A consensus-based decentralized training algorithm for deep neural networks with communication compression

Bo Liu\textsuperscript{a}, Zhengtao Ding\textsuperscript{a,}\textsuperscript{*}

\textsuperscript{a}Department of Electrical and Electronic Engineering, University of Manchester, Sackville Street Building, Manchester M13 9PL, UK

Abstract

Facing the challenge of distributed computing on processing large-scale data, this paper proposes a consensus-based decentralized training method with communication compression. First, the decentralized training method is designed based on the decentralized topology to reduce the communication burden on the busiest agent and avoid any agent revealing its locally stored data. The convergence of the decentralized training algorithm is then analyzed, which demonstrates that the decentralized trained model can reach the minimal empirical risk on the whole dataset, without the sharing of data samples. Furthermore, model compression combined with the error-compensated method is considered to reduce communication costs during the decentralized training process. At last, the simulation study shows that the proposed decentralized training with error-compensated communication compression is applicable for both IID and non-IID datasets, and exhibits much better performance than the local training method. Besides, the proposed algorithm with an appropriate compression rate shows comparable performance with decentralized training and centralized training, while saving a lot of communication costs.

Keywords: Distributed training; Consensus; Model compression; Neural network; Decentralized communication topology; Convergence.

1. Introduction

Recent years have witnessed the rapid development of artificial intelligence in both theory and application \cite{1}, especially for deep learning \cite{2}, which has been widely applied in many real-life scenarios, for example, face identification \cite{3}, machine translation \cite{4}, and natural language processing \cite{5}. Besides, the affordable price and exponentially rising computing power of intelligent terminal devices, such as private computers and intelligent phones, makes it possible for the application of artificial intelligence in our daily life. This combination of information technology and artificial intelligence greatly enriches our life but also brings some practical problems.

\textsuperscript{*}Corresponding author

Email address: zhengtao.ding@manchester.ac.uk (Zhengtao Ding)

Preprint submitted to Neurocomputing

December 4, 2020
First of all, a huge amount of data are generated and collected by different devices in our daily life, which makes it difficult and inefficient for a central server to gather and process all the data [6]. Besides, these devices are becoming increasingly multi-functional and intelligent, which are usually equipped with a variety of sensors to collect data, and even powerful hardware and software to process data. The advent and development of 5G and Internet of things (IoT) [7] allow intelligent agents to communicate with each other, not just the central server. This combination of intelligent devices with computational ability has promoted the progress of fog computing [8] and edge computing [9], which considerably reduces the computation and communication burden on the cloud server, and provides more efficient service. Furthermore, there is a growing concern on data privacy and security issues [10], which makes this central computing method unsuitable for processing private and sensitive data, such as medical data [11] and personal habit data [12]. Based on the above consideration, an increasing number of users prefer to store and process their data on the local agent rather than reveal it to a central server.

It is important that the local agent can still use the information of other agents for better modeling even though without actually sharing the original data. Therefore, we are facing the problem that how to deal with the massive data distributed stored on many connected intelligent devices with the limitation that each agent can not reveal its local data to a central node or any other agents. Distributed computation [13] seems a potential solution for this problem, which not only alleviates the computing burden on the central node but also makes full use of all local agents’ computational resources. Because of its excellent ability for large-scale and privacy-related problems, different distributed machine learning algorithms are proposed for both academic research and industrial applications [14, 15], which mainly include centralized training and decentralized training.

Federated learning [14] is a typical centralized training method, which allows many agents to cooperatively train a shared model without exchanging local data. During the training process, each agent downloads the latest model from the central server and then trains this model with the local data to obtain a local update. All agents’ local updates are subsequently collected and averaged to optimize the shared model by the centralized server. A recurrent neural network federated learning framework for keyboard next-word prediction is developed in [16], which shows that the federated algorithm outperforms the server-trained method on the language model, and benefits the privacy protection. Zhao et al. [17] analyzed the accuracy reduction of federated learning on non-IID (independent and identically distributed) data [18] and proposed a method to alleviate this reduction, where a small sub-dataset is globally shared to warm up the local model. Simulations show that this data-sharing strategy greatly improves the performance of federated learning on non-IID data.

Different from centralized training, decentralized training does not require a central node to collect and integrate all agents’ information. Scardapane et al. [19] designed a decentralized training architecture for echo state networks based on alternating direction method of multipliers, which only needs local communication
among connected agents and shows comparable performance with the centralized model on efficiency and accuracy. Tang et al. [20] proposed a variance reduction method for decentralized parallel training on the distributed stored data with large variance. The theoretical analysis and empirical simulation demonstrate that the proposed method is more robust to data variance and shows better performance than decentralized parallel stochastic gradient descent [21].

Another issue for the common distributed training algorithms is the expensive communication costs among connected agents, because all agents need to share their model parameters or gradients with others frequently during the training process. Sattler et al. [22] designed the sparse ternary compression to compress both the upstream and downstream communications for federated learning, which considerably reduces communication costs and is robust to the non-IID data. However, this compression method does not fit the decentralized training scenario, and may cause divergence. Tang et al. [23] provided extrapolation compression and difference compression for decentralized training, which is proved to be able to match the convergence rate of full precision and show good performance on networks with high latency and low bandwidth. Konevncy et al. [24] proposed two different methods to decrease communication costs on the uploading process, one of which is to learn a restricted update with a part of model parameters, while the other method computes a full model update and then compresses it. However, combining the error-compensated strategy is not explored to further improve its performance by decreasing the influence of the compression operation.

In this paper, we explore to propose a distributed training framework for deep neural networks with communication compression, which not only reduces communication costs but also ensures the model accuracy. Inspired by the idea from consensus control that decentralized connected agents can reach consensus with only local communication, we design a consensus-based decentralized training algorithm, which is proved to be able to make all agents converge to the optimum. Then, the error-compensated compression method is applied to reduce the communication burden during the consensus process of decentralized training. Simulation study demonstrates that the decentralized training with error-compensated communication compression shows great performance on both IID and non-IID datasets, while significantly saving communication costs.

The remainder of this paper is organized as follows. After introducing the preliminaries on centralized training and the consensus algorithm, section 2 presents the problem of distributed training based on some practical challenges. Section 3 designs the decentralized neural network with the convergence analysis, and proposes the decentralized training framework with communication compression combined with the error-compensated compression strategy. Section 4 conducts the experimental simulation to verify the effectiveness of the proposed algorithm. Section 5 concludes this paper.
2. Preliminaries and motivation

2.1. Centralized training

Distributed computation is an efficient processing manner for large-scale and privacy-related data for application in big data, wireless sensor networks, and Internet of Things. The main feature of distributed machine learning is about many connected agents cooperatively training a model, which speeds up the training process and avoid the sharing of data samples. It not only takes better advantage of the distributed computational resources but also resolves the privacy-related issues.

Considering the following distributed training problem, where different sub-datasets are distributed stored on many connected agents over a communication graph, and the target is to train a model with minimal empirical risk on the entire dataset.

\[
\min_{\theta} E(D, \theta) = \sum_{k=1}^{K} e(D_k, \theta_k),
\]

where the model parameter is defined as a row vector \( \theta \in \mathbb{R}^{1 \times N} \), with \( N \) being the number of parameters. \( \theta_k \) is the model parameter of agent \( k \), \( K \) is the number of agents, \( E(D, \theta) \) and \( e(D_k, \theta_k) \) are the empirical risks over the entire dataset \( D \) and the local dataset \( D_k \) of agent \( k \), respectively.

The parameter server architecture shown in Figure 1(a) is a typical communication topology for centralized training methods [25], where multiple working agents are connected with a central server. In this topology, each working agent only has access to its locally stored sub-dataset, and the communication between the server and the working agents is captured by the two operations of pull and push [26]. During each training iteration, each working agent pulls the latest model parameter from the server and computes the gradient on its local sub-dataset, which is then pushed to the server. After receiving the gradients from all the working agents, the server accumulates these gradients to update the model parameter.

![Figure 1: (a) Centralized communication topology, (b) Decentralized communication topology.](image-url)
In this centralized training case, the global gradient $\nabla E(\theta)$ on the entire empirical risk $E(D, \theta)$ decomposes into many local gradients $\nabla e(\theta_k)$ on local empirical risks $e(D_k, \theta_k)$. This is described as follows:

$$\nabla E(\theta) = \frac{\partial}{\partial \theta} E(D, \theta) = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \theta} l(x_i, y_i, \theta), \quad (2)$$

$$\nabla e(\theta_k) = \frac{\partial}{\partial \theta} e(D_k, \theta_k) = \frac{1}{n_k} \sum_{i=1}^{n_k} \frac{\partial}{\partial \theta} l(x_i, y_i, \theta_k), \quad (3)$$

where $l(x_i, y_i, \theta)$ is the loss of a single sample $(x_i, y_i)$ over the model parameter $\theta$, $\nabla E(\theta)$ and $\nabla e(\theta_k)$ are the gradients of $E(D, \theta)$ and $e(D_k, \theta_k)$, with $n$ and $n_k$ being the number of samples of the entire dataset $D$ and the sub-dataset $D_k$, respectively.

Combining (2) and (3), it is easy to get

$$\nabla E(\theta) = \sum_{k=1}^{K} \frac{n_k}{n} \frac{1}{n_k} \sum_{i=1}^{n_k} \frac{\partial}{\partial \theta} l(x_i, y_i, \theta_k) = \sum_{k=1}^{K} \frac{n_k}{n} \nabla e(\theta_k). \quad (4)$$

This shows that, with the central server averaging the gradients form all working agents, centralized training model on distributed stored sub-datasets generates the same gradient, and correspondingly has the same performance as the model built on the entire dataset.

2.2. Consensus algorithm

There are many popular distributed training algorithms designed for the centralized communication topology, such as federated learning [24], and asynchronous parallel gradient descent [27]. However, the heavy communication burden on the center node (server) is the main bottleneck of this centralized topology, as all working agents are required to communicate with the central server during the training process [28]. Too much pressure on the server may lead to communication blocking or even the failure of the server, especially for those communication systems with low bandwidth and high latency. In some extreme cases, the whole distributed training architecture would be out of operation, if the central node (server) fails.

To circumvent this problem, the decentralized communication topology is designed, where no center node is required, and all agents only communicate with their neighbors. As shown in Figure 1(b), this decentralized framework not only reduces the requirement for communication endurance but also is robust to the failure of any agent. A decentralized graph is represented by $G = (V, E)$, where $V = \{1, 2, ...K\}$ and $E \subset V \times V$ denotes the set of agents and the set of edges, respectively. The connection strength of all agents over the graph is described by a matrix $W = [w_{ij}] \in \mathbb{R}^{K \times K}$, where the element $w_{ij} > 0$ if the edge $(i, j) \in E$ or $i = j$, and $w_{ij} = 0$ otherwise [29]. Specifically, the larger value of $w_{ij}$ means more influence of agent $j$ on agent $i$, while $w_{ij} = 0$ denotes that these two agents are disconnected.
In this paper, we assume the communication topology is an undirected graph, and the connection strength matrix $W$ satisfies $w_{ij} = w_{ji} \in [0, 1], \forall (i, j)$ and $\sum_{j=1}^{K} w_{ij} = 1, \forall i$ to ensure the consensus of all agents. More specifically, we obtain the matrix $W$ using the Max-degree [30] strategy, that is

$$w_{ij} = \begin{cases} \frac{1}{d_{i}} & \text{if } i \text{ is connected to } j \\ \frac{1}{d_{i}} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases},$$

$$\sum_{k=1}^{K} w_{kj} = 1, \forall j$$

(5)

where $d_{i}$ denotes the number of agents that are connected to the agent $i$, namely the degree of agent $i$, and $d_{\text{max}}$ is the largest $d_{i}$ in the communication topology. Denoting the model of agent $k$ as a parameter vector $X_{k}$, the consensus update of $X_{k}$ is given by

$$X'_{k} = \sum_{j=1}^{K} w_{kj} X_{j},$$

(6)

where $X'_{k}$ denotes the updated $X_{k}$ by the consensus algorithm with a single consensus step.

Here, we define a mode parameter matrix $X = [X_{1}, X_{2}, ..., X_{K}]^{T} \in \mathbb{R}^{K \times N}$ for all agents, the updated rule of the consensus process is then given by

$$X' = WX,$$

(7)

where $X'$ is the updated parameter matrix $X$ of all agents after a single consensus step.

The consensus algorithm impels all agents in a decentralized topology to approach their mean value $ar{X} = \frac{1}{K} \sum_{k=1}^{K} X_{k}$ by repetitively computing (6), and it is commonly required more than ten consensus steps for all agents to achieve consensus.

2.3. Motivation

A huge amount of data is generated and collected by different intelligent devices every day, and these data are often concerned with privacy and security related issues. This makes it not appropriate to be processed by a centralized server. Decentralized training seems a promising solution for this problem, but the main issue of which is the massive communication costs among connected agents. Therefore, we expect to reduce the amount of communication during the decentralized training process. Besides, the diversity of data should also be taken into account. The distributed stored data may be particularly different on the distribution and size, as the data from different devices depend heavily on their environmental usage scenarios.

For these problems, we plan to design a decentralized training method, which should meet the following requirements: 1) It is suitable for decentralized communication topology to alleviate the communication burden on the center node and improves its extensibility; 2) There is no sharing of data samples among connected agents out of privacy issues; 3) Communication compression strategy is considered to reduce
communication costs during the training process; 4) It is robust to the non-IID dataset for the diversity of different agents.

3. Decentralized neural network with communication compression

3.1. Decentralized neural networks

Since that the consensus algorithm can drive all models in a decentralized model to their mean value, a simple idea for realizing decentralized training is to average all models with enough consensus steps after each training iteration. It is easy to demonstrate that this decentralized training method has almost the same performance as centralized training. However, the large number of consensus steps among connected agents leads to expensive communication costs, which is especially true for deep neural networks with millions of model parameters.

Even though a single consensus communication does not drive all agents to their average, but it still makes all agents approach each other. Given that the connection strength matrix $W$ satisfies $\sum_{j=1}^{K} w_{ij} = 1$ and $w_{ij} \in [0, 1)$, we have

$$\max_k \|\theta'_k\| = \max_k \| \sum_{j=1}^{K} w_{kj} \theta_j \| \leq \max_k \|\theta_k\|$$

(8)

Similarly, we also have $\min \|\theta'_k\| \geq \min \|\theta_k\|$, and this means that a consensus step always reduces the upper bound and improves the lower bound of all the agents in the communication topology. Inspired by the idea that gradient descent method optimizes the model parameter and the consensus algorithm draws all agents to approach each other, we propose the consensus-based decentralized training framework for neural networks, which requires consensus communication only once at every training step. This algorithm consists of two update phases, which are written as

$$\theta'_k = \theta_k - \eta \nabla c(\theta_k),$$

(9)

$$\theta'' = W \theta','$$

(10)

where $\theta'_k$ is the locally trained model parameter of $\theta_k$, $\theta' = [\theta'_1, \theta'_2...\theta'_K]^T \in \mathbb{R}^{K \times N}$ and $\theta''$ are the parameter matrix of all agents before and after a single consensus step.

Local training defined by (9) is the first phase, where all agents over the graph simultaneously train their model parameters based on locally stored sub-datasets using gradient descent method. The second phase is globally consensus update described by (10), where these locally trained agents share their model parameters with directly connected neighbors to globally update their model parameters using the consensus
algorithm. By repeating these two procedures, all the agents converge to the optimal model, and their overall performance is similar to the model trained on the entire dataset.

On the one hand, each agent only needs consensus communication once after every training step, which avoids repetitive consensus communication and alleviates the communication costs. On the other hand, all agents are not required to share their local data or upload them to a central server, which benefits privacy protection. Besides, this decentralized communication topology has great flexibility and expansibility, that is, the training process is unaffected in switching communication graphs, provided that the topology always remains a spanning tree.  

The process of the decentralized training algorithm for neural networks is detailed in Table 1.

Table 1: The process of decentralized neural networks.

<table>
<thead>
<tr>
<th>Algorithm 1: Decentralized training for neural networks.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inputs:</strong> The architecture of neural networks and sub-datasets ((D_1, D_2, \ldots, D_K)) stored on each agent, the communication topology with the number of agents (K), and the connection strength matrix (W).</td>
</tr>
<tr>
<td><strong>Outputs:</strong> The optimum neural network model parameter (\theta^*).</td>
</tr>
<tr>
<td>1: Initialize the neural network model parameter (\theta_k) of each agent over the communication topology.</td>
</tr>
<tr>
<td>2: Every agent optimizes the neural network using gradient descent with its stored sub-dataset (D_k) to obtain the locally optimized (\theta'_k).</td>
</tr>
<tr>
<td>3: Update the parameter matrix (\theta') for all agents with a single consensus communication to obtain (\theta'').</td>
</tr>
<tr>
<td>4: Back to step 2 with globally updated (\theta'').</td>
</tr>
<tr>
<td>5: Check the stop criterion (such as a predefined number of training iterations).</td>
</tr>
<tr>
<td>6: Return the optimum neural network parameter (\theta^*).</td>
</tr>
</tbody>
</table>

3.2. Convergence analysis

For the distributed training problem described in (1), the update rule for all agents’ model parameters using the Algorithm 1 is

\[
\theta^{t+1} = W(\theta^t - \eta \nabla e(\theta^t)),
\]

where \(\theta^t = [\theta_1^t, \theta_2^t, \ldots, \theta_K^t]^T\) and \(\nabla e(\theta^t) = [\nabla e(\theta_1^t), \nabla e(\theta_2^t), \ldots, \nabla e(\theta_K^t)]^T\) are the concatenation of all agents’ model parameters and gradients at the \(t\)th iteration, respectively.

**Assumption 1:** The gradients \(\nabla e(\theta_k)\) of all agents’ empirical risks are Lipschitz continuous with constant \(L_e > 0\).
Assumption 2: The connection strength matrix $W$ satisfies $W = W^T$ and $1^T W = 1$, where $1 = [1 \; 1 \ldots 1]^T \in \mathbb{R}^{K \times 1}$.

Theorem 1: The overall empirical risk $\sum_{k=1}^{K} e(\theta_k)$ of all agents converge to the minimum using the proposed decentralized training algorithm under Assumptions 1 and 2.

Proof: Inspired by [32], we design a function to represent the update process of the model parameters using the decentralized training method, which is defined as

$$\mathcal{L}(\theta) = \frac{1}{2\eta} \|\theta\|_{I-W}^2 + F(\theta),$$

where $\|\theta\|_{I-W}^2 = \langle \theta, (I-W)\theta \rangle$ with $I$ being $K \times K$ unit matrix, and $\nabla F(\theta) = W\nabla e(\theta)$.

The gradient of $\mathcal{L}(\theta)$ is

$$\nabla \mathcal{L}(\theta) = \frac{1}{\eta} (I-W)\theta + W\nabla e(\theta).$$

Then, we reformulate (11) as

$$\theta_{t+1} = \theta_{t} - \eta \left[ \frac{1}{\eta} (I-W)\theta_{t} + W\nabla e(\theta_{t}) \right]$$

$$= \theta_{t} - \eta \nabla \mathcal{L}(\theta_{t}),$$

where we find that (12) generates the same sequences of $\theta$ as (11), using gradient descent with the same learning rate $\eta$.

Also, it is obtained from (14) that

$$\langle \nabla \mathcal{L}(\theta_{t+1}), \theta_{t+1} - \theta_{t} \rangle = -\frac{\|\theta_{t+1} - \theta_{t}\|^2}{\eta}. \tag{15}$$

Since that all $\nabla e(\theta_k)$ is $L_e$-Lipschitz, $|\nabla e(\theta_{t+1}^k) - \nabla e(\theta_{t}^k)| \leq L_e |\theta_{t+1}^k - \theta_{t}^k|.$

$$|\nabla F(\theta_{t+1}) - \nabla F(\theta_{t})| = |W(\nabla e(\theta_{t+1}) - \nabla e(\theta_{t}))| \leq L_e |\theta_{t+1} - \theta_{t}|. \tag{16}$$

Thus,

$$|\nabla \mathcal{L}(\theta_{t+1}) - \nabla \mathcal{L}(\theta_{t})|$$

$$= \left| \frac{1}{\eta} (I-W)(\theta_{t+1} - \theta_{t}) + \nabla F(\theta_{t+1}) - \nabla F(\theta_{t}) \right|$$

$$\leq \left( \frac{1}{\eta} (1 - \lambda_{min}(W)) + L_e \right) |\theta_{t+1} - \theta_{t}|. \tag{17}$$

where $\lambda_{min}(W)$ is the minimum eigenvalue of the matrix $W$.

We define $L_I = \frac{1}{\eta} (1 - \lambda_{min}(W)) + L_e$, then $\nabla \mathcal{L}(\theta)$ is $L_I$-Lipschitz.

It is therefore

$$\mathcal{L}(\theta_{t+1}) - \mathcal{L}(\theta_{t}) \leq \langle \nabla \mathcal{L}(\theta_{t}), \theta_{t+1} - \theta_{t} \rangle$$

$$+ \frac{L_I}{2} |\theta_{t+1} - \theta_{t}|^2. \tag{18}$$
Combining (15) and (18) we have

\[
L(\theta^{t+1}) - L(\theta^t) \\
\leq -\frac{1}{2\eta}(1 + \lambda_{\text{min}}(W)) - L_e \|\theta^{t+1} - \theta^t\|^2.
\] (19)

Sum (19) from 0 to +∞, we get

\[
L(\theta^{n+1}) - L(\theta^0) \\
\leq -\frac{1}{2}(1 + \lambda_{\text{min}}(W)) - L_e \sum_{t=0}^{\infty} \|\theta^{t+1} - \theta^t\|^2.
\] (20)

Set the learning rate \(\eta\) to satisfy \(0 < \eta < \frac{1 + \lambda_{\text{min}}(W)}{L_e}\), it is obtained that \(L(\theta)\) is non-increasing and upper bounded by \(L(\theta^0) < +\infty\). therefore

\[
\sum_{t=0}^{\infty} \|\theta^{t+1} - \theta^t\| < +\infty,
\] (21)

\[
\|\theta^{t+1} - \theta^t\| \rightarrow 0,
\] (22)

and we have \(\theta^{t+1} - \theta^t \rightarrow 0 \in \mathbb{R}^{K \times N}\).

Combining with (11), when \(t \rightarrow \infty\) we have

\[
W(\theta^t + \eta \nabla e(\theta^t)) = \theta^t.
\] (23)

Multiply both the right and left sides of (23) with \(1^T\), it is obtained

\[
1^T W(\theta^t + \eta \nabla e(\theta^t)) = 1^T \theta^t.
\] (24)

Therefore, we have \(\sum_{i=1}^{n} \nabla e(\theta^t_i) = 0\) when \(t \rightarrow \infty\), which means that \(E(\theta) = \sum_{k=1}^{K} e(\theta_k)\) converges to the minimum.

3.3. Error-compensated compression strategy

During the consensus process, all agents need to share the model parameters with their neighbors, which brings heavy communication costs to the whole system, especially for deep neural networks with a huge number of model parameters. We, therefore, intend to compress the model parameter before each agent shares its model parameter with neighbors.

Bit-clipping [33] is a simple and efficient compression method, which directly sets the lower \(p\) bits into zero for any decimal real number. For example, provided \(p = 6\) we compress 1.2345678 into 1.2 with its lower 6 bits set to zero. This compression method saves a lot of storage memory but keeps the most valuable information on the model parameter. A straightforward idea is to share the model parameter after compressing the model parameter. In this way, only part model parameters are shared during the consensus process, which greatly reduces communication costs.
However, the compression method still loses some important information of the model parameter, which is harmful to the convergence. For this problem, the error-compensated compression method [34] is proposed, the main idea of which is to store the residue of the model parameter after the compression in the previous step and accumulate it to the current model parameter. Thus, the compression residue is not discarded but accumulated to update the model in the long run, which not only saves the communication resource but also benefits the convergence [33].

The update rule of the error-compensated compression strategy mainly consists of two steps. The first step is to compress the model parameter after combining it with the stored error, and this error is then updated in the second step.

\[ \tilde{\theta} = \theta + \delta, \]  
\[ \delta = \tilde{\theta} - C(\tilde{\theta}), \]

where \( \theta \) and \( \delta \) are the model parameter and the stored error, respectively. \( \tilde{\theta} \) and \( C(\tilde{\theta}) \) are the error-compensated model parameter before and after the compression process, with \( C \) being the compression strategy. The process of the error-compensated compress strategy is summarized in Table 2.

### Table 2: The procedure of the error-compensated compress strategy.

<table>
<thead>
<tr>
<th>Algorithm 2: Error-compensated compression strategy.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inputs:</strong> The model parameter ( \theta ), the stored error ( \delta ), the compression strategy ( C ).</td>
</tr>
<tr>
<td><strong>Outputs:</strong> The compressed model parameter ( C(\tilde{\theta}) ) and updated error ( \delta ).</td>
</tr>
<tr>
<td>1: Compute the error-compensated model parameter ( \tilde{\theta} = \theta + \delta ).</td>
</tr>
<tr>
<td>2. Compress the error-compensated model parameter to ( C(\tilde{\theta}) ) using the compression strategy ( C ).</td>
</tr>
<tr>
<td>3: Update the error ( \delta = \tilde{\theta} - C(\tilde{\theta}) ).</td>
</tr>
<tr>
<td>4. Return the compressed model parameter ( C(\tilde{\theta}) ) and the updated error ( \delta ).</td>
</tr>
</tbody>
</table>

### 3.4. Decentralized neural networks with error-compensated communication compression

Combining the consensus algorithm and the error-compensated compression strategy, we propose the decentralized training framework for neural networks with communication compression, which mainly consists of three steps. For each agent \( k \), it maintains a local model parameter \( \theta_k \), a compressed error-compensated model parameter \( C(\tilde{\theta}_k) \), and a stored error \( \delta_k \).

The first step is local training, where the model parameter \( \theta_k \) is locally updated using gradient descent by (9). The second step is model compression, where the stored error is added to the locally updated model parameter in (25), which is then compressed by the compression strategy \( C \) with error updating by (26). The third step is global consensus, the compressed error-compensated model parameters are shared among
connected neighbors, and the new consensus update process for each agent $k$ is

$$
\theta_k'' = w_{kk}\theta_k' + \sum_{j \in \mathcal{N}(k)} w_{ij}C(\tilde{\theta}_j),
$$

(27)

where $\mathcal{N}(k)$ represents the set of the neighbors of agent $k$, and it is notable that $k \notin \mathcal{N}(k)$.

The specific process of the decentralized neural networks with error-compensated communication compression is summarized in Table 3.

Table 3: The process of decentralized neural networks with error-compensated communication compression.

**Algorithm 3:** Decentralized training with error-compensated communication compression.

**Inputs:** The architecture of neural networks and sub-datasets ($D_1, D_2, \cdots D_K$) stored on each agent, the compression strategy $C$, the communication topology with the number of agents $K$, and the connection strength matrix $W$.

**Outputs:** The optimum neural network model parameter $\theta^*$.

1: Initialize the neural network model parameter $\theta_k$ and the stored error $\delta_k$ of each agent over the communication topology.

2: Every agent optimizes the neural network using gradient descent with its stored sub-dataset $D_k$ to obtain the locally optimized $\theta_k'$.

3: Compress the locally trained model parameter $\theta_k'$ with the stored error $\delta_k$ using Algorithm 2 to get $C(\tilde{\theta}_k)$ and the updated error $\delta_k$.

4: Receive $C(\tilde{\theta}_j)$ from its neighbor $j \in \mathcal{N}(k)$ and globally update the parameter $\theta_k'$ over the graph using (27) to obtain $\theta_k''$.

5: Return to step 2 with updated error $\delta_k$ and globally optimized $\theta_k''$.

6: Check the stop criterion (such as a predefined number of training iterations).

7: Obtain the optimum neural network model parameter $\theta^*$.

4. Simulation and discussion

4.1. Experimental setup

We benchmark the distributed training framework with communication compression on the handwritten digit images dataset MNIST [35], which is a standard image classification task. This dataset consists of 55000 training samples and 10000 test samples, where each sample (image) is given one of the 10 labels representing the handwritten digit from 0 to 9. The convolutional neural network (CNN) [36] is used for this image classification task, and we use a five-layer CNN in this simulation, which consists of two convolution layers, two pooling layers, and a fully connected layer with more than 3 million parameters.
Specifically, 32 and 64 filters of the size $5 \times 5$ kernel with 1 stride are used for the first and second convolution layers, respectively, and each convolution layer is followed by a max-pooling layer. The fully connected and output layers contain 1024 and 10 nodes, respectively. Besides, the rectified linear unit (ReLU) activation function is adopted for both convolution and fully connected layers, while the softmax activation function is applied for the output layer.

The decentralized communication topology in Figure 1(b) is taken as an example, where ten sub-datasets are stored on these agents over the graph, and ten CNNs with the same structure are also initialized on each agent. In this distributed training framework, there is no exchange of data samples, and all agents are only allowed to share the information of model parameters among their directly connected neighbors. On the one hand, we consider the ideal situation that the sub-datasets stored on different agents are IID data. For the balanced IID data case, the entire dataset is partitioned and distributed evenly stored on the ten agents over the communication graph. On the other hand, it is more reasonable in the practical scenario that the training sub-datasets stored on different agents depend heavily on their local environments and usage patterns, these sub-datasets are non-IID and highly divergent from the perspective of size and distribution. In the unbalanced non-IID data case, we consider the extreme situation that each agent has access to the data with only one label. The entire dataset is partitioned based on the label (from 0 to 9) and distributed on the ten agents over the graph. For both the IID and non-IID data cases, the following five algorithms are compared in this simulation, and the Adam optimizer with $10^{-4}$ learning rate is taken as the gradient descent method for all algorithms.

- **Local training**: each agent trains its CNN model only based on the locally stored sub-dataset, and there is no communication among these agents.

- **Centralized training**: all agents are connected in a centralized topology, and the central server is used to average all local models after every local training step.

- **Decentralized training (Algorithm 1)**: all agents are connected over the decentralized communication graph, each agent trains the model on its sub-dataset and share their full model parameters among directly connected neighbors, with the consensus algorithm globally updating their model parameters.

- **Decentralized training with error-compensated communication compression (Algorithm 3)**: the consensus algorithm is used to globally update the model parameters after local training, with the error-compensated compressed model information shared among directly connected neighbors.

- **Decentralized training with communication compression (Algorithm 4)**: the consensus algorithm is used to globally update the model parameters after local training, while the compressed model information using bit-clipping only (without error-compensation) are shared among directly connected neighbors.
4.2. Simulation results

In general, the model parameter is stored and shared in the floating-point number form, which usually has 8 significant digits. Bit-clipping with $p = 6$ (compression rate is $1/4$) is taken as the compression method for Algorithm 2 in this simulation, which saves $3/4$ communication costs. For both IID and non-IID datasets, we evaluate the performance of all algorithms on the same test dataset, which contains 10000 samples with all of the ten labels.

Figures 2 and 3 are the simulation results of the above five algorithms on the IID and the non-IID datasets, respectively, where the averaged classification accuracy of all the ten agents is taken as the performance for each algorithm. We find from Figure 2 that the classification accuracy on the test dataset of all the five algorithms increases dramatically at the beginning training process and then achieves convergence. It is notable that the decentralized training with communication compression (Algorithm 4) shows much worse performance than the other four algorithms. Especially, instead of improving the model accuracy, decentralized training with bit-clipping only (without error-compensation) significantly decreases agents’ performance. This means that the sharing of incomplete or inaccurate model information hurts the model accuracy to a large extent. Except for Algorithm 4, the local training exhibits the lowest model accuracy, although the classification accuracy of local training rises faster than decentralized training and decentralized training with error-compensated communication compression (Algorithm 3) at the beginning training stage. The centralized training and decentralized training with error-compensated communication compression has nearly the same performance, while the decentralized training shows a little better accuracy than them from the partially enlarged view in Figure 2. In particular, the decentralized training with error-compensated communication compression algorithm saves $3/4$ communication costs while the brought decrease in accuracy is little and negligible.

It is seen from figure 3 that the classification accuracy of the local training algorithm on the non-IID dataset is always around 0.1 over the whole training process, this is because each agent learns the samples with only one label. Notably, Algorithm 4 on the non-IID dataset also cannot achieve convergence, and the model accuracies of which are around 0.1. This shows that decentralized training on the non-IID dataset is more sensitive to the sharing of incomplete model information than on the IID dataset. Besides, centralized training and decentralized training have comparable good accuracy on the non-IID dataset, even though samples with only one label are available for each agent. This means that the consensus process effectively helps each agent learn from other agents’ sub-datasets, as long as all agents are connected over a graph with a spanning tree. For Algorithm 4, the averaged accuracy of all agents shows only a little worse performance than that of decentralized training. However, it has far better performance than the local training algorithm, which means that the decentralized training algorithm with an error-compensated compression strategy is robust to the non-IID dataset.

To further analyze the influence of the compression rate of the compression strategy with and without
error-compensation on the classification accuracy of decentralized training, we test the decentralized training with communication compression and error-compensated communication compression (Algorithm 3 and Algorithm 4) using bit-clipping with \( p \) from 0 to 7, on both IID and non-IID datasets. Correspondingly, the compression rate ranges from 0 to 1 at 1/8 intervals, with the compression rate of 0 and 1 representing the
local training and the decentralized training without communication compression (Algorithm 1), respectively.

![Figure 4: The influence of the compression rate on both IID and non-IID datasets.](image)

Figure 4 describes the performance of Algorithms 3 and 4 on both IID and non-IID datasets, with the compression rate ranging from 0 to 1, where each point shows the averaged value, the lower bound, and the upper bound of the classification accuracies for the ten agents in the decentralized communication topology. The classification accuracies of these points that do not converge are around 0.1, which equals the reciprocal of the number of classification classes. As mentioned above, the local training method does not converge on the non-IID dataset, while it shows fair performance on the IID dataset. With 1/8 compression rate, neither Algorithm 3 nor Algorithm 4 achieve convergence on IID or non-IID datasets, which means that too aggressive compression harms the convergence of the decentralized training algorithms with communication compression even though the error compensated strategy is applied.

It is easy to find from Figure 4 that Algorithm 3 shows far better performance than Algorithm 4 on both IID and non-IID datasets. More specifically, with the increase of compression rate, Algorithm 3 starts to show good performance on both IID and non-IID datasets when the compression rate is larger than 1/8, while Algorithm 4 cannot achieve convergence or falls in local optimum with poor accuracy until the compression rate reaches 5/8. When the compression rate reaches and exceeds 6/8, there is almost no difference between Algorithm 3 and 4, as the main model information is kept and only an unimportant part is clipped in these cases.
For the IID dataset, the performance of decentralized training with a 2/8 communication compression rate is far better than that of the local training model (compression rate is 0), which demonstrates that the error-compensated communication compression strategy significantly improves the model accuracy with only 1/4 more communication costs. After that, the classification accuracy rises slightly at the 3/8 compression rate and then almost stays the same with the increase of the compression rate. Similarly, for the performance on the non-IID dataset, the classification accuracy of the decentralized training with error-compensated communication compression first grows up (compression rate form 1/8 to 3/8) and then almost keeps unchanged with the increase of the compression rate. It is also notable that the gap between the upper and the lower bound of decentralized training with communication compression on the IID dataset is much smaller than that on the non-IID dataset. This shows that the unbalance of the dataset is harmful to the convergence of the decentralized training method.

From the overall view, the decentralized training with error-compensated communication compression using bit-clipping with $p = 5$ (compression rate is 3/8) shows nearly the same performance as the centralized training and decentralized training algorithm, but saves 5/8 communication costs. This verified that the proposed decentralized training with error-compensated communication compression method is effective and efficient.

5. Conclusion

The combination of artificial intelligence, information technology, and IoT drives the development of distributed computing to offer more efficient and diversified services. This development considerably enriches our daily life but also brings new challenges to distributed machine learning, including the huge burden on the server for processing massive data, privacy protection, communication costs on distributed training, and dealing with non-IID data.

In this paper, we propose a consensus-based decentralized training framework with error-compensated communication compression to deal with both IID and non-IID datasets. The consensus-based decentralized training algorithm is designed to free the center node, which not only benefits the privacy protection but also ensures the flexibility and expansibility of the communication system. It is proved in the convergence analysis that the decentralized neural networks converge to the model with minimal empirical risk on the whole dataset by consensus communication on model parameters, even though there is no sharing of data among all agents. The error-compensated compression method with bit-clipping is then applied to compress the model parameter before sharing, which significantly reduces communication costs. In the simulation study, it is shown that decentralized training with error-compensated communication compression is applicable on both IID and non-IID datasets, and has much better performance than the local training even with a small compression rate of 1/4. With a 3/8 compression rate, it shows almost the same classification accuracy as...
decentralized training and centralized training but saves a lot of communication costs. Furthermore, both decentralized training methods with and without error-compensated communication compression exhibit more differences among agents on the non-IID dataset than that on the IID dataset.

References


[22] F. Sattler, S. Wiedemann, K. Müller, W. Samek, Robust and communication-efficient federated learning from non-i.i.d. data, IEEE Transactions on Neural Networks and Learning Systems (2019) 1–14.