Communication-Efficient Federated Deep Learning With Layerwise Asynchronous Model Update and Temporally Weighted Aggregation

Yang Chen, Xiaoyan Sun, *Member, IEEE*, and Yaochu Jin, *Fellow, IEEE*

***Abstract*— Federated learning obtains a central model on the server by aggregating models trained locally on clients. As a result, federated learning does not require clients to upload their data to the server, thereby preserving the data privacy of the clients. One challenge in federated learning is to reduce the client–server communication since the end devices typically have very limited communication bandwidth. This article presents an enhanced federated learning technique by proposing an asynchronous learning strategy on the clients and a temporally weighted aggregation of the local models on the server. In the asynchronous learning strategy, different layers of the deep neural networks (DNNs) are categorized into shallow and deep layers, and the parameters of the deep layers are updated less frequently than those of the shallow layers. Furthermore, a temporally weighted aggregation strategy is introduced on the server to make use of the previously trained local models, thereby enhancing the accuracy and convergence of the central model. The proposed algorithm is empirically on two data sets with different DNNs. Our results demonstrate that the proposed asynchronous federated deep learning outperforms the baseline algorithm both in terms of communication cost and model accuracy.**

***Index Terms*— Aggregation, asynchronous learning, deep neural network (DNN), federated learning, temporally weighted aggregation.**

I. Introduction

SMARTPHONES, wearable gadgets, and distributed wireless sensors usually generate huge volumes of privacy-sensitive data. In many cases, service providers are interested in mining information from these data to provide personalized services, for example, to make more relevant recommendations to clients. However, the clients are usually not willing to allow the service provider to access the data for privacy reasons.

Federated learning is a recently proposed privacy-preserving machine learning framework [1]. The main idea is to train local models on the clients, send the model parameters to the server, and then aggregate the local models on the server. Since all local models are trained upon data that are locally stored in clients, data privacy can be preserved. The whole process of the typical federated learning is divided into communication rounds, in which the local models on the clients are trained on the local data sets. For the kth client, where $k \in S$, and S refers to the participating subset of m clients, its training samples are denoted as $\mathcal{P}_k$, and the trained local model is represented by the model parameter vector ω^k. In each communication round, only models of the clients belonging to the subset S will download the parameters of the central model from the server and use them as the initial values of the local models. Once the local training is completed, the participating clients send the updated parameters back to the server. Consequently, the central model can be updated by aggregating the updated local models, i.e., $\omega = Agg(\omega^k)$ [1]–[3].

In this setting, the local models of each client can be any type of machine learning models, which can be chosen according to the task to be accomplished. In most existing work on federated learning [1], deep neural networks (DNNs), e.g., long short-term memory (LSTM), are employed to conduct text-word/text-character prediction tasks. In recent years, DNNs have been successfully applied to many complex problem-solving, including text classification, image classification, and speech recognition [4]–[6]. Therefore, DNNs are widely adopted as the local model in federated learning, and the stochastic gradient descent (SGD) is the most popular learning algorithm for training the local models.

As aforementioned, one communication round includes parameter download (on clients), local training (on clients), trained parameter upload (on clients), and model aggregation (on the server). Such a framework appears to be similar to a distributed machine learning algorithm [7]–[12]. In federated learning, however, only the models' parameters are uploaded and downloaded between the clients and server, and the data of local clients are not uploaded to the server or exchanged between the clients. Accordingly, the data privacy of each client can be preserved.

Compared with other machine learning paradigms, federated learning is subject to the following challenges [1], [13].

1) *Unbalanced Data:* The data amount on different clients may be highly imbalanced because there are light and heavy users.
2) *Non-IID Data:* The data on the clients may be strongly non-independent and identically distributed (IID) because of the different preferences of different users. As a result,

Manuscript received March 7, 2019; revised July 7, 2019; accepted November 1, 2019. This work was supported by the National Natural Science Foundation of China under Grant 61876184 and Grant 61473298. *(Yang Chen and Xiaoyan Sun are co-first authors.) (Corresponding author: Yaochu Jin.)*

Y. Chen is with the School of Information and Control Engineering, China University of Mining and Technology, Xuzhou 221116, China, and also with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: fedora.cy@gmail.com).

X. Sun is with the School of Information and Control Engineering, China University of Mining and Technology, Xuzhou 221116, China (e-mail: xysun78@hotmail.com).

Y. Jin is with the Department of Computer Science, University of Surrey, Guildford GU2 7XH, U.K. (e-mail: yaochu.jin@surrey.ac.uk).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2019.2953131

local data sets are not able to represent the overall data distribution, and the local distributions are different from each other, too. The IID assumption in distributed learning that training data are distributed over local clients uniformly at random [14] usually does not hold in federated learning.

3) *Massively Distributed Data:* The number of clients is large. For example, the clients may be mobile phone users [2], which can be enormous.

4) *Unreliable Participating Clients:* It is common that a large portion of participating clients is often offline or on unreliable connections. Again in case the clients are mobile phone users, their communication state can vary a lot and, thus, cannot ensure their participation in each round of learning [1].

Apart from the above-mentioned challenges, the total communication cost is often used as an overall performance indicator of federated learning due to the limited bandwidth and battery capacity of mobile phones. Of course, like other learning algorithms, the learning accuracy, which is mainly determined by the local training and the aggregation strategy, is also of great importance. Accordingly, the motivation of our article is to reduce the communication cost and improve the accuracy of the central model, assuming that DNNs are used as the local learning models. Inspired by interesting observations in fine-tuning of DNNs [15], a layerwise asynchronous update strategy for local model updating and aggregation is proposed to improve the communication efficiency in each round. Note that in the field of evolutionary optimization, evolutionary algorithms have also adopted asynchronous strategies for enhancing computational efficiency, e.g., parallel asynchronous particle swarm optimization (PAPSO) [16]. Concretely, motivations behind our layerwise asynchronous strategy and the one adopted by PAPSO, i.e., higher communication efficiency versus computational efficiency, make the essential difference. Driven by these aforementioned motivations, our proposed algorithm addresses the parameters from shallow and deeps layers in an asynchronous manner, while PAPSO updates particle positions and velocities in parallel and asynchronously.

The main contributions of this article are as follows. First, an asynchronous strategy that aggregates and updates the parameters in the shallow and deep layers of DNNs at different frequencies is proposed to reduce the number of parameters to be communicated between the server and clients. Second, a temporally weighted aggregation strategy is suggested to more efficiently integrate information of the previously trained local models in model aggregation to enhance the learning performance.

The remainder of this article is organized as follows. In Section II, related work is briefly reviewed. The detail of the proposed algorithm, especially the asynchronous strategy, the temporally weighted aggregation, and the overall framework are described in Section III. Section IV presents the experimental results and discussions. Finally, conclusions are drawn in Section V.

II. Related Work

Konečný *et al.* [2] developed the first framework of federated learning and also experimentally proved that existing machine learning algorithms are not suited for this setting. Konečný *et al.* [3] proposed two ways to reduce the uplink communication costs, i.e., structured updates and sketched updates, using data compression/reconstruction techniques. A more recent version of federated learning, federated averaging (FedAVG), was reported in [1], which was developed for obtaining a central prediction model of Google's Gboard app and can be embedded in a mobile phone to protect the user's privacy. The pseudocode of FedAVG is provided in Algorithm 1 [1].

Algorithm 1 FedAVG [1]

```
1: function ServerExecution                         ▷ Run on the server
2:     initialize $w_0$
3:     for each round $t = 1, 2, \ldots$ do
4:         $m \leftarrow \max(C \cdot K, 1)$
5:         $S_t \leftarrow$ (random set of $m$ clients)
6:         for each client $k \in S_t$ in parallel do
7:             $w_{t+1}^k \leftarrow$ ClientUpdate$(k, w_t)$
8:         end for
9:         $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
10:    end for
11: end function
12:
13: function ClientUpdate$(k, w)$                   ▷ Run on client $k$
14:    $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size $B$)
15:    for each local epoch $i$ from 1 to $E$ do
16:        for batch $b \in \mathcal{B}$ do
17:            $w \leftarrow w - \eta \bigtriangledown \ell(w; b))$
18:        end for
19:    end for
20:    return $w$ to server
21: end function
```

In the following, we briefly explain the main components of FedAVG.

1) **Server Execution** consists of the *initialization* and *communication rounds*.

   a) *Initialization:* Line 2 initializes parameter ω_0.

   b) *Communication Rounds:* Line 4 obtains m, the number of participating clients; K indicates the number of local clients, and C corresponds to the fraction of participating clients per round, according to which line 5 randomly selects participating subset S_t. In lines 6–8, subfunction $ClientUpdate$ is called in parallel to get ω_{t+1}^k. Line 9 executes aggregation to update ω_{t+1}.

2) **Client Update:** The subfunction gets k and ω. B and E are the local mini-batch size and the number of local epochs, respectively. η is the learning rate. Line 14 splits data into batches; where $\mathcal{P}_k$ corresponds to the set of indices of the data points on client k, with $n_k = |\mathcal{P}_k|$. Lines 15–19 execute the local SGD on batches. Line 20 returns the local parameters.

The equation in line 9, $\omega_{t+1} \leftarrow \sum_{k=1}^{K}(n_k/n) * \omega_{t+1}^k$, described the aggregation strategy, where n_k is the sample size of the kth client and n is the total number of training samples. ω_{t+1}^k comes from client k in round $t+1$; however, it is not always updated in the current communication round.

For clients that do not participate, their models remain unchanged until they are chosen to participate. In model aggregation, the parameters uploaded from clients in the current round and those in previous ones contribute equally to the central model. In FedAVG, parameters in the shallow and deep layers are always synchronously updated. The parameters uploaded from the clients not participating in the current round are the same as in the previous rounds.

Apart from reducing communication costs, other studies of federated learning have focused on protocol security. For example, to tackle differential attacks, Gayer *et al.* [17] proposed an algorithm incorporating a preserving mechanism into federated learning for client-sided differential privacy. Bonawitz *et al.* [18] designed an efficient and robust transfer protocol for secure aggregation of high-dimensional data.

While privacy preserving and reduction of communication costs are two important aspects to take into account in federated learning, the main concern remains to be the enhancement of learning performance of the central model on all data. The loss function of federated learning is defined as $F(\omega) = \sum_{k=1}^{K} (n_k/n) f_k(\omega)$, where $f_k(\omega)$ is the loss function of the kth client model. Clearly, the performance of federated learning heavily depends on the model aggregation strategy.

Not much work has been reported on reducing the communication cost by reducing the number of parameters to be uploaded and downloaded between clients and the server except for some recent work reported most recently [19]. In this article, we present a layerwise asynchronous model learning model that updates only part of the model parameters to reduce communication and a temporally weighted aggregation strategy that gives a higher weight on more recent models in aggregation to enhance learning performance.

III. Layerwise Asynchronous Model Update and Temporally Weighted Aggregation

A. Layerwise Asynchronous Model Update

The most intrinsic requirement for decreasing the communication cost of federated learning is to upload/download as little data as possible without deteriorating the performance of the central model. To this end, we present in this article a layerwise asynchronous model update strategy that updates only part of the local model parameters to reduce the amount of data to be uploaded or downloaded.

Our idea was primarily inspired by the following interesting observations made in fine-tuning DNNs [15], [20].

1) Shallow layers in a DNN learn the general features that are applicable to different tasks and data sets, meaning that a relatively small part of the parameters in DNNs (those in the shallow layers) represent features general to different data sets.
2) By contrast, deep layers in a DNN learn *ad hoc* features related to specific data sets, and a large number of parameters focus on learning features in specific data.

These above-mentioned observations indicate that the relatively smaller number of parameters in the shallow layers is more pivotal for the performance of the central model in federated learning. Accordingly, parameters of the shallow layers should be updated more frequently than those parameters in the deep layers. Therefore, the parameters in the shallow and deep layers in the models can be updated asynchronously, thereby reducing the amount of data to be sent between the server and clients. We term this layerwise asynchronous model update.

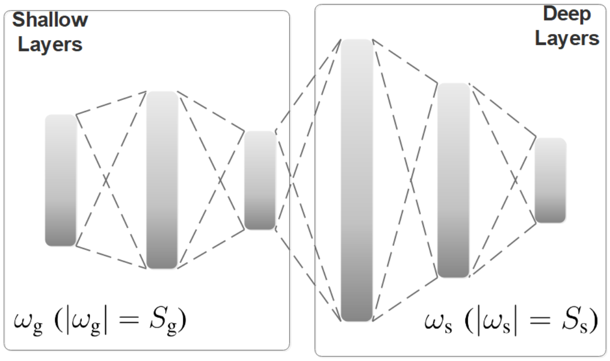

Shallow Layers
Deep Layers
ω_g $(|\omega_g| = S_g)$
ω_s $(|\omega_s| = S_s)$

Fig. 1. Illustration of shallow and deep layers of a DNN.

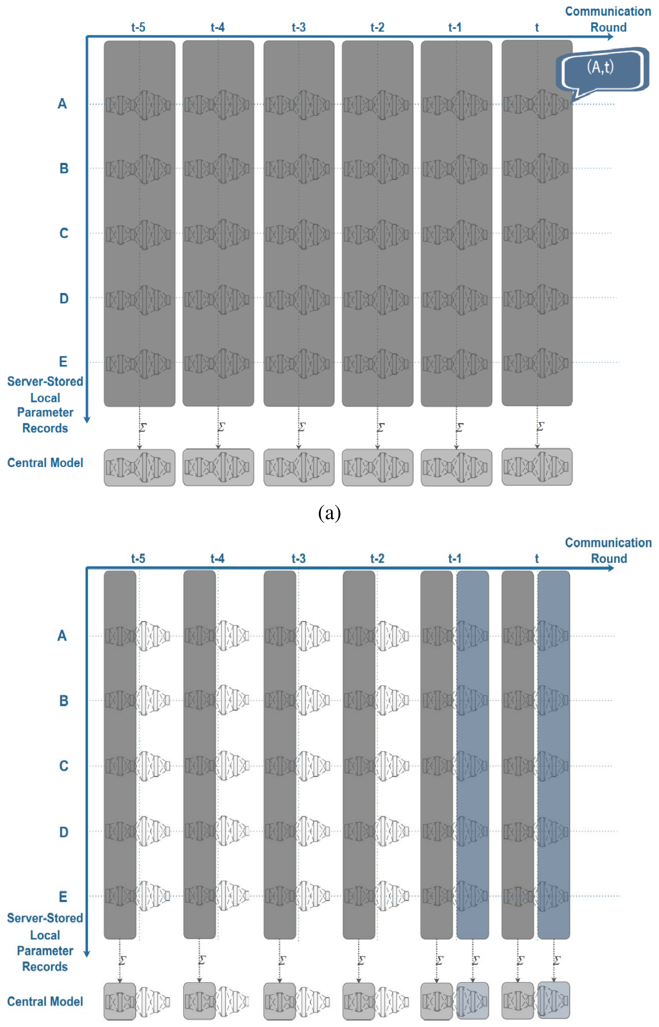
The DNN employed for the local models on the clients is shown in Fig. 1. As shown in Fig. 1, it can be separated into shallow layers for learning general features and deep layers for learning specific-feature layers features, which are denoted as ω_g and ω_s, respectively. The sizes of ω_g and ω_s are denoted as S_g and S_s, respectively, and typically, $S_g \ll S_s$. In the proposed asynchronous learning strategy, ω_g will be updated and uploaded/downloaded more frequently than ω_s. Assume that the whole federated process is divided into loops, and each loop has T rounds of model updates. In each loop, ω_g will be updated and communicated in every round, while ω_s will be updated and communicated in only fe rounds, where $fe < T$. As a result, the number of reduced parameters to be exchanged between the server and clients, i.e., the reduced communication cost, will be $fe{*}S_s$. This way, the communication cost can be significantly reduced, since S_s is usually very large in DNNs.

An example is given in Fig. 2 to show the asynchronous learning strategy. The abscissa and ordinate denote the communication round and the local client, respectively. In this example, there are five local devices, i.e., $\{A, B, C, D, E\}$, and a server. Point (A, t) indicates that client A is participating in updating the global model in round t.

Fig. 2(a) provides an illustration of a conventional synchronous aggregation strategy, where both ω_g and ω_s are uploaded/downloaded in each round. The gray rounded rectangles in the bottom represent the aggregation, in which both shallow and deep layers participate in all rounds. By contrast, Fig. 2(b) shows the proposed asynchronous learning strategy. In this example, there are six computing rounds ($t-5$, $t-4, \ldots, t$) in the loop, and the parameters of deep layers are exchanged in rounds $t-1$ and t only. As a result, the number of reduced parameters to be communicated is $2/3 * S_s$.

B. Temporally Weighted Aggregation

In federated learning, the aggregation strategy (line 9 in Algorithm 1) usually weights the importance of each local



(a)

(b)

Fig. 2. Parameter exchange. (a) Synchronous model update strategy. (b) Asynchronous model update strategy.

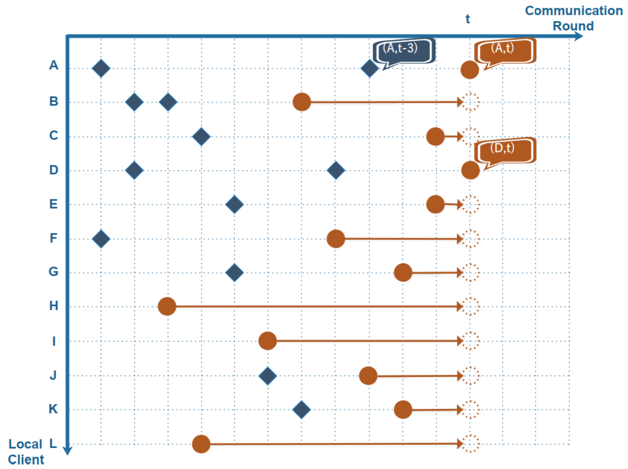


Fig. 3. Conventional aggregation strategy.

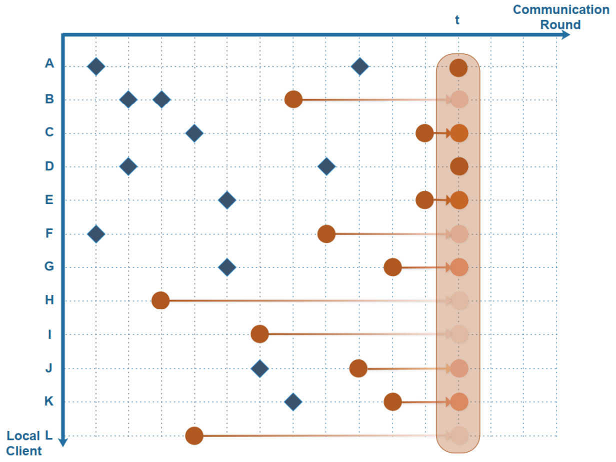


Fig. 4. Illustration of the temporally weighted aggregation.

model based on the size of the training data on the corresponding client. That is, the larger n_k is, the more the local on client k will contribute to the central model, as shown in Fig. 3. In that example, the blue diamonds represent the previously trained local models, which do not take part in the $t+1$th update of the central model, and only the orange dots are the most recently updated local model on each client and will participate in the current round of aggregation. All participating local models (orange dots in Fig. 3) will be weighted by their data size only regardless the computing round in which these models are updated. In other words, local models updated in round $t-p$ are as important as those updated in round t, which might not be reasonable.

In federated learning, however, the training data on each participating client are changing in each round and, therefore, models that are more recently updated should have a larger weight in the aggregation. Accordingly, the following model aggregation method taking into account of timeliness of the local models is proposed.

$$\omega_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} * (e/2)^{-(t-\text{timestamp}^k)} * \omega^k \tag{1}$$

where e is the natural logarithm used to depict the time effect, t means the current round, and timestamp^k is the round in which the newest ω^k was updated. The proposed temporally weighted aggregation is shown in Fig. 4. Similar to the setting used in Fig. 3, all clients participating in the aggregation are denoted by an orange dot, while others are represented by blue diamonds. The darker the color is, the larger the weight of this local model will have in the aggregation.

C. Framework

The framework of the proposed federated learning with a layerwise asynchronous model update and temporally weighted aggregation is shown in Fig. 5. A detailed description of the main components will be given in Section III-D.

D. Temporally Weighted Asynchronous Federated Learning

The pseudocode for the two main components of the proposed temporally weighted aggregation asynchronous (ASTW)

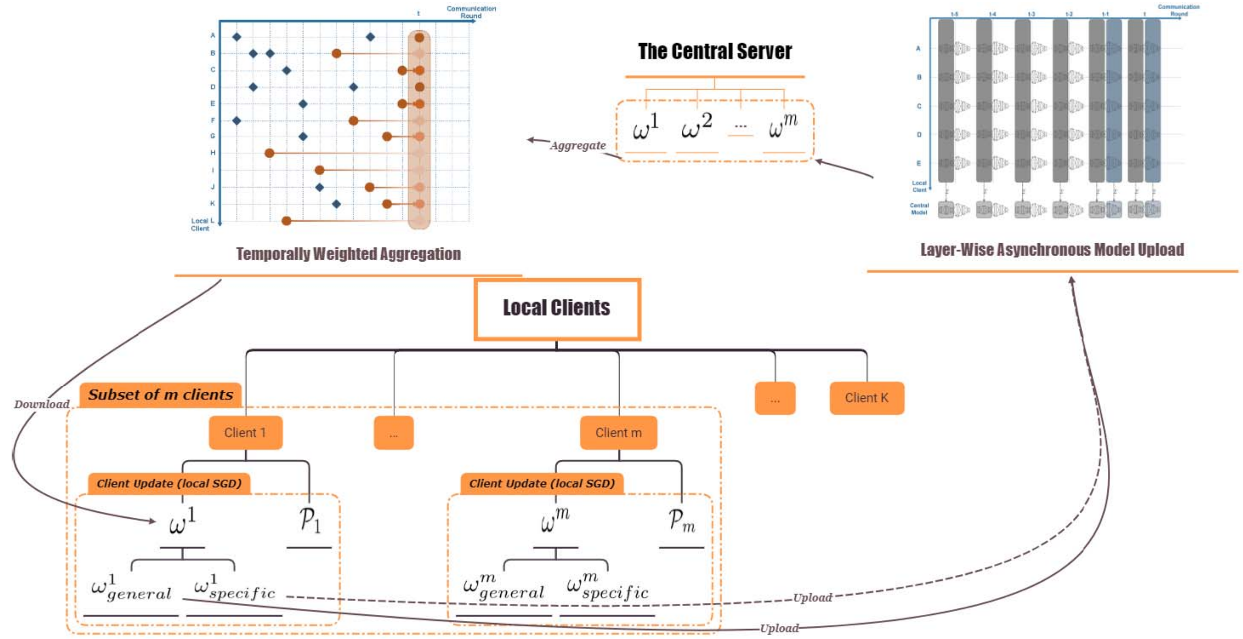


Fig. 5. Federated learning with a layerwise asynchronous model update and temporally weighted aggregation.

federated learning ASTW_FedAVG, one implemented on the server, and the other on the clients is given in Algorithms 2 and 3, respectively.

The part to be implemented on the server consists of an initialization step followed by a number of communication rounds. In initialization (Algorithm 2, lines 2–6), the central model ω_0, timestamps $\text{timestamp}_\text{g}$, and $\text{timestamp}_\text{s}$ are initialized. Timestamps are stored and to be used to weight the timeliness of corresponding parameters in aggregation.

The training process is divided into loops. Lines 8–12 in Algorithm 2 set $flag$ to be true in the last 1/freq rounds in each loop. Assume there are $rounds\_in\_loop$ rounds in each loop. Lines 13 and 14 randomly select a participating subset S_t of the clients according to C, which is the fraction of participating clients per round. In lines 15–24, subfunction $ClientUpdate$ is called in parallel to get $\omega^k/\omega^k_\text{g}$, and the corresponding timestamps are updated. $flag$ specified whether all layers or the shallow layers only will be updated and communicated. Then in lines 25–28, the aggregation is performed to update ω_g. Note that compared with 1, a parameter a is introduced into the weighting function in line 25 or line 27 to examine the influence of different weightings in the experiments. In this article, a is set to e or $e/2$.

The implementation of the local model update (Algorithm 3) is controlled by three parameters, k, ω, and $flag$, where k is the index of the selected client, $flag$ indicates whether all layers or the shallow layers will be updated. B and E denote the local mini-batch size and the local epoch, respectively. In Algorithm 3, line 2 splits data into batches, whereas lines 3–7 set all layers or shallow layers of the local model to be downloaded according to $flag$. In lines 8–12, local SGD is performed. Lines 13–17 return local parameters.

IV. Experimental Results and Analyses

A. Experimental Design

We perform a set of experiments using two popular DNN models, i.e., convolution neural networks (CNNs) for image processing and LSTM for human activity recognition (HAR), respectively, to empirically examine the learning performance and communication cost of the proposed ASTW_FedAVG. A CNN with two stacked convolution layers and one fully connected layer [21], [22] is applied on the Modified National Institute of Standards and Technology (MNIST) handwritten digit recognition data set, and an LSTM with two stacked LSTM layers and one fully connected layer is chosen to accomplish the HAR task [23], [24]. Both the MNIST and the HAR data sets are adapted to test the performance of the proposed federated learning framework for different real-world scenarios, e.g., non-IID distribution, unbalanced amount, and massively decentralized data sets, which will be discussed in detail in Section IV-B.

FedAVG [1] is selected as the baseline algorithm since it is the state-of-the-art approach. The proposed ASTW_FedAVG is also compared with two variants, namely, temporally weighted federated learning (TWFL) that adopts temporally weighted aggregation without the layerwise asynchronous model update, and AS_FedAVG that employs the layerwise asynchronous model update without using temporally weighted aggregation. Thus, four algorithms, i.e., FedAVG, ASTW_FedAVG, TW_FedAVG, and AS_FedAVG will be compared in the following experiments.

The most important parameters in the proposed algorithm are listed in Table I. The parameter $freq$ controls the frequency for updating and exchanging the parameters in the deep layers ω_s between the server and the local clients in a loop.

Algorithm 2 Server Component of ASTW_FedAVG

```
function SERVEREXECUTION                    ▷ Run on the server
    initialize $\omega_0$
    for each client $k \in \{1, 2, \dots, K\}$ do
        $timestamp_{\mathrm{g}}^{k} \leftarrow 0$
        $timestamp_{\mathrm{s}}^{k} \leftarrow 0$
    end for
    for each round $t = 1, 2, \dots$ do
        if $t$ mod $rounds\_in\_loop \in set_{ES}$ then§
            $flag \leftarrow$ True
        else
            $flag \leftarrow$ False
        end if
        $m \leftarrow \max(C * K, 1)$
        $S_t \leftarrow$ (random set of $m$ clients)
        for each client $k \in S_t$ in parallel do
            if $flag$ then
                $\omega^k \leftarrow$ ClientUpdate$(k, \omega_t, flag)$
                $timestamp_{\mathrm{g}}^{k} \leftarrow t$
                $timestamp_{\mathrm{s}}^{k} \leftarrow t$
            else
                $\omega_{\mathrm{g}}^{k} \leftarrow$ ClientUpdate$(k, \omega_{g,t}, flag)$
                $timestamp_{\mathrm{g}}^{k} \leftarrow t$
            end if
        end for
        $\omega_{g,t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} * f_{\mathrm{g}}(t,k) * \omega_{\mathrm{g}}^{k}$ †
        if $flag$ then
            $\omega_{s,t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} * f_{\mathrm{s}}(t,k) * \omega_{\mathrm{s}}^{k}$ ‡
        end if
    end for
end function
    § $rounds\_in\_loop = 15$ and $set_{ES} = \{11, 12, 13, 14, 0\}$
    † $f_{\mathrm{g}}(t,k) = a^{-(t - timestamp_{\mathrm{g}}^{k})}$
    ‡ $f_{\mathrm{s}}(t,k) = a^{-(t - timestamp_{\mathrm{s}}^{k})}$
```

Algorithm 3 Client Component of ASTW_FedAVG

```
function CLIENTUPDATE$(k, w, flag)$        ▷ Run on client $k$
    $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size $B$)
    if $flag$ then
        $\omega \leftarrow \omega$
    else
        $\omega_{\mathrm{s}} \leftarrow \omega$
    end if
    for each local epoch $i$ from 1 to $E$ do
        for batch $b \in \mathcal{B}$ do
            $\omega \leftarrow \omega - \eta * \nabla \ell(w; b)$
        end for
    end for
    if $flag$ then
        return $\omega$ to server
    else
        return $\omega_{\mathrm{s}}$ to server
    end if
end function
```

TABLE I

PARAMETER SETTINGS

Notion	Parameter Range
$freq$	{3/15, 5/15*, 7/15}
a	{e/2*, e}
K	{10, 20*}
m	{1, 2*}

* Default setting

For instance, $freq = 5/15$ means that only in the last 5 of the 15 rounds, the parameters in the deep layers ω_{s} will be uploaded/downloaded between the server and clients. a is a parameter for adjusting the time effect in model aggregation. K and m are environmental parameters controlling the scale or complexity of the experiments, K denotes the number of local clients, and m is the number of participating clients per round.

B. Settings on Data Sets

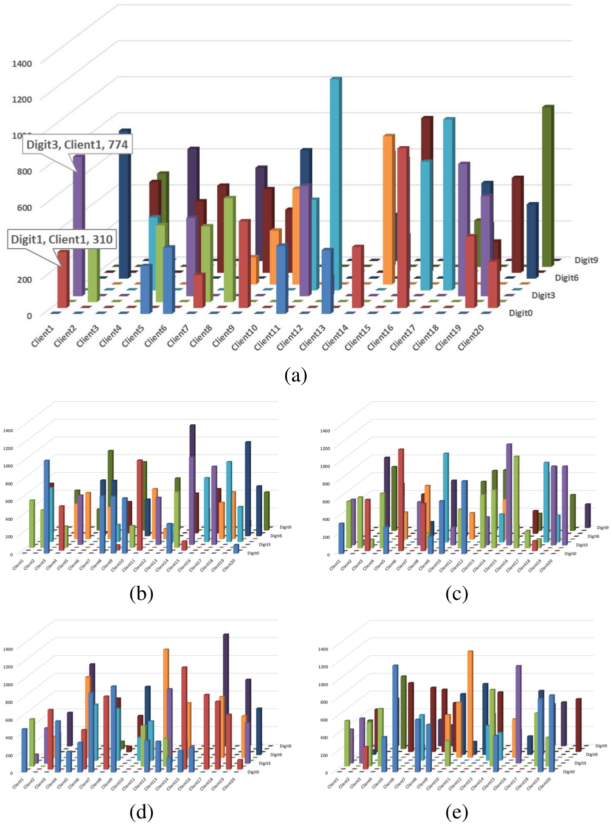
As discussed in Section I, the federated learning framework has its particular challenges, such as non-IID, unbalanced, and massively decentralized data sets. Therefore, the data sets used in our experiments should be designed to reflect these challenges. The generation of the client data set is described in detail in Algorithm 4, which is controlled by four parameters $Labels$, N_c, S_{min}, and S_{max}, where N_c controls the number of classes in each local data set, S_{min} and S_{max} specify the range of the size of the local data, and $Labels$ indicates the names of classes involved in the corresponding tasks.

Algorithm 4 Generation of Local Data Sets.

```
Input: $Labels$, $N_C$, $S_{min}$, and $S_{max}$
Output: non-IID and unbalanced local dataset $\mathcal{P}_k$
$classes \leftarrow$ Choices($Labels$,$N_C$)
$L \leftarrow$ Len($Labels$)
$weights \leftarrow$ Zeros($L$)
$\mathcal{P} \leftarrow$ Zeros($L$)
for each $class \in classes$ do
    $weights_{class} \leftarrow$ Random(0,1)
end for
$sum \leftarrow \sum_{class=1}^{L} weights_{class}$
$num \leftarrow$ Random$(S_{min}, S_{max})$
for each $class \in classes$ do
    $\mathcal{P}_{class} \leftarrow \frac{weights_{class}}{sum} \times num$
end for
$\mathcal{P}_k \leftarrow \mathcal{P}$
```

1) Handwritten Digit Recognition Using CNN: The MNIST data set has ten different kinds of digits and one digit is a 28 × 28 pixel gray-scale image. To partition the data over local clients, we first sort them by their digit labels and divide them into ten shards, i.e., 0, 1, …, 9. Then, Algorithm 4 is performed to compute $\mathcal{P}_k$, which is the kth client's partition coefficient corresponding to these shards. In this task, $Labels = \{0, 1, 2, \dots, 9\}$; N_c is randomly chosen from $\{2, 3\}$, given $K = 20$, $S_{min} = 1000$ and $S_{max} = 1600$. For the sake of easy analyses, five partitions/local data sets, namely,

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



(a)

(b)

(c)

(d)

(e)

Fig. 6. 3-D column charts of pregenerated data sets. (a) 1@MNIST. (b) 2@MNIST. (c) 3@MNIST. (d) 4@MNIST. (e) 5@MNIST.

TABLE II
PARAMETERS SETTINGS FOR THE CNN

Layer	Shapes
conv2d_1	$5 \times 5 \times 1 \times 32$
conv2d_1	32
conv2d_2	$5 \times 5 \times 32 \times 64$
conv2d_2	64
dense_1	1024×512
dense_1	512
dense_2	512×10
dense_2	10

1@MNIST, 2@MNIST, ..., 5@MNIST are predefined. Their corresponding 3-D column charts are plotted in Fig. 6.

The architecture of the CNN used for the MNIST task has two 5×5 convolution layers (the first one has 32 channels and the other has 64) and 2×2 max-pooling layers. Then, a fully connected layer with 512 units and ReLu activation is followed. An output layer is a softmax unit. The parameter settings for CNN are listed in Table II.

2) Human Activity Recognition Using LSTM: In the HAR data set, each data is a sequence of images with a label out of six activities. Similar operations are applied on the HAR data set to divide the data set over local clients. Here, $Labels = \{0, 1, 2, \ldots, 5\}$; N_c is randomly chosen from $\{2, 3\}$, given $K = 20$, $S_{min} = 250$ and $S_{max} = 500$.

The architecture of the LSTM used in this article has two 5×5 LSTM layers (the first one with $cell\_size = 20$ and $time\_steps = 128$ and the other with $cell\_size = 10$ and the same $time\_steps$), a fully connected layer with 128 units and the ReLu activation, and a softmax output layer. The corresponding parameters of the LSTM is given in Table III.

TABLE III
PARAMETER SETTINGS FOR THE LSTM

Layer	Shapes
lstm_1	9×100
lstm_1	25×100
lstm_1	100
lstm_2	25×100
lstm_2	25×100
lstm_2	100
dense_1	25×256
dense_1	256
dense_2	256×6
dense_2	6

TABLE IV
EXPERIMENTAL RESULTS ON $freq$

freq	Accuracy*	Round**
3/15	96.72% (0.0037)	115.74 (32.19)
5/15	97.08% (0.0037)	87.82 (20.03)
7/15	97.12% (0.0046)	95.43 (14.64)

* AVG (STDEV) of best overall accuracy within 200 rounds.
** ASTW_FedAVG on 1@MNIST reaches the accuracy 95.0% within 200 rounds.

C. Results and Analysis

Two sets of experiments are performed in this subsection. The first set of experiments examines the influence of the most important parameters, $freq$, a, K, and m, on the performance of the two strategies using the CNN for the 1@MNIST data set. The second set of experiments compares four algorithms in terms of the communication cost and learning accuracy using the LSTM on the HAR data set since the HAR is believed to be more challenging than the MNIST data set.

1) Effect of the Parameters: The experiments here are not meant for a detailed sensitivity analysis. These experiments and related discussions aim to offer a basic understanding of parameter settings that may be helpful in practice. We mainly conduct research on the parameters listed in Table I.

In investigating the influence of a particular parameter, all others are set to be their default value. Experiment on $freq$ are carried out for $freq = \{3/15, 5/15, 7/15\}$, given $a = e/2$, $K = 20$, and $m = 2$.

Two metrics are adopted in this article for measuring the performance of the compared algorithms. One is the best accuracy of the central model within 200 rounds, and the other is to the required rounds before the central model's accuracy reaches 95.0%. Note that the same computing rounds mean the same communication cost. All experiments are independently run for ten times, and their average (AVG) and standard deviation (STDEV) values are presented in Tables IV–VI. In Tables IV–VI, the average value is listed before the standard deviation in parenthesis.

TABLE V

EXPERIMENTAL RESULTS ON a

a	Accuracy*	Round**
e***	96.92% (0.0041)	75.26 (2.09)
e/2	97.08% (0.0037)	87.82 (20.03)

* AVG (STDEV) of best overall accuracy within 200 rounds.
** ASTW_FedAVG on 1@MNIST reaches the accuracy 95.0% within 200 rounds.
*** e ≈ 2.72

TABLE VI

EXPERIMENTAL RESULTS ON THE SCALABILITY OF THE ALGORITHM

Scalability		FedAVG	ASTW_FedAVG
K	m		
10	1	94.26% (0.0083)	**94.76% (0.0102)**
10	2	**96.16% (0.0167)**	96.11% (0.0909)
20	2	96.83% (0.0097)	**97.16% (0.0096)**
30	3	96.50% (0.0020)	**97.79% (0.0067)**
30	9	97.14% (0.0029)	**98.22% (0.0070)**

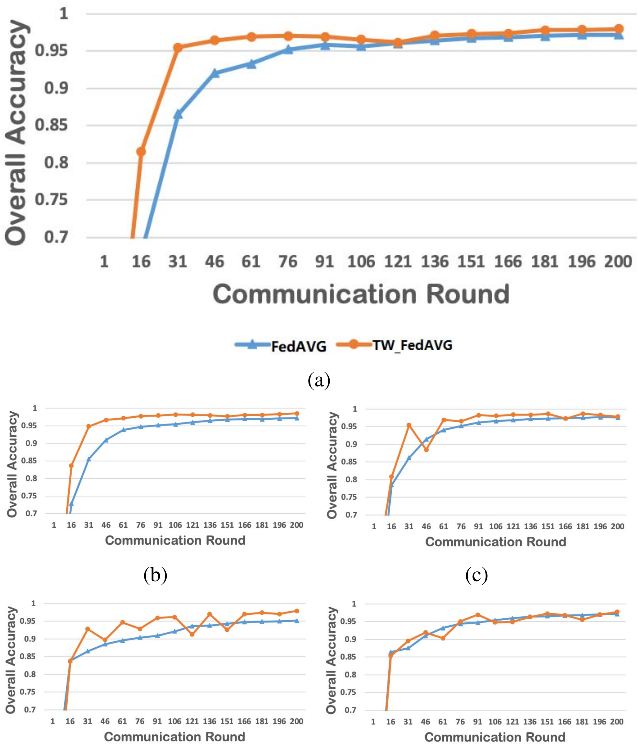
* AVG (STDEV) of best overall accuracy within 200 rounds.

Based on the results in Tables IV–VI, the following observations can be made.

1) *Analysis on freq:* From the results presented in Table IV, we can conclude that the lower the exchange frequency is, the fewer communication costs will be required, which is very much expected. However, a too low $freq$ will deteriorate the accuracy of the central model.
2) *Analysis on a:* a is a parameter that controls the influence of time effect on the model aggregation. When a is set as e, the more recently updated local models are more heavily weighted in the aggregated model, and a takes the value $e/2$, the previously updated local models will have a greater impact on the central model. The results in Table V indicate that $e/2$ is a better option for CNN on the 1@MNIST data set. When $a = 1$, the algorithm is reduced to AS_FedAVG, meaning that the parameters uploaded in different rounds will be of equal importance in the aggregation.

Both parameters, K and m, leverage the scalability of federated learning. The AVG and STDEV values are calculated based on the recognition accuracy of the CNN on the five predefined data sets. Different combinations of K and m are separately assessed. Based on the results presented in Table VI, the following three conclusions can be made.

First, a larger number of involved clients (a larger m) leads to higher recognition accuracy. Second, ASTW_FedAVG outperforms FedAVG in most cases, as indicated in the results in the first, third, fourth, and fifth rows in Table VI. Third, FedAVG is slightly better when K, i.e., the total number of clients, is smaller and C is higher, as shown by the results listed in the second row of the table. This implies that the advantage of the proposed algorithm over the traditional federated learning will become more apparent as the number of clients increases. This is encouraging since, in most real-world problems, the number of clients is typically very large.



(a)

(b) (c)

(d) (e)

Fig. 7. Comparative studies on temporally weighted aggregation on data set MNIST using the CNN. (a) 1@MNIST. (b) 2@MNIST. (c) 3@MNIST. (d) 4@MNIST. (e) 5@MNIST.

2) Comparison on Accuracy and Communication Cost: The following experiments are conducted for testing the overall performance of the algorithms under comparison using the default values of the parameters. Five local data sets are predefined for MNIST and HAR, respectively. For instance, local data set 1 of task MNIST is termed as 1@MNIST. The importance of the temporal weighting is first demonstrated by comparing the changes in the accuracy over the computing rounds. The baseline FedAVG and TW_FedAVG without the asynchronous update are compared, and the results on the MNIST and HAR data sets are shown in Figs. 7 and 8, respectively.

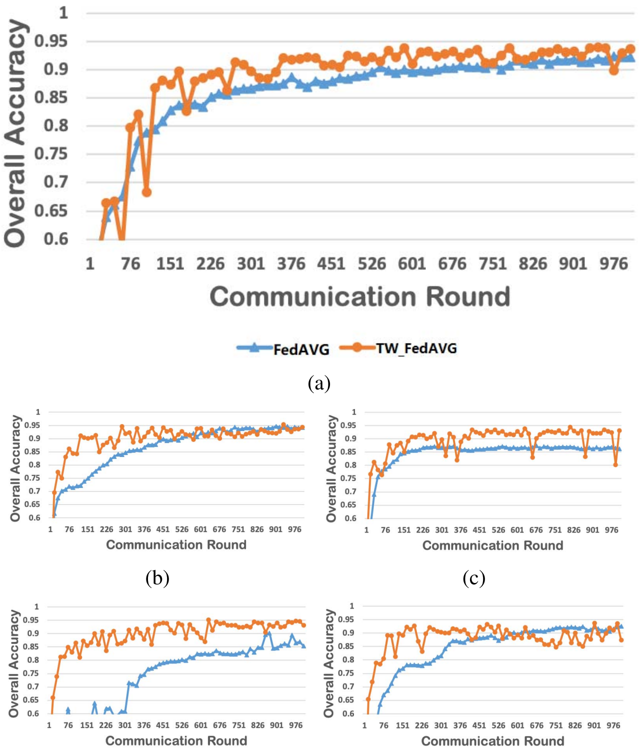
The following conclusions can be reached from the results in Figs. 7 and 8. First, the proposed temporally weighted aggregation helps the central model to converge to an acceptable accuracy. On the 1@MNIST and 2@MNIST data sets, TW_FedAVG needs about 30 communication rounds to achieve an accuracy reaches to 95.0%, while the traditional FedAVG needs about 75 rounds to achieve similar accuracy, leading to a reduction of 40% communication cost. Similar conclusions can also be drawn on data sets 3@MNIST, 4@MNIST, and 5@MNIST, although the accuracy of TW_FedAVG becomes more fluctuating. Second, on the HAR data set, a more difficult task, TW_FedAVG can mostly achieve higher accuracy than FedAVG except on 5@HAR. Notably, TW_FedAVG only needs about 75 communication rounds to achieve an accuracy of 90%, while FedAVG requires about 750 rounds, resulting in a nearly 90% reduction of the communication cost. Even on 5@HAR,

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE VII
EXPERIMENTS ON PERFORMANCE

Dataset_ID@Task	**FedAVG**		**TW_FedAVG**		**ASTW_FedAVG**		**AS_FedAVG**	
	Round (Accuracy)*	*C. Cost***	*Round (Accuracy)**	*C. Cost***	*Round (Accuracy)**	*C. Cost***	*Round (Accuracy)**	*C. Cost***
1@MNIST*	75 (97.2%)	6.16	**31 (97.9%)**	0.74	106 (97.7%)	1	175 (95.4%)	2.46
2@MNIST*	85 (97.2%)	6.76	**32 (98.5%)**	1.33	61 (98.1%)	**1**	—(94.8%)†	—†
3@MNIST*	73 (97.7%)	5.99	**31 (98.7%)**	1.13	70 (97.9%)	**1**	—(94.9%)†	—†
4@MNIST*	196 (95.2%)	8.18	**61 (98.0%)**	1.14	136 (96.1%)	**1**	—(93.1%)†	—†
5@MNIST*	98 (97.1%)	3.28	**76 (97.8%)**	3.17	61 (96.1%)	**1**	109 (96.7%)	2.66
1@HAR**	526 (92.3%)	4.72	**166** (94.0%)	**0.69**	358 **(94.6%)**	1	—(89.2%) ‡	—‡
2@HAR**	451 (94.7%)	5.65	**119** (95.4%)	**0.57**	313 **(95.9%)**	1	—(82.9%) ‡	—‡
3@HAR**	—(87.3%) ‡	—‡	**174 (94.4%)**	**0.69**	376 (93.2%)	1	—(88.5%) ‡	—‡
4@HAR**	856 (90.2%)	7.05	**181 (95.1%)**	1.28	211 (94.1%)	**1**	751 (91.2%)	5.30
5@HAR**	571 (92.6%)	5.49	**155 (93.6%)**	**0.57**	404 (93.1%)	1	646 (92.5%)	2.38

* Cells are filled when the accuracy reaches 95% within 200 rounds.
** Cells are filled when the accuracy reaches 90% within 1000 rounds.
† Cells are marked when it can not reach 95% within 200 rounds.
‡ Cells are marked when it can not reach 90% within 1000 rounds.
* Rounds are needed to reach a certain accuracy, after which the best accuracy (within 200/1000 rounds) is listed in parenthesis.
** Total communication cost; the C. cost of ASTW_FedAVG is termed as 1.



(a)

(b)

(c)

(d)

(e)

Fig. 8. Comparative studies on temporally weighted aggregation on data set HAR using the LSTM. (a) 1@HAR. (b) 2@HAR. (c) 3@HAR. (d) 4@HAR. (e) 5@HAR.

TW_FedAVG shows a much faster convergence than FedAVG in the early stage. Finally, the temporally weighted aggregation may result in some fluctuations in the learning performance, which may be attributed to the fact that the contributions of some high-quality local models are less weighted in some communication rounds.

Finally, the comparative results of the four algorithms on ten test cases generated from MNIST and HAR tasks are given in Table VII. The listed metrics include the number of rounds needed, the classification accuracy (listed in parenthesis), and total communication cost (Cost. C for short). From these results, the following observations can be made.

1) Both ASTW_FedAVG and TW_FedAVG outperform FedAVG in most cases in terms of the total number of rounds, the best accuracy, and the total communication cost.
2) TW_FedAVG achieves the best performance on most tasks in terms of the total number of rounds and the best accuracy. The temporally weighted aggregation strategy accelerates the convergence of the learning and improved the learning performance.
3) ASTW_FedAVG performs slightly better than TW_FedAVG on MNIST in terms of the total communication cost, while TW_FedAVG works better than ASTW_FedAVG on the HAR data sets. The layerwise asynchronous model update strategy significantly contributes to reducing the communication cost per round.
4) AS_FedAVG performs the worst among the four compared algorithms. When comparing the performance of the AS_FedAVG only adopting the asynchronous strategy and the ASTW_FedAVG using both of them, the asynchronous one always needs the help of the other strategy.

V. CONCLUSION AND FUTURE WORK

This article aims to reduce the communication costs and improve the learning performance of federated learning by suggesting an asynchronous model update strategy and a temporally weighted aggregation method. Empirical studies comparing the performance and communication costs of the canonical federated learning and the proposed federated learning on the MNIST and human action recognition data sets demonstrate that the proposed asynchronous federated learning with temporally weighted aggregation outperforms the canonical one in terms of both learning performance and communication costs.

This article follows the assumption that all local models adopt the same neural network architecture and share the same hyperparameters, such as the learning rate of SGD. In future research, we are going to develop new federated learning algorithms allowing clients to evolve their local models to further improve the learning performance and reduce communication costs.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[2] J. Konecný, B. McMahan, and D. Ramage, "Federated optimization: Distributed optimization beyond the datacenter," Nov. 2015, *arXiv:1511.03575*. [Online]. Available: https://arxiv.org/abs/1511.03575

[3] J. Konecný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *CoRR*, Oct. 2016.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[5] H.-C. Shin *et al.*, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE Trans. Med. Imag.*, vol. 35, no. 5, pp. 1285–1298, May 2016.

[6] K. Greff, R. K. Srivastava, J. Koutnìk, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[7] C. Ma *et al.*, "Distributed optimization with arbitrary local solvers," *Optim. Methods Softw.*, vol. 32, no. 4, pp. 813–848, 2017.

[8] S. J. Reddi, J. Konecný, P. Richtárik, B. Póczós, and A. Smola, "Aide: Fast and communication efficient distributed optimization," Aug. 2016, *arXiv:1608.06879*. [Online]. Available: https://arxiv.org/abs/1608.06879

[9] O. Shamir, N. Srebro, and T. Zhang, "Communication-efficient distributed optimization using an approximate newton-type method," in *Proc. Int. Conf. Mach. Learn.*, Jan. 2014, pp. 1000–1008.

[10] Y. Zhang and X. Lin, "Disco: Distributed optimization for self-concordant empirical loss," in *Proc. Int. Conf. Mach. Learn.*, Jun. 2015, pp. 362–370.

[11] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *Proc. OSDI*, vol. 14, 2014, pp. 571–582.

[12] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1223–1231.

[13] J. Konecný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," Oct. 2016, *arXiv:1610.02527*. [Online]. Available: https://arxiv.org/abs/1610.02527

[14] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011.

[15] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3320–3328.

[16] B.-I. Koh, A. D. George, R. T. Haftka, and B. J. Fregly, "Parallel asynchronous particle swarm optimization," *Int. J. Numer. Methods Eng.*, vol. 67, no. 4, pp. 578–595, Jul. 2006.

[17] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," Dec. 2017, *arXiv:1712.07557*. [Online]. Available: https://arxiv.org/abs/1712.07557

[18] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct./Nov. 2017, pp. 1175–1191.

[19] H. Zhu and Y. Jin, "Multi-objective evolutionary federated learning," *IEEE Trans. neural Netw. Learn. Syst.*, to be published.

[20] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," Apr. 2014, *arXiv:1404.5997*. [Online]. Available: https://arxiv.org/abs/1404.5997

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[23] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," in *Proc. ESANN*, Apr. 2013, pp. 437–442.

[24] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," in *Proc. 9th Int. Conf. Artif. Neural Netw.*, Sep. 1999, pp. 850–855.

Yang Chen received the Ph.D. degree in control theory and control engineering from the School of Information and Control Engineering, China University of Mining and Technology, Xuzhou, China, in 2019.

He is currently a Post-Doctoral Research Fellow with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His current research interests include deep learning, secure machine learning, edge computing, anomaly detection, evolutionary computation, and intelligence optimization.

Xiaoyan Sun (M'12) received the Ph.D. degree in control theory and control engineering from the China University of Mining and Technology, Xuzhou, China, in 2009.

She is currently a Professor with the School of Information and Control Engineering, China University of Mining and Technology. Her current research interests include interactive evolutionary computation, big data, and intelligence optimization.

Yaochu Jin (M'98–SM'02–F'16) received the B.Sc., M.Sc., and Ph.D. degrees from Zhejiang University, Hangzhou, China, in 1988, 1991, and 1996, respectively, and the Dr.-Ing. degree from Ruhr-University Bochum, Bochum, Germany, in 2001.

He was a Finland Distinguished Professor funded by the Finnish Funding Agency for Innovation (Tekes) and a Changjiang Distinguished Visiting Professor appointed by the Ministry of Education, Beijing, China. He is currently the Distinguished Chair Professor in computational intelligence with the Department of Computer Science, University of Surrey, Guildford, U.K., where he heads the Nature Inspired Computing and Engineering Group. His research has been funded by the EU, EPSRC, Royal Society, NSFC, and the industry, including Honda, Germany, Airbus, U.K., and Bosch, Germany and U.K. His current research interests include data-driven surrogate-assisted evolutionary optimization, evolutionary learning, neural architecture search, privacy-preserving and adversarial machine learning, and evolutionary developmental systems.

Dr. Jin was an IEEE Distinguished Lecturer from 2013 to 2015 and from 2017 to 2019, and the past Vice President for Technical Activities of the IEEE Computational Intelligence Society from 2014 to 2015. He was the General Co-Chair of the 2016 IEEE Symposium Series on Computational Intelligence, the Registration Chair of the 2016 IEEE World Congress on Computational Intelligence, and the Chair of the 2020 IEEE Congress on Evolutionary Computation. He was a recipient of the Best Paper Award of the 2010 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology, the 2015 and 2017 *IEEE Computational Intelligence Magazine* Outstanding Paper Award, and the 2018 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Award. He is the Editor-in-Chief of the IEEE TRANSACTIONS ON COGNITIVE AND DEVELOPMENTAL SYSTEMS and Coeditor-in-Chief of *Complex & Intelligent Systems*. He was named as a Highly Cited Researcher for 2019 by the Web of Science Group.