448–456. Cornell University, USA (2015)
30. Peng, C., et al.: In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 6181–6189. Megdet: a large mini-batch object detector. IEEE, Fort Lauderdale, FL, USA (2018)
31. Ulyanov, D., Vedaldi, A., Lempitsky, V.: Instance normalization: the missing ingredient for fast stylization. (vol. 1607). Cornell University, USA (2016)
32. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. *Stat.* 1050, 21 (2016)
33. Wu, Y., He, K.: Group normalization. In: Proceedings of the European conference on computer vision (ECCV), pp. 3–19. Cornell University, USA (2018)
34. Dai, Z., Heckel, R.: Channel normalization in convolutional neural networks avoids vanishing gradients. In: International conference on machine learning, Workshop Deep Phenomena, Cornell University, USA (2019)
35. Krizhevsky, A., et al.: Learning multiple layers of features from tiny images. Citeseer, Princeton, New Jersey, USA (2009)
36. He, K., et al.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778. IEEE, Fort Lauderdale, FL, USA (2016)

How to cite this article: Zhou Y, He L, Yang SH. Developing normalization schemes for data isolated distributed deep learning. *IET Cyber-Phys. Syst., Theory Appl.* 2021;1–11. <https://doi.org/10.1049/cps2.12004>