Journal of Parallel and Distributed Computing

AMBLE: Adjusting Mini-Batch and Local Epoch for Federated Learning with Heterogeneous Devices

--Manuscript Draft--

Manuscript Number:	
Article Type:	VSI:Edge-Intelligence-IoT [MGE - Andrzej Goscinski]
Section/Category:	Special Issue Papers
Keywords:	Federated Learning System Heterogeneity Local Mini-batch SGD Federated Averaging
Corresponding Author:	[redacted] KOREA, REPUBLIC OF
First Author:	Juwon Park
Order of Authors:	Juwon Park
	[redacted]
	[redacted]
	[redacted]
Abstract:	As data privacy becomes increasingly important, federated learning is a technique for training deep learning models while ensuring the data privacy of devices. By using federated learning, it is possible to obtain e ects similar to processing the entire data at once while independently processing data from various devices and institutions that have data, without collecting distributed local data in the central server. However, it is challenging to consider the system heterogeneity of devices in federated learning. In this paper, we propose AMBLE, which adaptively adjusts the local mini-batch and local epoch size for heterogeneous devices in federated learning, synchronously updating the parameters. Our proposed scheme, AMBLE, can add more computations during the waiting time caused by stragglers and can scale the local learning rate to improve the model convergence rate and accuracy. We confirm that federated learning with AMBLE can be stably trained with a faster convergence speed and higher accuracy than FedAvg and adaptive batch size scheme, in both the non-IID and IID cases.
Suggested Reviewers:	Blesson Varghese Queen's University Belfast B.Varghese@qub.ac.uk Dr. Varghese achieved great renown for his works in Edge Computing.
	Hasan Bulut Ege University: Ege Universitesi hasan.bulut@ege.edu.tr Dr. Bulut achieved great renown for his works in Deep Learning, NLP, and Cloud Computing.
	Daniel Ramage Google Inc dramage@google.com Dr. Ramage is the co-author of Federated Averaging (FedAvg) paper.
Opposed Reviewers:	

Cover Letter

[redacted]

20th June 2021

Dear Editors,

We are delighted to submit our manuscript titled “AMBLE: Adjusting Mini-Batch and Local Epoch for Federated Learning with Heterogeneous Devices” to be considered for publication in the special issue on Distributed Intelligence at the Edge for the Future Internet of Things, of Elsevier Journal of Parallel and Distributed Computing. We confirm that this work is original and has not been published elsewhere, nor is it currently under consideration for publication elsewhere.

In this article, we present our AMBLE, which adaptively adjusts the local mini-batch and local epoch size for heterogeneous devices in federated learning, synchronously updating the parameters. Our proposed scheme, AMBLE, can add more computations during the waiting time caused by stragglers and can scale the local learning rate to improve the model convergence rate and accuracy. We confirm that federated learning with AMBLE can be stably trained with a faster convergence speed and higher accuracy than FedAvg and adaptive batch size scheme, in both the non-IID and IID cases.

We believe these findings will be interest of the readers of your journal. We have no co [redacted]
to disclose and please address all correspondence concerning this manuscript to me at
We do not provide the list of suggested reviewers currently, since we are not sure abo
process. If it is still required, please let us know.

Thank you for your consideration of this manuscript.

Sincerely,

[redacted]

Highlights

- AMBLE is the scheme for heterogeneous devices in federated learning.
- AMBLE adjusts local mini-batch and local epoch adaptively.
- AMBLE solves the straggler problem caused by system heterogeneity.
- AMBLE can add more computation while waiting time caused by stragglers.
- AMBLE adopt learning rate scaling to improve performance.

AMBLE: Adjusting Mini-Batch and Local Epoch for Federated Learning with Heterogeneous Devices

Juwon Park[a], [redacted]

Abstract

As data privacy becomes increasingly important, federated learning is a technique for training deep learning models while ensuring the data privacy of devices. By using federated learning, it is possible to obtain effects similar to processing the entire data at once while independently processing data from various devices and institutions that have data, without collecting distributed local data in the central server. However, it is challenging to consider the system heterogeneity of devices in federated learning. In this paper, we propose AMBLE, which adaptively adjusts the local mini-batch and local epoch size for heterogeneous devices in federated learning, synchronously updating the parameters. Our proposed scheme, AMBLE, can add more computations during the waiting time caused by stragglers and can scale the local learning rate to improve the model convergence rate and accuracy. We confirm that federated learning with AMBLE can be stably trained with a faster convergence speed and higher accuracy than FedAvg and adaptive batch size scheme, in both the non-IID and IID cases.

Keywords: Federated Learning, System Heterogeneity, Local Mini-batch SGD, Federated Averaging

1. Introduction

With the development of digital technologies, such as smartphones, wearable devices, and the Internet of Things (IoT), the amount of data independently created, collected, and stored by individual devices and institutions is rapidly increasing. To discover the relations or phenomena of the collected data, various works are performed, in particular, deep learning is the most favored methodology. Deep learning is suited well to discover hidden meanings from a large amount of data, thereby requires large and diverse sets of data to improve performance. To meet these requirements, it is necessary to collect data generated by various devices or organizations, but privacy issues arise during this process. To ensure data privacy, Google presented federated learning as the next generation of AI learning methods and proposed the federated averaging (FedAvg) algorithm [1] for deep learning models with multiple local devices, where data is decentralized and one centralized server updates the deep learning model. By using federated learning, it is possible to obtain effects similar to processing the entire data at once while independently processing data from various devices and institutions that have data, without collecting distributed local data in the central server.

Federated learning allows training without data leakage in situations where data privacy, such as clinical data in hospitals, must be protected. In modern society, where many tasks are processed through personal smartphones and desktops, federated learning is a very efficient method in which individual devices can be utilized. Nowadays, the advantage of federated learning has been recognized and used in many fields. Gboard [2], the Google keyboard app, uses federated learning to more accurately predict the words and emojis that are expected to be typed on tens of millions of devices. Previously, a new word was recommended only when a user used it a few times; however, now, as a result of federated learning, Gboard learns new words after their use by thousands of users without monitoring what the user is typing.

Meanwhile, there are some challenges to overcome in federated learning. Based on the analysis of Li et al. [3], they defined three challenges of federated learning. (1) *Expensive communication*: federated learning comprises numerous devices, and communication overhead can be much larger because of network bottlenecks. (2) *System heterogeneity*: in federated learning, the system performance of each device may vary depending on the CPU, cellular type, and battery life. (3) *Statistical heterogeneity*: in federated learning, the training data are not identically and independently distributed (non-IID).

To reduce communication overhead in federated learning, existing studies have focused on local mini-batch SGD [1] and compression schemes [4, 5]. In the local mini-batch stochastic gradient descent (SGD), the local value of the gradient is updated iteratively and transferred to the centralized server. Hence, the model not only converges with fewer rounds but also achieves higher accuracy than the case without these methods; this is because the number of communications with the centralized server is reduced. In compression schemes, gradient sparsification [4] and model quantization [5] can reduce the number of gradients and model size to be transferred. Furthermore, recent works [6] that aim to efficiently communicate federated learning have employed edge computing environment, which is

*Corresponding author.
Email address: [redacted]

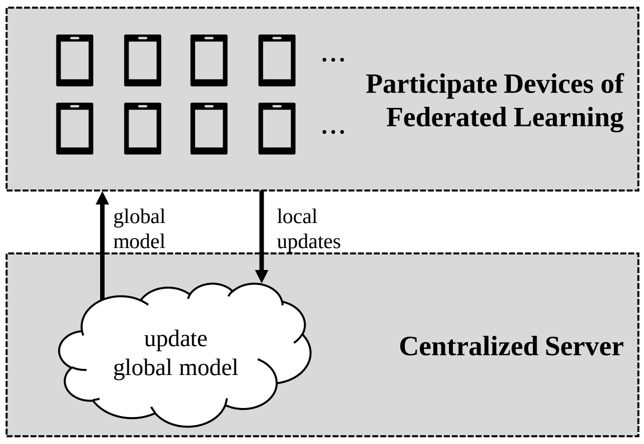

Figure 1: Architecture of federated learning

a kind of decentralized server architecture, to avoid the occurrence of communication overhead in a centralized server architecture. However, the edge server environment requires a large amount of hardware and a diverse type of equipment for data processing. Thus, it is quite costly to construct an environment that satisfies these requirements.

Synchronous updates in federated learning are simple to implement and guarantee high accuracy but are more vulnerable in the face of device heterogeneity [3]. Stragglers can occur for several reasons, including device heterogeneity, connection failures, and imbalanced data distribution among devices. To update the global model in federated learning, asynchronous updating is a scheme used to address the problem of stragglers in heterogeneous devices [7, 8]. However, unlike the synchronous update, the asynchronous synchronization method has a fatal problem, i.e., the stale gradient problem. Because each model obtains the gradient of the model while other models asynchronously update the gradients of the model, the accuracy of the deep learning model compared to the synchronous update approach. For this reason, there have been studies [1, 9, 10, 11, 12] that use a synchronous method in federated learning to update the global model. Moreover, synchronous updating in federated learning has faster convergence rates and higher training accuracy than asynchronous updating [1, 13, 14]. Recently, there have been studies [15, 16] in adaptively adjusting the mini-batch size to minimize waiting time while synchronous updating; by these works, we have been motivated to leverage the adaptive mini-batch approach.

The goal of federated learning is to update the global model — trained using the entire dataset — without sharing data during the training phase. However, it is difficult to create a globally optimized model in federated learning because the training data are not identically and independently distributed (non-IID). Accordingly, most studies on federated learning have focused on developing a model that exhibits good performance, even in non-IIDs. Non-IID refers to the following phenomenon: the data between devices are independent of each other and form different probability distributions. According to FedAvg, based on local mini-batch SGD, multiple local updates can guarantee higher accuracy for non-IID cases and reduce communication overhead by reducing the number of gradient transfers [1]. However, FedAvg uses fixed sizes of local epochs and local mini-batches as well as a fixed local learning rate, without considering heterogeneous devices. As a result, if federated learning is conducted in an environment consisting of heterogeneous devices, stragglers appear during the synchronous update process.

In this paper, we propose a new scheme that adjusts the local mini-batch size and local epoch size adaptively (AMBLE) for federated learning with heterogeneous devices. In AMBLE, we leverage synchronous updates with a local mini-batch SGD to update the global model. In the local mini-batch SGD with federated learning, the communication overhead is reduced and a deep learning model with high accuracy is achieved. If the clients are composed of heterogeneous devices, some straggler devices stall the other devices. Thus, resources are wasted as most devices stall due to stragglers. Hence, we introduce an additional computation process that adjusts the local mini-batch size and local epoch size for each device to compensate for the time that most devices stall. However, in AMBLE, each device has a different local epoch size and local mini-batch size. Moreover, the frequency of data usage varied and the accuracy decreased. To solve this problem, we adopt linear LR scaling [17] to scale the learning rate according to the local mini-batch size and local epoch size. The contributions of our work are summarized as follows:

- We propose AMBLE, which adjusts the local mini-batch size and local epoch size for heterogeneous devices in federated learning. AMBLE can add more computation during the waiting time caused by stragglers and scale the local learning rate to improve the model convergence rate.
- We implement a prototype AMBLE for federated learning using PyTorch. In our experiment, we show the effect of learning rate scaling on adaptive local mini-batch size and local epoch size. In addition, we show that AMBLE performs better than FedAvg in both non-IID and IID cases.

The remainder of this paper is organized as follows. In Section 2, we explain the background of federated learning and explain the related works of local mini-batch SGD and adaptive batch size approach. Then, our proposed scheme AMBLE for federated learning with heterogeneous devices are introduced in Section 3. The experimental results and evaluation of our proposed scheme are presented in Section 4. The conclusion of our work and future work are discussed in Section 5.

2. Background and Related Work

2.1. Federated Learning

Federated learning uses data stored in mobile devices for machine learning model training without leaking. The central server — i.e., the cloud — aggregates all gradients to update a global model and sends the model back to the mobile device for inference. Traditional distributed deep learning has a relatively

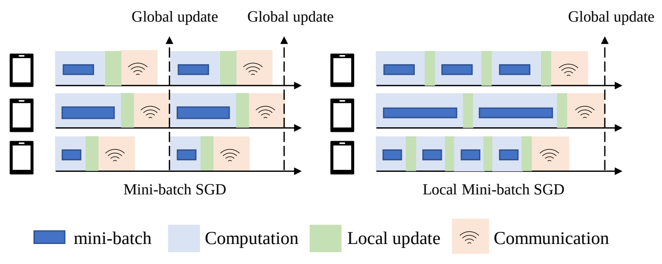

Figure 2: Mini-batch SGD vs. Local Mini-batch SGD [3]

stable network environment, a lot of storage space and a higher computational power of nodes than the federated learning environment [18, 19]. In contrast, federated learning is subject to limitations, such as the sensitivity of personal data, unstable network environment, low computing power, limited battery, and small storage space [20, 21, 22]. Existing distributed deep learning has been researched for optimizing communication, increasing computational speed, and maintaining accuracy; however, studies addressing data requiring security and privacy guarantees, such as federated learning, remain insufficient.

Mobile devices generate a large amount of data through interaction with users and most data are sensitive to personal information. Photos, user's location, chat history, passwords, etc. are data that require personal information protection and which the user does not want to be leaked from their device. This is the same even if the anonymity of the data is guaranteed; i.e., there is a limitation in that data cannot be collected on a central server, such as a data center, which makes it difficult to apply data to traditional distributed deep learning and machine learning. To utilize these data, federated learning applies code to data instead of applying data to code. Rather than sending the data to the central server to the mobile device, the model was trained with the data of each mobile device. Because the data of each device is not transmitted to the server, it is stored only in the device and threats to the data from external attackers are reduced, compared to training by transmitting the data to the central server, ensuring security and privacy [23, 24].

Because mobile devices participating in federated learning are in a wireless communication environment, the network is unstable and there may be between thousands and millions of participants. The central server selects only mobile devices with limited conditions as participants for stable model training. Participating in the training while the user is using the device affects the user's device use performance; hence, only devices connected via Wi-Fi communication can participate in the training in an environment where the device is not being used and is charging [21].

In the protocol of federated learning, devices that satisfy the above-mentioned charging, Wi-Fi connection, and idle status notify the server that they are ready to register as participants. After selecting the optimal number of devices, the server transmits the data structure, such as graph information for calculation and tasks to be performed on the selected participants. Devices that are not selected are notified to reconnect the next time. The selected devices are calculated using the global model received from the server and the local data stored in the device. When the calculation is complete, the update is transferred to the central server, which reflects the update to the global model. The process up to this point is one round and the participant and server remain connected for the duration of the round. If drop-out occurs due to communication failure during training, the server ignores the participant and proceeds with the round. The federated learning protocol is designed such that no failure occurs even if the round proceeds, ignoring the participant who failed the connection [9].

2.2. Local Mini-batch SGD

Deep learning has a higher complexity than traditional modeling techniques, such as polynomial regression, and the training process involves solving high-level non-convex optimization problems. Therefore, the convergence reached is always different depending on the initial conditions. The optimization method using the entire training data related to the convergence process is referred to as full-batch gradient descent (BGD) and the optimization method of sampling some of training data every epoch is referred to as SGD. If the mini-batch size is one, it is called SGD; otherwise, it is referred to as mini-batch SGD. The mini-batch SGD has a lower risk of falling into the local optimal than BGD and is more advantageous for parallel processing than SGD. In addition, memory usage is less than BGD because only a part of the datasets is used, not the entire dataset. Because of these advantages, mini-batch SGD is most commonly used when training deep learning models on large-scale datasets.

In FedAvg [1], the authors proposed two learning methods for federated learning: federated SGD (FedSGD) and federated averaging (FedAvg). FedSGD applied the existing SGD method of training models locally to federated learning. There is a hyperparameter C required by FedSGD that refers to the batch size based on the devices. For example, if C=1, all participating devices are used and, if C=0.5, only 50% of participating devices are used. Each local updated weight is updated by calculating the average value, as shown in the following equation (g: gradient, w: weight, η: learning rate, n: the number of data, K: the number of devices):

$$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k \tag{1}$$

FedAvg is an algorithm based on a local mini-batch SGD update to overcome the limitations of FedSGD in federated learning. In this method, each device transmits the gradient to the server after repeatedly performing training for a certain number of times, E. By dividing each device into batch size and learning to give a mini-batch effect, the convergence time of the global model can be shortened. Local mini-batch SGD for federated learning can guarantee higher accuracy for non-IID datasets. case in either the IID case and it can reduce the communication overhead that reduces the number of communication rounds.

In FedAvg [1], the authors evaluated the performance of FedAvg compared to FedSGD as a baseline. FedAvg, which re-

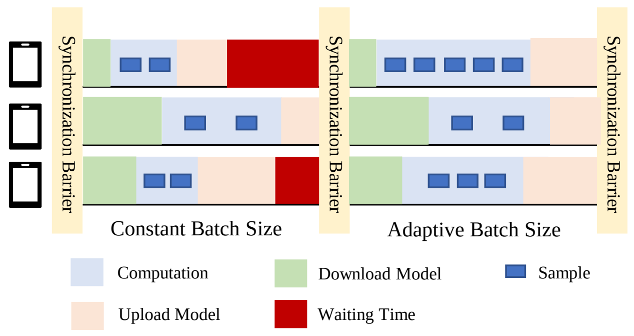

Figure 3: Adaptive Batch Size [15]

peatedly updates locally, demonstrated better performance on both the IID and non-IID datasets, compared with the FedSGD. However, FedAvg does not consider heterogeneous devices. In other words, local epochs and mini-batch sizes were applied equally to all devices. FedAvg causes stragglers during the synchronous update process when the selected devices consist of heterogeneous devices. In our work, we focus on heterogeneous devices and synchronous updates in federated learning. Therefore, we propose an adaptive local mini-batch SGD for heterogeneous devices based on the FedAvg algorithm.

In addition to the local update method, there are related works — such as compression schemes and model quantization — to reduce communication overhead in federated learning. Gradient sparsification [4, 25] can reduce the number of gradients transferred to the central server in each round. In addition, the model quantization [5, 26] approach can reduce the size of the global model communicated in each round. Wang et al. [25] proposed a convex optimization formula — gradient sparsification — to minimize the coding length of stochastic gradients to reduce communication overhead in distributed deep learning. In FedGKT [26], the authors proposed FedGKT to solve a problem in which it is difficult to use the large convolutional neural network (CNN) model as a global model in federated learning. FedGKT uses a knowledge distillation algorithm [27] to reduce the cost of computation at the edge and increase the computational cost of the central server. The central server trains a large CNN model to rate model A. At the edge, the knowledge of model A is locally distilled to train a relatively lightweight model. This recursive structure is executed for each round. To relieve the network bottleneck on the central server, decentralized training, such as edge computing, has also been proposed [6] to reduce communication overhead in the central server. However, edge servers require more local hardware and various equipment for data processing and hardware for processing computing processes are required, which also increases costs. Various pieces of equipment for data processing and hardware for processing computing processes are required, which also increases costs. In our work, we only considered the local update method and did not leverage additional communication efficiency methods. However, the compression scheme, model quantization, and decentralized method can be used in the proposed structure.

Table 1: Notations used in this paper

Notation	Definition
K	The number of devices
k	The index of the kth device, where $k \in [1, K]$
C	The fraction of devices
B_k	Local mini-batch size of device k
E_k	Local epoch size of device k
η_k	Learning rate of device k
P_k	Set of indexes of data points on device k
N_{round}	The number of round
d_k	The kth device
n_k	The number of data that the kth device owns
δ	The computation time of model updating in each iteration

2.3. *Adaptive Batch Size Approach*

In distributed deep learning, to distribute data, one global model is copied to several devices and each is trained. The gradients of the local models are updated through the divided data and the updates of these parameters are performed separately owing to the divided data. The synchronization of these parameters is an important topic in distributed deep learning. The parameter synchronization method of deep learning models is largely divided into synchronous and asynchronous methods. Because the synchronous method updates the model by adding the gradients learned by each model at each step, the parameters of each model are always kept the same. However, a disadvantage is that the parameters can only be updated after the gradient calculation of all equipment is finished; thus, it adapts to the learning speed of the slowest component. In the asynchronous update method, multiple replicated models update their learned gradients as soon as they learn the data assigned to them. The advantage of the asynchronous method is that the amount of data processed per time is large. There may be deviations in the learning time of each device because the next learning can be performed by updating the parameters as each device learns quickly.

When synchronous updating of parameters is performed in a distributed environment, such as federated learning, a straggler may occur when it consists of heterogeneous devices. The stragglers delay the learning time and cause waiting times for other devices. This reduces the overall learning performance. Asynchronous updating is a scheme for addressing the straggler problem in heterogeneous devices [7, 8]. However, unlike the synchronous update, the asynchronous synchronization method has a fatal problem, i.e., the stale gradient problem. In the asynchronous update, local models update the gradients, whereas each model obtains the gradient of the model. Therefore, optimization does not occur efficiently and the efficiency of using the data is reduced. This means that using an asynchronous update to train the model is less accurate than a synchronous update. Thus, in our study, we focus on synchronous updates in federated learning.

Some related studies have used an adaptive batch size scheme to address stragglers in synchronous updates. Ma et al. [15]

Algorithm 1 Federated learning process with AMBLE

```
procedure StartFL()
    $w_0$.initialize(rand());
    AdaptiveSet();
    for $t$ in $[1, N_{round}]$ do
        $X \leftarrow C \cdot K$;
        $R_t \leftarrow \{d_{x_0}, d_{x_1}, ..., d_{x_{X-1}}\}$; // X devices are
            randomly selected
        for $k$ in $[1, K]$ do
            DownloadModel();
            LocalUpdate($k$,$w_t$);
        end for
        GlobalUpdate($k$,$t$);
    end for
end procedure
procedure LocalUpdate(k,$w_t$)
    // Each element of $\beta$ stands for a batch
    $\beta \leftarrow$ (split $P_k$ by size of $B_k$);
    for $i$ in $[1, E_k]$ do
        for $b \in \beta$ do
            $w^k \leftarrow w^k - \eta_k \nabla F_k(w^k)$;
        end for
    end for
end procedure
$w^k_{t+1} \leftarrow w^k$;
SendGradient($w^k_{t+1}$);
procedure GlobalUpdate(k,t)
    $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w^k_{t+1}$;
end procedure
```

Algorithm 2 Adaptive hyper-parameters set in AMBLE

```
procedure AdaptiveSet()
    for $k$ in $[1, K]$ parallel do
        profilingDevices($k$);
        $T \leftarrow \max(T_k, T)$;
        $T \leftarrow T - \alpha - \gamma$;
        $E_k \leftarrow$ baseline;
        $E_k^{temp} \leftarrow E_k$;
        $B_k \leftarrow$ baseline;
        $B_k^{temp} \leftarrow B_k$;
        $\eta_k \leftarrow$ baseline;
        $E_k \leftarrow \frac{T}{\tau_{(k)}}$;
        $\eta_k \leftarrow \eta_k \frac{E_k^{temp}}{E_k}$;
        $B_k \leftarrow \frac{\delta \cdot E_k \cdot n_k \cdot c_k}{T \cdot c_k - M \cdot s \cdot E_k \cdot n_k}$;
        $\eta_k \leftarrow \eta_k \frac{B_k^{temp}}{B_k}$;
    end for
end procedure
```

proposed an algorithm that adaptively adjusts the local mini-batch size for heterogeneous devices, which minimizes the waiting time caused by stragglers. They also determined the relationship between mini-batch size and learning rate and formulated a learning rate scaling rule in terms of batch size. However, they did not consider the communication overhead of federated learning and non-IID datasets. If the non-IID dataset is trained, the proposed method cannot be expected to improve performance. Our work differs from this in that we have proposed a scheme that also considers non-IID datasets.

BOA [16] proposed a batch-orchestration algorithm (BOA) that solves the learning time delay caused by stragglers through batch size adjustment used in distributed learning. Hence, a computational execution time prediction model was first established through a GPU learning profiling for the deep learning model to be trained and, based on this, sophisticated control was processed in terms of batch size. In addition, they proposed worker set tuning to improve the learning speed through GPU worker set management. However, they only focused on the local mini-batch size and did not consider learning rate scaling. If learning rate scaling is not performed, a biased gradient value is derived for each iteration as the batch size gap of each worker increases; thus, the learning convergence is slowed and the accuracy is lowered. In our work, we leverage linear LR scaling [17] to balance the learning of different local mini-batch sizes for each device.

3. AMBLE for Heterogeneous Devices

In federated learning, there is a possibility that each device is configured differently in computational performance or network connection. This system heterogeneity causes stragglers due to the process of synchronously updating the global model in the parameter server. Thus, the total training time is affected by the training time of the slowest device among the stragglers, resulting in a delay. To solve the problem caused by these stragglers, we adaptively adjust the local mini-batch size and local epoch size (AMBLE) during the device's waiting time to perform more computations to affect learning.

3.1. Federated Learning Process with AMBLE

We present a federated learning process (algorithm 1) based on the FedAvg algorithm [1]. To reduce communication overhead, we leveraged FedAvg's local mini-batch SGD to reduce the amount of communication. Unlike FedAvg, our proposed scheme, AMBLE, considers heterogeneous devices in the synchronous update process. FedAvg using a fixed local mini-batch size and local epoch size can cause stragglers caused by heterogeneous devices that are inefficient because it causes the waiting time of other devices. Thus, resources are wasted because most devices stall because of stragglers. AMBLE adaptively adjusts the local mini-batch size and local epoch size of each device, such that one round ends within the slowest device training time T. Devices other than the slowest scale to larger local epoch sizes and smaller mini-batch sizes. Hence, AMBLE can add more computation in one round than FedAvg. Table 1 lists the key notations used in this study.

Table 2: Specification of the devices

Device	CPU	GPU	Memory	OS
TX1	Quad-Core ARM Cortex-A57 MPCore	256-core NVIDIA Maxwell TM GPU	4GB	Ubuntu 16.04.1 LTS
TX1	Quad-Core ARM Cortex-A57 MPCore	256-core NVIDIA Maxwell TM GPU	4GB	Ubuntu 16.04.1 LTS
TX2	Quad-Core ARM Cortex-A57 MPCore	256-core NVIDIA Pasca TM GPU	8GB	Ubuntu 16.04.1 LTS
Desktop	Intel Core TM i7-8700 CPU 3.20GHz	NVIDIA GeForce RTX 2080	32GB	Ubuntu 20.04.1 LTS
Desktop	Intel Core TM i7-8700 CPU 3.20GHz	NVIDIA GeForce RTX 3080	32GB	Ubuntu 20.04.1 LTS

In algorithm 1, the amount of computation is controlled by C, E_k and B_k. C is the fraction of devices that perform computations in each round. E_k is the local epoch size of device k over its local dataset in each round. B_k is the local mini-batch size used for updating device k updates. When federated learning starts, the hyper-parameters such as local mini-batch size and local epoch size are adjusted adaptively by profiling devices. Then, the fraction of devices C in each round is selected, if C=1, all participating devices are used, and if C=0.5, only 50% of the participating devices are used. After selecting devices that participate, each device downloads the global model from the centralized server. Then each device computes the gradient of the loss for all data held by these devices. In the local update, $w^k \leftarrow w^k - \eta_k \nabla F_k(w^k)$ for E_k times and then uploads w^k_{t+1} to the centralized server. When all devices have finished uploading gradients, the centralized server averages transferred gradients to perform a global model update $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w^k_{t+1}$. This process is repeated in each round.

3.2. Adaptive Hyper-parameters Set in AMBLE

AMBLE adaptively sets the local mini-batch size and local epoch size adaptively for federated learning. Unlike the adaptive batch size approach, AMBLE adaptively adjusts the local epoch size first and then fine-tunes it through the local mini-batch size. Also, since the adaptive batch approach cannot adjust the mini-batch size to less than 1, waiting time may exist even after adjustment. However, since AMBLE's local epoch size is not limited, waiting time can be further minimized. If the local mini-batch size and local epoch size are different for each device, the gradients may become imbalanced; thus, we scale the learning rate based on the baseline mini-batch size and local epoch size.

Algorithm 2 adaptively sets the hyper-parameters of each device, including the local mini-batch size, local epoch size and local learning rate in AMBLE. First, we profiled device k to derive c_k which is the computation capacity of device k (CPU/GPU frequencies) and s, which is the required number of CPU/GPU cycles to compute a gradient of one iteration. In addition, it can derive the time of downloading global model from the server α and uploading local model to the server γ. Then, we set the training time T based on the slowest device and $T \leftarrow T - \alpha - \gamma$ to derive only the time for local updates. Therefore, we can derive the total local update time $\tau(k)$ using the following formula:

$$\tau(k) = E_k \cdot \frac{n_k}{B_k} \cdot \left(\frac{M \cdot B_k \cdot s}{c_k} + \delta\right) \tag{2}$$

From the above formula, M refers to the model size, and it can be seen that the factors affecting the total time of local update in each round are the local epoch size E_k, the local mini-batch size B_k and the computation time of model updating δ, which occur every iteration. Since the number of iterations is determined for each local epoch according to the size of B_k, this affects how many times δ computes. Thus, we can set the local epoch size E_k, dividing T by $\tau(k)$. Since the epoch size must be a positive integer, it is set as the divided quotient.

$$B_k \leftarrow \frac{\delta \cdot E_k \cdot n_k \cdot c_k}{T \cdot c_k - M \cdot s \cdot E_k \cdot n_k} \tag{3}$$

We set the local mini-batch size to fit the total local update time as T. Based on the equation for deriving $\tau(k)$, the following equation for deriving B_k according to T was established: after the local mini-batch size and local epoch size were set, linear LR scaling [17] was used for learning rate scaling according to the mini-batch size and local epoch size. If learning rate scaling is not applied, overfitting may occur for specific data, and the gradient value may become unbalanced, resulting in lower test accuracy.

$$\eta_k \leftarrow \eta_k \frac{E_k^{temp}}{E_k}, \eta_k \leftarrow \eta_k \frac{B_k^{temp}}{B_k} \tag{4}$$

4. Experiment and Evaluation

4.1. Experiment Setup

We implemented a prototype of federated learning with AMBLE and all models using PyTorch [28]. For the implementation we used Python 3.8.5, Pytorch 1.8.1 and torchvision 0.9.1. As an experimental environment, we configured five heterogeneous devices with two NVIDIA Jetson TX1, one NVIDIA Jetson TX2 and two desktop PC, which is consisted of one RTX 3080 and one RTX 2080. In our experiment, we chose the datasets and models based on FedAvg [1]. The two models are for the MNIST task [29], i.e., a CNN that has two convolution layers (5 × 5, 32 channels, 64 channels, 2 × 2 max pooling), two fully connected layers (512 units), and the ReLu activation function, which we refer to as the CNN-MNIST and a multilayer perceptron (MLP), with two hidden layers (200 units) and the ReLu activation function, which we refer to as MLP-MNIST. For the IID and non-IID settings, we partition the MNIST data over the devices. The data is shuffled and an IID case is then constructed, which splits the data into five devices, each receiving 12000 examples. For the non-IID case, we create shards of size 6000, which sort the data by digit labels, dividing it into 10 and assigning two shards to each of the five

Table 3: Parameter sets for FedAvg, AMB (baseline) and AMBLE (our scheme) on MNIST and CIFAR-10.

	MNIST			CIFAR-10		
	FedAvg	AMB	AMBLE	FedAvg	AMB	AMBLE
Number of devices	5	5	5	5	5	5
Fraction of devices	1	1	1	1	1	1
Mini-batch size	10	Adaptive	Adaptive	50	Adaptive	Adaptive
Local epoch size	5	5	Adaptive	5	5	Adaptive
Learning rate	0.01	LR scaling	LR scaling	0.02	LR scaling	LR scaling
Optimizer	SGD	SGD	SGD	Adam	Adam	Adam

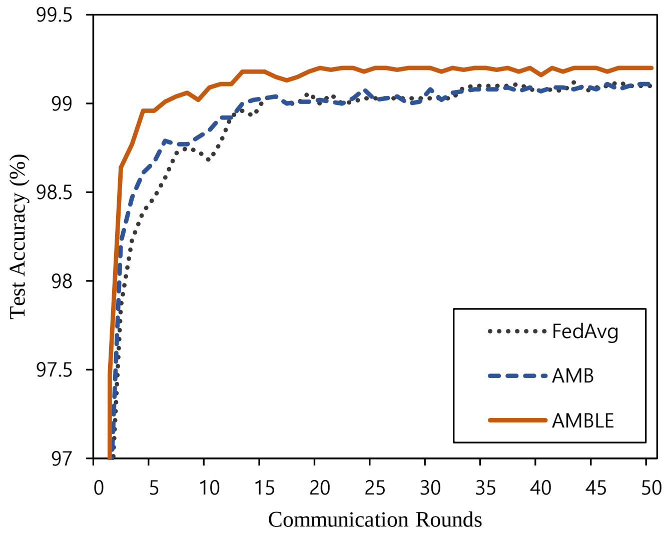

Figure 4: Test accuracy curves of CNN-MNIST (IID case)

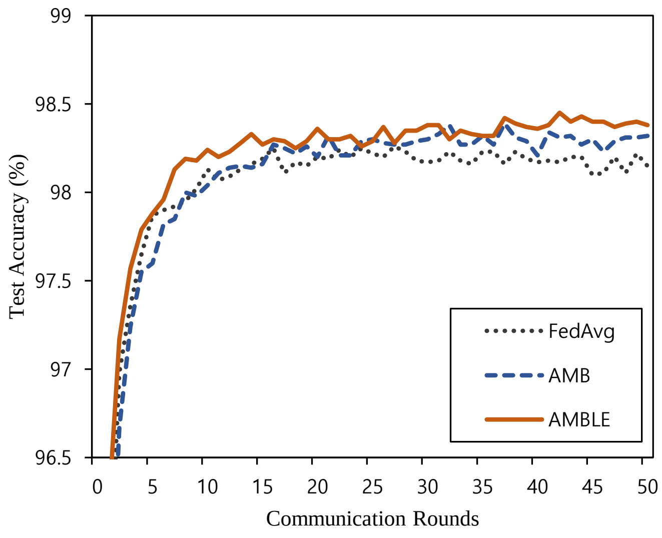

Figure 5: Test accuracy curves of MLP-MNIST (IID case)

devices. For CIFAR-10 dataset [30], we use CNN model from the Pytorch tutorial, which consists of two convolutional layers and two fully connected layers. The CIFAR-10 dataset consists of 10 classes of 32×32 images with three RGB channaels. There are 50000 training dataset and 10,000 testing dataset. For the IID and non-IID settings, we partition the CIFAR-10 dataset over the devices. The data is shuffled and an IID case is then constructed, which splits the data into five devices, each receiving 10000 examples. For the non-IID case, we create shards of size 5000, which sort the data by class labels, dividing it into 10 and assigning two shards to each of the five devices.

Our experimental baseline is FedAvg, which sets a fixed local mini-batch and local epoch size with a fixed learning rate. We also compare with adaptive mini-batch scheme (AMB), which sets a fixed local epoch size and adaptive mini-batch size with linear LR scaling to minimize waiting time caused by stragglers. Table 3 presents the parameter set for FedAvg, AMB and AMBLE on MNIST and CIFAR-10. For MNIST dataset, FedAvg trains with a fixed learning rate of 0.01 for all devices whereas AMB and AMBLE train with a learning rate that applies linear LR scaling for each device. We set the fraction of the devices to 1 and use SGD optimizer [31] with 0.5 momentum. For CIFAR-10 dataset, FedAvg trains with a fixed learning rate of 0.15 for all devices whereas AMB and AMBLE train with a learning rate that applies linear LR scaling for each device. We set the fraction of the devices to 1 and use Adam optimizer [32] with 1e-4 weight decay.

4.2. Experiment Result

4.2.1. MNIST with IID Case

We conducted an experiment to demonstrate the performance of the proposed scheme, AMBLE. Table 4 shows the number of communication rounds to reach a target test accuracy for FedAvg, AMB, and AMBLE with IID and non-IID setting. In the Table 4, the numbers outside and within the brackets indicate the number of communication rounds, and the corresponding speedups over FedAvg, respectively. For CNN-MNIST with IID case, the baseline FedAvg was reached a target test accuracy 99% by 15 rounds. Compare to FedAvg, AMBLE was reached the target test accuracy by 6 rounds, which is 2.5× faster. It can be seen that AMBLE can reach the target accuracy at a faster rate than AMB which is 1.15× faster than FedAvg. For MLP-MNIST with IID case, the baseline FedAvg was reached a target test accuracy 98% by 9 rounds. Compare to FedAvg, AMBLE was reached the target test accuracy by 8

Table 4: Number of communication rounds to reach a target accuracy for FedAvg, AMB, and AMBLE. The numbers outside and within the brackets indicate the number of communication rounds, and the corresponding speedups over FedAvg, respectively.

	IID			Non-IID		
	CNN-MNIST	MLP-MNIST	CNN-CIFAR	CNN-MNIST	MLP-MNIST	CNN-CIFAR
Target Accuracy	99%	98%	60%	94%	93%	58%
FedAvg (baseline)	15 (1.00×)	9 (1.00×)	78 (1.00×)	57 (1.00×)	72 (1.00×)	47 (1.00×)
AMB	13 (1.15×)	8 (1.13×)	55 (1.42×)	46 (1.24×)	50 (1.44×)	28 (1.68×)
AMBLE	6 (2.50×)	6 (1.50×)	12 (6.50×)	17 (3.35×)	27 (2.67×)	12 (3.92×)

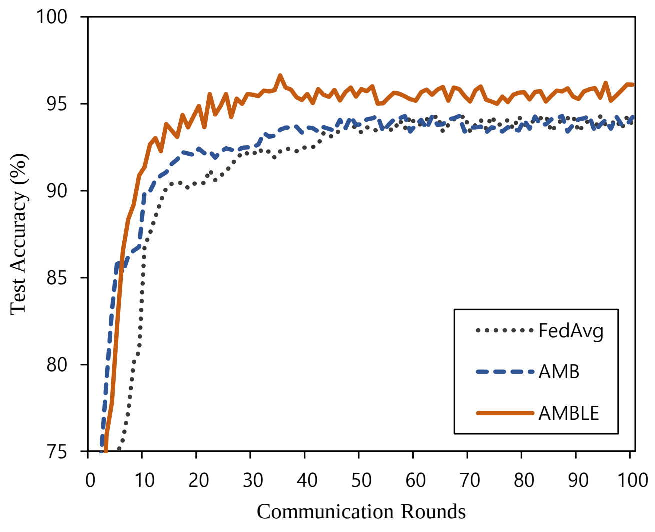

Figure 6: Test accuracy curves of CNN-MNIST (Non-IID case)

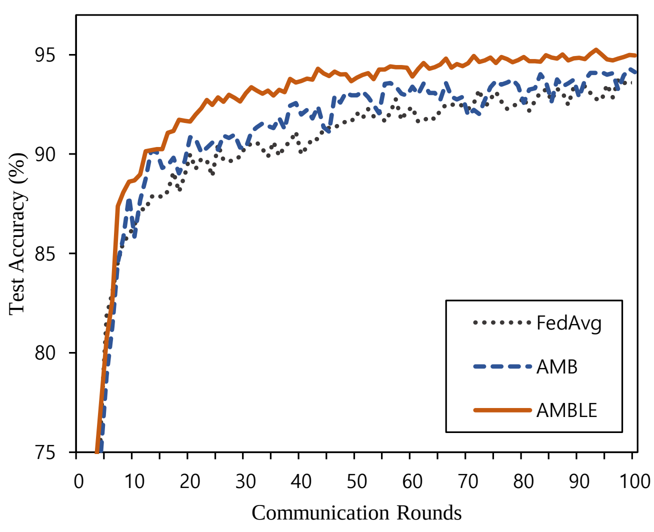

Figure 7: Test accuracy curves of MLP-MNIST (Non-IID case)

rounds, which is 1.5× faster. It can be seen that AMBLE can reach the target accuracy at a faster rate than AMB which is 1.13× faster than FedAvg. We demonstrated through this experiment that AMBLE can reduce the total training time delay caused by the stragglers.

Figures 4 and 5 show the test accuracy curves of CNN-MNIST and MLP-MNIST for the IID case, respectively. For CNN-MNIST with IID case, it can be seen that FedAvg with a fixed mini-batch and local epoch size for 50 rounds converges to about 91% test accuracy. Compare to FedAvg, AMBLE converges to 99.2% and AMB converges to 99.1%, just like FedAvg. For MLP-MNIST with IID case, it can be seen that FedAvg for 50 rounds converges to about 98.2% test accuracy. Compare to FedAvg, AMBLE converges to 98.4% and AMB converges to 98.3%. For both the MLP model and the CNN model, AMBLE converges with higher test accuracy than AMB and FedAvg on the MNIST dataset in the IID case. Although there is no big difference in the performance of AMBLE in the test accuracy, it can prove the improved performance of AMBLE in reaching the target test accuracy. This shows that the total training time delay problem caused by stragglers can be mitigated.

4.2.2. MNIST with Non-IID Case

We also evaluated the performance of the AMBLE in the non-IID case. For CNN model, the baseline FedAvg was reached a target test accuracy 94% by 57 rounds. Compare to FedAvg, AMBLE was reached the target test accuracy by 17 rounds, which is 3.35× faster. It can be seen that AMBLE can reach the target accuracy at a faster rate than AMB which is 1.24× faster than FedAvg. For MLP model, the baseline FedAvg was reached a target test accuracy 93% by 72 rounds. Compare to FedAvg, AMBLE was reached the target test accuracy by 27 rounds, which is 2.67× faster. It can be seen that AMBLE can reach the target accuracy at a faster rate than AMB which is 1.44× faster than FedAvg.

Figures 6 and 7 show the test accuracy curves of CNN-MNIST and MLP-MNIST for the non-IID case, respectively. For CNN-MNIST with non-IID case, it can be seen that FedAvg for 100 rounds converges to about 94.1% test accuracy. Compare to FedAvg, AMBLE converges to 96.1% and AMB converges to 94.2%. It can be seen that AMBLE converges with 2% higher accuracy than FedAvg, and 1.9% higher than AMB. For MLP-MNIST with non-IID case, it can be seen that FedAvg for 100 rounds converges to about 93.6% test accuracy. Compare to FedAvg, AMBLE converges to 95.2% and AMB converges to 94.2%. It can be seen that AMBLE converges

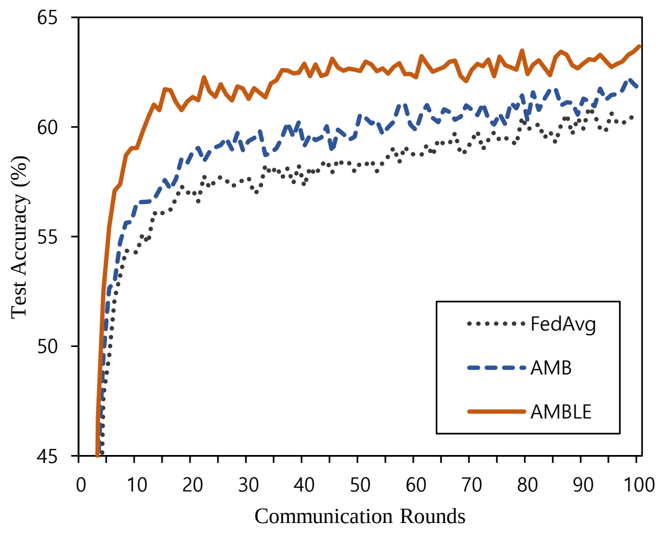

Figure 8: Test accuracy curves of CNN-CIFAR (IID case)

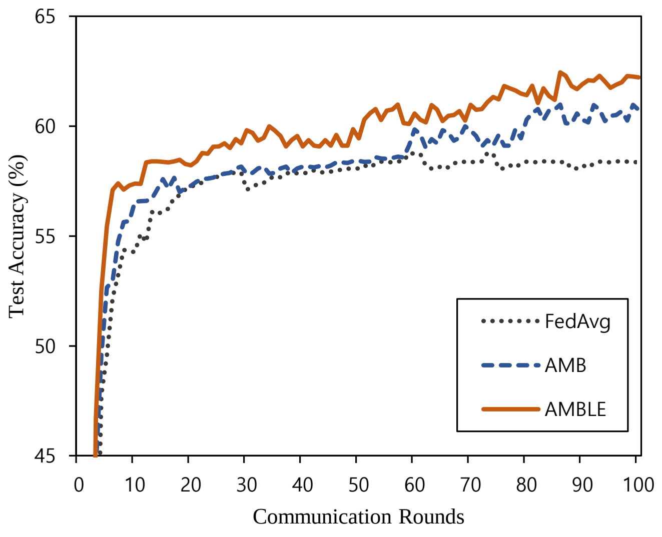

Figure 9: Test accuracy curves of CNN-CIFAR (Non-IID case)

with 1.8% higher accuracy than FedAvg, and 1% higher than AMB. Through experiments, we proved that the test accuracy of AMBLE, the proposed scheme, showed significantly higher performance than FedAvg and AMB in the Non-IID case than in the IID case.

4.2.3. CIFAR-10 with IID Case

In addition to the MNIST dataset, we conducted additional experiments using the CIFAR-10 dataset with the CNN model to prove that AMBLE's performance improved over FedAvg and AMB. In the CIFAR-10 experiment, the lightweight MLP model used by MNIST took too long to train and did not train properly, so we experimented with only the CNN model. We evaluated the performance of the AMBLE in the non-IID case. The baseline FedAvg was reached a target test accuracy 60% by 78 rounds. Compare to FedAvg, AMBLE was reached the target test accuracy by 12 rounds, which is 6.5× faster. It can be seen that AMBLE can reach the target accuracy at a faster rate than AMB which is 1.42× faster than FedAvg.

Figures 8 shows the test accuracy curves of CNN-CIFAR for the IID case. It can be seen that FedAvg for 100 rounds converges to about 60.3% test accuracy. Compare to FedAvg, AMBLE converges to 63.7% and AMB converges to 61.5%. It can be seen that AMBLE converges with 3.4% higher accuracy than FedAvg, and 2.2% higher than AMB. Through experiments, we proved that AMBLE showed better performance than FedAvg and AMB not only in MNIST but also in CIFAR-10.

4.2.4. CIFAR-10 with Non-IID Case

We also evaluated the performance of the AMBLE with CIFAR-10 in the non-IID case. The baseline FedAvg was reached a target test accuracy 58% by 47 rounds. Compare to FedAvg, AMBLE was reached the target test accuracy by 12 rounds, which is 3.92× faster. It can be seen that AMBLE can reach the target accuracy at a faster rate than AMB which is 1.68× faster than FedAvg.

Figures 9 shows the test accuracy curves of CNN-CIFAR for the non-IID case. It can be seen that FedAvg for 100 rounds converges to about 58.4% test accuracy. Compare to FedAvg, AMBLE converges to 62.2% and AMB converges to 60.7%. It can be seen that AMBLE converges with 3.8% higher accuracy than FedAvg, and 1.5% higher than AMB. Through experiments, we proved that AMBLE showed better performance than FedAvg and AMB not only in CIFAR with IID case but also in CIFAR with non-IID case.

5. Conclusion and Future Works

In federated learning, participate devices can be configured with heterogeneous devices. Heterogeneity of devices can occur stragglers in federated learning. To solve the stragglers problem, we proposed a new AMBLE scheme that adaptively adjusts the local mini-batch size and local epoch size for heterogeneous devices in federated learning, which synchronously updates the deep learning model. Our experiments showed that AMBLE has higher accuracy and faster convergence speed than FedAvg and AMB with both non-IID and IID cases.

In future works, we will conduct a study on an algorithm that can achieve higher accuracy than existing techniques by efficiently adjusting the learning rate decay for each round in federated learning. In addition, we will conduct research to develop a new federated learning platform in which our proposed scheme is applied in real smartphone devices. For our new federated learning platform, we will also conduct research to present a new architecture of federated learning with serverless parameter server. Serverless computing has the advantage of reducing server management because the physical environment provided by the cloud service provide is used, direct server management is not required. In federated learning, updating gradients and the global model are event-driven. Thus, serverless computing can be used for the parameter server of federated learning. We will evaluate the billing system when our future federated learning platform with serverless computing is more cost-effective than leveraging existing legacy cloud computing. In serverless computing, cold start time problem can be occurred. There has been studied reusing serverless instances to reduce cold start time and keep the instances warm. However, reusing serverless instances is expensive and ignores the event-driven characteristics of serverless computing. Therefore, future work should be conducted to reduce the cold start time with cost-efficiency.

Acknowledgment

This research was jointly supported by the Basic Science Research Program (2021R1F1A1062779) of the National Research Foundation of Korea (NRF) funded by the Ministry of Education, and the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2021-2018-0-01431) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation). We would like to thank Editage (www.editage.co.kr) for English language editing.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 1273–1282.

[2] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, D. Ramage, Federated learning for mobile keyboard prediction, arXiv preprint arXiv:1811.03604 (2018).

[3] T. Li, A. K. Sahu, A. Talwalkar, V. Smith, Federated learning: Challenges, methods, and future directions, IEEE Signal Processing Magazine 37 (3) (2020) 50–60.

[4] P. Han, S. Wang, K. K. Leung, Adaptive gradient sparsification for efficient federated learning: An online learning approach, arXiv preprint arXiv:2001.04756 (2020).

[5] M. M. Amiri, D. Gunduz, S. R. Kulkarni, H. V. Poor, Federated learning with quantized global model updates, arXiv preprint arXiv:2006.10672 (2020).

[6] L. Liu, J. Zhang, S. Song, K. B. Letaief, Client-edge-cloud hierarchical federated learning, in: ICC 2020-2020 IEEE International Conference on Communications (ICC), IEEE, 2020, pp. 1–6.

[7] W. Dai, A. Kumar, J. Wei, Q. Ho, G. Gibson, E. Xing, High-performance distributed ml at scale through parameter server consistency models, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 29, 2015, pp. 79–87.

[8] Y. Lu, X. Huang, Y. Dai, S. Maharjan, Y. Zhang, Differentially private asynchronous federated learning for mobile edge computing in urban informatics, IEEE Transactions on Industrial Informatics 16 (3) (2019) 2134–2143.

[9] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, H. B. McMahan, et al., Towards federated learning at scale: System design, arXiv preprint arXiv:1902.01046 (2019).

[10] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, K. Chan, When edge meets learning: Adaptive control for resource-constrained distributed machine learning, in: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, 2018, pp. 63–71.

[11] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, K. Chan, Adaptive federated learning in resource constrained edge computing systems, IEEE Journal on Selected Areas in Communications 37 (6) (2019) 1205–1221.

[12] R. Carli, A. Chiuso, L. Schenato, S. Zampieri, A pi consensus controller for networked clocks synchronization, IFAC Proceedings Volumes 41 (2) (2008) 10289–10294.

[13] C. Chen, W. Wang, B. Li, Round-robin synchronization: Mitigating communication bottlenecks in parameter servers, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 532–540.

[14] J. Chen, X. Pan, R. Monga, S. Bengio, R. Jozefowicz, Revisiting distributed synchronous sgd, arXiv preprint arXiv:1604.00981 (2016).

[15] Z. Ma, Y. Xu, H. Xu, Z. Meng, L. Huang, Y. Xue, Adaptive batch size for federated learning in resource-constrained edge computing, IEEE Transactions on Mobile Computing (2021).

[16] E. Yang, S.-H. Kim, T.-W. Kim, M. Jeon, S. Park, C.-H. Youn, An adaptive batch-orchestration algorithm for the heterogeneous gpu cluster environment in distributed deep learning system, in: 2018 IEEE International Conference on Big Data and Smart Computing (BigComp), IEEE, 2018, pp. 725–728.

[17] A. Krizhevsky, One weird trick for parallelizing convolutional neural networks, arXiv preprint arXiv:1404.5997 (2014).

[18] J. Chen, X. Pan, R. Monga, S. Bengio, R. Jozefowicz, Revisiting distributed synchronous sgd, arXiv preprint arXiv:1604.00981 (2016).

[19] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, et al., Large scale distributed deep networks, in: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, 2012, p. 1223–1231.

[20] J. Konečnỳ, H. B. McMahan, D. Ramage, P. Richtárik, Federated optimization: Distributed machine learning for on-device intelligence, arXiv preprint arXiv:1610.02527 (2016).

[21] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, D. Bacon, Federated learning: Strategies for improving communication efficiency, arXiv preprint arXiv:1610.05492 (2016).

[22] J. Konečnỳ, B. McMahan, D. Ramage, Federated optimization: Distributed optimization beyond the datacenter, arXiv preprint arXiv:1511.03575 (2015).

[23] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, K. Seth, Practical secure aggregation for federated learning on user-held data, arXiv preprint arXiv:1611.04482 (2016).

[24] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, K. Seth, Practical secure aggregation for privacy-preserving machine learning, in: proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 1175–1191.

[25] J. Wangni, J. Wang, J. Liu, T. Zhang, Gradient sparsification for communication-efficient distributed optimization, arXiv preprint arXiv:1710.09854 (2017).

[26] C. He, M. Annavaram, S. Avestimehr, Group knowledge transfer: Federated learning of large cnns at the edge, Advances in Neural Information Processing Systems 33 (2020).

[27] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, arXiv preprint arXiv:1503.02531 (2015).

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, arXiv preprint arXiv:1912.01703 (2019).

[29] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[30] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images (2009).

[31] S. Ruder, An overview of gradient descent optimization algorithms, arXiv preprint arXiv:1609.04747 (2016).

[32] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

Author Biography

[redacted]

Declaration of interests

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

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