From Server-Based to Client-Based Machine Learning: A Comprehensive Survey

RENJIE GU, CHAOYUE NIU, FAN WU, and GUIHAI CHEN,
Shanghai Jiao Tong University, China
CHUN HU, CHENGFEI LYU, and ZHIHUA WU, Alibaba Group, China

In recent years, mobile devices have gained increasing development with stronger computation capability and larger storage space. Some of the computation-intensive machine learning tasks can now be run on mobile devices. To exploit the resources available on mobile devices and preserve personal privacy, the concept of client-based machine learning has been proposed. It leverages the users' local hardware and local data to solve machine learning sub-problems on mobile devices and only uploads computation results rather than the original data for the optimization of the global model. Such an architecture can not only relieve computation and storage burdens on servers but also protect the users' sensitive information. Another benefit is the bandwidth reduction because various kinds of local data can be involved in the training process without being uploaded. In this article, we provide a literature review on the progressive development of machine learning from server based to client based. We revisit a number of widely used server-based and client-based machine learning methods and applications. We also extensively discuss the challenges and future directions in this area. We believe that this survey will give a clear overview of client-based machine learning and provide guidelines on applying client-based machine learning to practice.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **Multi-agent systems**; **Mobile agents**; *Distributed algorithms*;

Additional Key Words and Phrases: Mobile intelligence, machine learning, distributed system, decentralized training, federated learning

ACM Reference format:
Renjie Gu, Chaoyue Niu, Fan Wu, Guihai Chen, Chun Hu, Chengfei Lyu, and Zhihua Wu. 2020. From Server-Based to Client-Based Machine Learning: A Comprehensive Survey. *ACM Comput. Surv.* 54, 1, Article 6 (December 2020), 36 pages.
https://doi.org/10.1145/3424660

This work was supported in part by National Key R&D Program of China No. 2019YFB2102200; in part by China NSF grant No. 61972252, 61972254, 61672348, and 61672353; in part by Joint Scientific Research Foundation of the State Education Ministry No. 6141A02033702; and in part by Alibaba Group through the Alibaba Innovation Research Program. The opinions, findings, conclusions, and recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the funding agencies or the government. The authors also want to sincerely thank Dr. Shuo Yang for offering valuable suggestions on polishing the initial version of this work.
Authors' addresses: R. Gu, C. Niu, F. Wu (corresponding author), and G. Chen, Shanghai Jiao Tong University, 800 Dongchuan Rd., Shanghai, China, 200240; emails: {grj165, rvince}@sjtu.edu.cn, {fwu, gchen}@cs.sjtu.edu.cn. C. Hu, C. Lyu, and Z. Wu, Alibaba Group, 969 Wenyi Rd. (W), Hangzhou, China, 311121; emails: {shiji.hc, chengfei.lcf, zhihua.wzh}@alibaba-inc.com.

0360-0300/2020/12-ART6 $15.00
https://doi.org/10.1145/3424660

1 INTRODUCTION

Machine learning, especially deep learning, has become a hot topic, attracting tremendous attention from both academia and industry. The core idea of machine learning is to use large amounts of data to train a model that can generalize well to unseen test samples. However, with the increase of data volume and the enhancement of model capacity, it is infeasible for a single server to accomplish complex learning tasks in a centralized way. To address this problem, the concept of server-based distributed machine learning was proposed in [82], where multiple servers, connected through shared data buses or a fast local area network, exchange essential information (e.g., training losses and gradients) to collaboratively train a model. Although this framework is highly scalable and has been widely deployed in practice, it may not always be cost effective and efficient to build a high-performance server cluster. In addition to cost, security and privacy are major concerns when machine learning involves sensitive user data, such as typed texts in natural language processing, user profiles in personalized recommendations, and health records in medical diagnosis. Specifically, the servers in both centralized and distributed machine learning frameworks require direct accesses to training data and thus need to collect and store user data, which inevitably suffers outsider and insider attacks [79–81]. For example, a malicious hacker may invade the datacenter, compromise parts of servers, and leak private databases. Further, if the servers are untrusted, they may share user data with other unauthorized entities or even trade for profits. In a nutshell, how to reduce the server cluster's operation cost, how the trusted servers can securely maintain user data, and how to defend untrusted servers are bottlenecks of the server-based machine learning.

Meanwhile, with the rapid proliferation and development of mobile devices, the idea of doing machine learning tasks on mobile devices has also emerged. For example, applications, such as face recognition and speech recognition, are all based on machine learning and are common among mobile phones. To support these applications, a full-sized machine learning model is first trained on servers using large amounts of data, and then it is tailored and delivered to mobile devices to do inference and make predictions locally. This framework brings all the burdens to the central servers, wasting the resources of mobile devices, whose processors, memory space, and disk space are now powerful and abundant enough to support various kinds of computation tasks. In addition, many off-the-shelf machine learning frameworks (e.g., the TensorFlow Lite module in TensorFlow [1]) are now available. Developers can now readily adopt these end-to-end tools to build machine learning models for their mobile applications. The above evidences have shown that it is feasible to deploy distributed training tasks on mobile devices, which is also called client-based training.

Client-based training has advantages in cost reduction and privacy preservation. In particular, machine learning problems are distributed to mobile devices and solved locally so that high-performance servers and user data transmission/maintenance are no longer required. The idea of client-based training can be traced back to 2015, when Shokri and Shmatikov [100] proposed distributed deep learning without sharing datasets among multiple parties. Later that year, Google researchers designed federated optimization [54], aiming to improve communication efficiency during learning with decentralized datasets. The idea was also referred to as Federated Learning and further developed in the following years [53, 55, 56, 70]. These works can be generally viewed as a specific type of client-based training that mainly focuses on how to make use of data without uploading them to the server, so that the privacy of users can be well preserved. If we enable on-device training and only upload the computation results, the leakage of sensitive information can be relieved. The reason is that attacks against the computation results without accessing the raw data are much harder. Moreover, since raw data is processed locally, client-based training is now able to make use of the data that is too much to be uploaded (so that centralized machine

learning doesn't take them into consideration), which gives us great opportunity to improve the model performance.

Compared to existing surveys in this area, this survey focuses on the evolution process of machine learning. From server-based machine learning to client-based machine learning, the application scenarios and the research problems have greatly changed. We summarize the changes and further investigate the underlying motivations. We also review the research focuses at different stages of machine learning development. In particular, considering that a lot of new features and new demands have emerged in client-based machine learning, we analyze the applicability of existing server-based algorithms to client-based machine learning. We finally point out some future directions of client-based machine learning. In a nutshell, this survey not only is a review of the development of machine learning but also can work as a good reference for designing new client-based learning algorithms on the basis of conventional server-based methods.

The rest of this survey is organized as follows. In Section 2 and Section 3, we introduce the general machine learning process and the methodology of server-based distributed training. In Section 4 and Section 5, we explain the motivations of client-based machine learning (including inference and training), list the challenges, and discuss current advances of client-based training using federated learning and split learning as examples. In Section 6, we discuss the open problems and future directions of client-based training. Finally, we conclude the survey in Section 7.

2 CENTRALIZED MACHINE LEARNING

Machine learning [11] is a study of mathematical models that can automatically learn and make predictions based on a set of observed data. The concept of centralized machine learning means that operations and executions of the model are all done on a central machine. Centralized machine learning has been widely used to extract insights behind huge amounts of data.

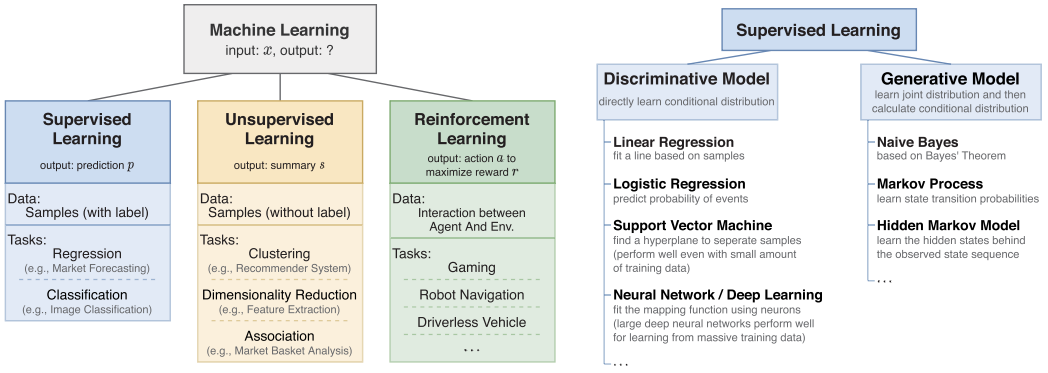
In this section, we briefly review concepts and techniques of centralized machine learning. To better understand the methodology of machine learning, we first introduce the paradigms of machine learning methods in Section 2.1. Next, several concepts that are commonly used in machine learning will be illustrated in Section 2.2. Then, the task of machine learning will be defined in Section 2.3. A general process of machine learning will be presented in Section 2.4. After that, several machine learning optimizers will be introduced in Section 2.5. Finally, the applicability of these optimizers to client-based training will be discussed in Section 2.6.

2.1 Machine Learning Paradigm

If we classify machine learning algorithms based on the kind of input data samples and the kind of output, we have the following three basic machine learning paradigms as shown in Figure 1(a).

Supervised Learning. In supervised learning, every data sample is made up of several input features and a label. The learning process is to approximate a mapping function from the features to the label. After that, given new input features, the label for the data can be predicted using the mapping function. This scheme is the most popular machine learning scheme, which has been used in a variety of tasks. An example of supervised learning is the classification task, which is to classify an object based on its features, such as classifying fruit according to its color, shape, and weight. If the supervised learning task is to predict a continuous variable such as market pricing, then this is a regression task. We can further classify supervised learning based on the model type, as shown in Figure 1(b). We mainly focus on supervised learning (especially discriminative models) in this survey.

Unsupervised Learning. In contrast to supervised learning, unsupervised learning is where we only have input features but no corresponding labels. Thus, the goal for unsupervised learning



(a) Machine Learning (b) Supervised Learning

Fig. 1. Taxonomy of machine learning and supervised learning.

is to learn the distribution of the data and show how the data points are different from each other. A typical example of unsupervised learning is the clustering problem, which is to discover the groupings of the data, such as grouping users by their behaviors.

Reinforcement Learning. Reinforcement learning is quite different from supervised learning and unsupervised learning. Labeled input/output pairs and explicit correction on sub-optimal options are not needed for training an agent using reinforcement learning. Instead, the agent tries to find a trade-off between exploration and exploitation through its interaction with the environment. For good choices or actions, the agent gains rewards from the interpreter. Otherwise, it is punished. Reinforcement learning is widely used in research about robots or computer gaming agents.

Others. In addition to the three basic paradigms, there also exist some other paradigms that can be viewed as a combination of the basic ones (e.g., semi-supervised learning). Usually, the choice of machine learning paradigms depends on the kind of the problem going to be solved.

2.2 Machine Learning Concepts

To better understand machine learning, in this section, we introduce and illustrate several key concepts that are commonly used in machine learning (especially supervised learning).

Stage and Dataset. The whole process of machine learning is mainly made up of three kinds of stages, which are training, validation, and test. The dataset $D = \{d_1, d_2, \ldots, d_n\}$ is the set of all n data samples used for the machine learning process. A data sample $d_i = (x_i, y_i)$ contains a feature vector x_i and a corresponding label y_i. Usually D can be divided into three disjoint sub-sets, which are the training set D_{train}, the validation set D_{vld}, and the test set D_{test}, corresponding to the three stages. D_{train} accounts for the largest proportion of D. In the training stage, loss is calculated on D_{train} and then optimization to the model is done. The validation stage aims to prevent overfitting. Since samples in D_{vld} are not used in any training iterations, the error calculated on them will increase significantly if the model overfits D_{train}. Moreover, D_{vld} is also helpful for tuning hyperparameters between training epochs. The test stage is to evaluate the performance of the final model on D_{test}. Thus, a typical machine leaning process is as follows: (1) run training and validation iteratively until the model converges, and (2) run test to evaluate the model performance.

Feature. In machine learning, a feature vector x is a k-dimensional vector containing numerical features that are observable or measurable properties of instances, objects, or phenomena.

We input feature vectors to the machine so that it knows how instances are different from each other.

Label. A label y is the identity of the instance. Unlike features that describe the instances to the machine, labels tell the machine what the instances really are.

Model. The machine learning model $f(w;\cdot)$ includes a structure f and a parameter vector w. The model $f(w;\cdot)$ is the core of machine learning. It serves as the mapping function that maps the input feature vector x to the output label $f(w;x)$. For a good model, its output label $f(w;x)$ is close to the true label y of the feature vector x. Usually, the structure and the parameters of the model are stored as multiple vectors or matrices. The mapping process is done through matrix multiplication. Various kinds of models such as support vector machines and artificial neural networks have been designed for different machine learning tasks.

Loss Function. The loss function $L(y, f(w;x))$ (e.g., mean square error, hinge loss, and softmax loss) describes how far the predicted label $f(w;x)$ deviates from the true label y. Since a pair (x, y) is also known as a data sample d, we can also denote the loss function by $L(w;d)$ for convenience. The design of the loss function L reflects the goal of the training process, which is to train the model to make predictions as accurately as possible. For example, mean square error is defined as

$$\text{MSE} = \sum_{i=1}^{n}(y_i - f(w;x_i))^2, \tag{1}$$

where n is the number of samples, w stands for the model parameters, x_i is the feature vector of the ith sample, $f(x_i)$ is the predicted label for the ith sample, and y_i is the true label of the ith sample. Thus, in a regression problem using mean square error as the loss function, the model w will get more punishment if the predicted label $f(x_i)$ is further from the true label y_i. The value of the loss function L over D_{train} is calculated as

$$L(w;D_{train}) = \frac{1}{n_{train}}\sum_{i=1}^{n_{train}} L(w;d_i) = \frac{1}{n_{train}}\sum_{i=1}^{n_{train}} L(y_i, f(w;x_i)), \tag{2}$$

where n_{train} is the number of samples in D_{train}. Here $L(w;D_{train})$ is also known as the empirical risk. The change in the value of $L(w;D_{train})$ shows the training progress of the model w. The lower the value of $L(w;D_{train})$ is, the better the model w is trained on D_{train}. Note that norms should be added to the loss function to avoid the overfitting problem. In particular, overfitting means the model is too closely fit to D_{train} and has a poor performance when dealing with new data.

Training (Optimization). After the loss is calculated, the model will be optimized iteratively to achieve a better generalization ability. This is also known as the training process. Typical optimization algorithms such as gradient decent are used to adjust parameters of the model according to the loss and the model structure. Considering that heavy calculation is required in this step, computational tricks like back-propagation are usually applied to improve system efficiency.

Inference (Prediction). With a trained model $f(w;\cdot)$, we can input a feature vector x to it. Then, after some calculation, the model outputs a predicted label $f(w;x)$. This process is also known as the inference process. Usually inference is done by doing several times of matrix multiplication.

2.3 Task Definition

In machine learning, we first design the model structure f according to what kind of problem we are going to solve. Then the goal is to find out a parameter vector w that minimizes the expectation

of the loss function. This can be expressed as

$$\underset{w}{\operatorname{argmin}} \mathbf{E}[L(y, f(w; x))], \tag{3}$$

where $x \in X$ is the input feature and $y \in Y$ is the output label. It is assumed that the feature space X and the label space Y obey a joint probability distribution $P(x, y)$. Since the real $P(x, y)$ is unknown and is impossible to be figured out on most occasions, we are not able to directly optimize our model parameter w to minimize the expectation of the loss function $\mathbf{E}[L(y, f(w; x))]$. As an approximation, we minimize the empirical risk $L(w; D_{train})$ on our training set D_{train}. This approximation requires D_{train} to be a set that contains Independent and Identically Distributed (IID) random variables drawn from $P(x, y)$. Now the core learning task is expressed as

$$\underset{w}{\operatorname{argmin}} L(w; D_{train})) = \underset{w}{\operatorname{argmin}} \frac{1}{n_{train}} \sum_{i=1}^{n_{train}} L(w; d_i) = \underset{w}{\operatorname{argmin}} \frac{1}{n_{train}} \sum_{i=1}^{n_{train}} L(y_i, f(w; x_i)), \tag{4}$$

where n_{train} is the number of samples in D_{train}, and $d_i = (x_i, y_i)$ is the ith sample in D_{train}.

2.4 Machine Learning Process

The procedure of machine learning can be divided into two parts: (1) designing a suitable model structure based on the machine learning task and (2) examining the performance of the model and optimizing it accordingly using large amounts of data.

Model Design. For the model design part, there already exists many well-designed models that have been proposed to solve different kinds of tasks, as shown in Figure 1(b). A good survey on popular traditional machine learning algorithms and models can be found in [116]. What's more, in recent years, deep learning [58] has proved to be very effective in many areas such as image recognition and natural language processing. The word "deep" is used to describe the multi-layer structure of the neural network used by it. The concept of deep learning can be further divided into Convolutional Neural Networks (CNNs), Recursive Neural Networks (RNNs), Generative Adversarial Networks (GANs), and so on according to the neural network structure. Considering that the scale of a deep learning model can be very large, it is usually trained on high-performance servers. In general, deep learning is a powerful machine learning technique that is based on neural networks and benefits from the increase in the amount of training data.

Model Optimization. Regarding the model optimization part, it is done through machine learning optimizers. Generally speaking, an optimizer focuses on how to optimize the model to reach its best performance based on the given dataset. For example, Gradient Descent (GD) calculates the gradient of the loss function and uses this gradient to optimize the model parameters. In each training iteration, according to the calculated gradient, the model parameter w takes a step toward the optimal point where $L(w; D_{train})$ is minimized. However, the computation cost of GD is so high that it is not suitable for being applied to those models with a large number of parameters. Thus, many different schemes have been designed to find a better balance between the convergence speed and the computational cost. For those machine learning algorithms that are not suitable for using GD-based methods as their optimizers, they have their specific optimization algorithms, such as Sequential Minimal Optimization (SMO) for SVM and Canopy for k-means.

2.5 GD-Based Optimizers

Although there are many kinds of optimizers in centralized machine learning, in this survey, we mainly focus on the GD-based optimizers that originate from centralized machine learning and are now being widely used in distributed training and deep learning. In this section, we introduce

Table 1. GD-Based Optimizers

Name	Mathematical Format	Convergence	Speed	Experimental Scale
SGD	Gradient (1 sample)	Sublinear (all)	Baseline	An email dataset (100 machines) [127]
SVRG	Gradient (all samples)	Linear (strongly convex) Sublinear (convex) Sublinear (non-convex)	Slow ↓	MNIST [59] and CIFAR-10 [39] (1 machine)
Adam	SGD + Moment	Not Guaranteed	Fast ↑↑	MNIST and CIFAR-10 (1 machine)
Hogwild!	SGD	Linear (strongly convex, constant stepsize)	Equivalent	RCV1 [60], Netflix, KDD, Jumbo, DBLife, and Abdomen (10 machines)
ASGD	SGD	Sublinear (strongly convex)	Equivalent	A speech dataset and ImageNet [90] (128 workers)
EASGD	SGD + Elastic Update	? (complicated form)	Fast ↑	CIFAR-10 (16 workers), ImageNet (8 workers)
DC-ASGD	SGD + Delay Compensation	Sublinear (strongly convex)	Fast ↑	CIFAR-10 (8 workers), ImageNet (16 workers)

several GD-based optimizers and analyze their advantages. We list the basic information of them in Table 1. Note that the column "Speed" in the table has considered both the computational cost per iteration and the total number of iterations to reach convergence. Among these optimizers, SGD, SVRG, Adam, and Hogwild! are designed for serial centralized machine learning or parallel multi-thread centralized machine learning. ASGD, EASGD, and DC-ASGD are actually designed for a *server-worker* scheme, which means they should be classified as distributed training techniques. However, we still choose to introduce them here for the convenience of comparing them with other optimizers and studying how GD-based optimizers have developed from centralized to distributed.

SGD. Stochastic Gradient Descent (SGD) [89] was first proposed in 1951. The main advantage of SGD is that it greatly reduces the computational cost in each iteration compared with GD. Its core equations are very similar to those of GD and are shown as follows:

$$g_t = \nabla L\left(w_{t-1}; d_i\right), \qquad w_t = w_{t-1} - \eta \cdot g_t, \tag{5}$$

where t is the timestamp for the current training iteration, w_{t-1} is the model at time $t-1$, g_t is the gradient of the loss function L with model w_{t-1} and a randomly selected data sample d_i, and η is the learning rate. The feature that it only uses one sample to compute gradients in each iteration greatly reduces the computational cost. However, SGD should not be directly applied to client-based training since it cannot handle the bias caused by the non-IID local dataset (will be introduced in Section 5.4). Modification to SGD is necessary to fit the scenario of client-based training (e.g., FedAvg [70]). Although the solution for client-based training on mobile devices is unlikely to choose the original SGD as the optimizer, we have to say that SGD still works well on many other occasions (such as server-based distributed training) due to its low computational cost and high training efficiency.

SVRG. Stochastic Variance Reduced Gradient (SVRG) [46] aims at accelerating the convergence speed of SGD by applying noise reduction methods. Compared with GD, SGD does much less computation in each iteration but has a lower convergence speed. Bottou et al. [14] discovered that one reason for this is the existence of noise in the estimate of the gradient, which also can be considered as the variance of gradients. Thus, SVRG uses the overall gradient to make corrections

and thus reduces the noise. The core equations are shown as follows:

$$g'_t = \nabla f\left(w_{t-1}; d_i\right) - \nabla f\left(\bar{w}; d_i\right) + \bar{g}, \qquad w_t = w_{t-1} - \eta \cdot g'_t, \tag{6}$$

where $\bar{w}$ is an averaged model that is updated every k iterations and $\bar{g}$ is the gradient averaged among all data samples at point $\bar{w}$. The first term $\nabla f(w_{t-1}; d_i)$ on the right side of the first equation is exactly the g_t used in SGD. The core idea of SVRG is to make the upper bound of gradients' variance keep reducing during training by using correction $(-\nabla f(\bar{w}; d_i) + \bar{g})$. McMahan et al. [70] found out that SVRG can cooperate well with some distributed optimization algorithms like DANE [96] by working as the local optimizer. The main factor that limits SVRG's applicability to client-based training is the high computational cost of periodically calculating the overall gradient $\bar{g}$.

Adam. Adam is an optimization algorithm based on SGD aiming to accelerate the convergence speed by adaptively tuning the learning rate. It was proposed in 2014 [50]. Before Adam, there already existed some algorithms trying to improve SGD through making use of the moment/momentum of gradients, such as SGD with Momentum (SGDM) [83], AdaGrad [26], and RMSProp [111]. Adam combines the advantage of AdaGrad and RMSProp and uses both the first moment estimate and the second moment estimate. Its equations are given as follows:

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t, & v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot (g_t)^2, \\ \hat{m}_t &= \frac{m_t}{1-(\beta_1)^t}, & \hat{v}_t &= \frac{v_t}{1-(\beta_2)^t}, & w_t &= w_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}. \end{aligned} \tag{7}$$

Here, m_t is the first moment estimate and v_t is the second moment estimate. $\beta_1, \beta_2 \in [0, 1)$ are exponential decay rates for the moment estimates. $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected version of the moment estimates. ϵ is a small constant used to prevent division by zero. Experiments show that Adam can accelerate training. Considering that the number of communication rounds may be limited in client-based training, an efficient optimizer that can speed up the training process may help a lot. However, Wilson et al. [115] and Sashank et al. [92] discovered that on some special occasions, Adam may fail to reach convergence. This can be the main barrier for applying Adam to client-based training.

Hogwild!. Hogwild! [86] aims to prove that parallel SGD can be implemented without any locking. It shows that when the optimization problem is strongly convex and sparse, most updates only modify small subsets of all parameters, which means the whole update process can be run asynchronously without locking. However, it has only been tested on traditional machine learning problems like sparse SVM and matrix completion. Whether Hogwild! is suitable for complex tasks, such as deep learning and client-based training, still remains unknown.

ASGD. Asynchronous SGD (ASGD) [24] is a simple attempt for making use of more workers to train a huge deep network through asynchronous methods with SGD. Compared with synchronous SGD, ASGD won't suffer from the straggler problem, which is a huge obstacle to deploying distributed machine learning on heterogeneous mobile devices. Since no waiting is needed, all workers can make the best use of their resources and together accelerate the training. The problem is that in asynchronous methods, the delayed gradients may be unsuitable for being applied to the current updated model. The delay error can cause fluctuation in weights and have negative effects on the model according to [8, 63]. This delay error problem can be even worse in client-based training due to the large number of workers and the high frequency of model updates.

EASGD. The purpose of Elastic ASGD (EASGD) [123] is to reduce the communication cost between workers and the parameter server during parallel training. Each worker's local model is not

Table 2. Applicability of GD-Based Machine Learning Optimizers to Client-Based Training

Name	Time Complexity	Space Complexity	Scalability	Asynchronization	Delay Solution
SGD	Baseline ✓	Basic Model ✓	100 machines ✓	No ✗	/
SVRG	High↑↑ ✗	Historical Gradient, Historical Model ✗	1 machine	No ✗	/
Adam	High↑	Historical Gradient ✗	1 machine	No ✗	/
Hogwild!	Equivalent ✓	Basic Model ✓	10 machines	Yes ✓	None ✗
ASGD	Equivalent ✓	Basic Model ✓	128 workers ✓	Yes ✓	None ✗
EASGD	Equivalent ✓	Basic Model ✓	8 workers	Yes ✓	Not Needed ✓
DC-ASGD	Equivalent ✓	Historical Model ✗	16 workers	Yes ✓	Compensating ✓

replaced by the global model in each communication round. The communication and coordination of work among all workers is controlled by an elastic force that links the local parameters with a center variable stored by the parameter server. The update rules are shown as follows:

$$w_t^i = w_{t-1}^i - \eta \cdot \left(g_t^i + \rho \cdot \left(w_{t-1}^i - \bar{w}_{t-1}\right)\right), \qquad \bar{w}_t = \bar{w}_{t-1} + \eta \cdot \sum_{i=1}^{p} \rho \cdot \left(w_{t-1}^i - \bar{w}_{t-1}\right), \tag{8}$$

where i is a random index of a worker, p is the number of workers, ρ is the control parameter for the elasticity, and $\bar{w}_t$ is the center variable. The center variable $\bar{w}_t$ is updated as a moving average that is taken in both time and space over all local parameters. The elastic design allows workers to do more exploration in its nearby parameter space, which can do good to the model performance. The effectiveness of EASGD has only been analyzed for quadratic and strongly convex objectives. It is worthy to worry that the model may become even worse if the elastic hyperparameter isn't set properly and causes the workers to explore too far away from the center variable.

DC-ASGD. Delay Compensated ASGD (DC-ASGD) [126] focuses on mitigating the error caused by delayed gradients in ASGD. The main idea of DC-ASGD is to use the first-order term in Taylor series to compensate for the delayed gradient and use an approximation of the Hessian Matrix to reduce the computational cost. At time $t + \tau$, to update model $w_{t+\tau}$, the original delayed gradient $g(w_t)$ for old model w_t will be replaced by the delay-compensated gradient, which is expressed as

$$g\left(w_t\right) + \lambda g\left(w_t\right) \odot g\left(w_t\right) \odot \left(w_{t+\tau} - w_t\right), \tag{9}$$

where λ is a variance control parameter set by the server. The only additional information needed for compensation is the historical model w_t, which means this method is easy to implement. However, since experiments have been done with no more than 16 workers, DC-ASGD's performance under large numbers of workers still needs to be studied. In addition, another key problem of applying it to client-based training is that the server needs to spend large additional storage to store the historical models for all workers.

2.6 Applicability of GD-Based Optimizers to Client-Based Training

In this section, we compare the applicability of the GD-based optimizers to client-based training in detail, as shown in Table 2. The main difficulty of applying these optimizers to client-based training is that mobile devices may not have enough resources to run them on large models. A possible solution is to use distributed optimization algorithms (which will be introduced in Section 3.5) to decompose an original large problem into multiple small sub-problems. The aforementioned optimizers should cooperate with distributed optimization algorithms and work as the local optimizer. Therefore, we analyze the applicability of GD-based optimizers from two aspects: (1) whether the

time and space complexities are acceptable and affordable for mobile devices and (2) whether the optimizers support distributed computing in terms of *scalability, asynchronization, and delay solution.*

Time Complexity. For client-based training, optimizers with lower time complexity are preferred since they require fewer resources. Hard et al. [37] demonstrate that running SGD on mobile devices is feasible. We regard SGD as the baseline here. For SVRG, its periodical calculation of the overall gradient incurs huge local computation overhead. For Adam, to accelerate the gradient descent step, it introduces additional gradient processing steps that contain several times of matrix multiplication. So Adam's time complexity is relatively higher but still acceptable. For other GD-based optimizers, the local computation step is just SGD. Thus, regarding the time complexity, most of these GD-based optimizers (except for SVRG) are applicable to client-based training.

Space Complexity. Some optimizers use additional information to accelerate training or mitigate errors. This incurs additional space overhead. For SVRG, it keeps an overall gradient and a model to reduce the variance of gradients. For Adam, it keeps the last gradient and the moment to accelerate training. For DC-ASGD, it needs to store a historical model for each worker for delay compensation. Therefore, these three optimizers require additional memory and storage space. For other GD-based optimizers, they do not need additional space. Therefore, regarding the space complexity, SGD, Hogwild!, ASGD, and EASGD are preferred.

Scalability. We use "machines" to express that multiple machines cooperate with each other. We use "workers" to show that a group of workers are managed by a central server and may not need to communicate with each other. For synchronous optimizers, their scalability is limited by the time-consuming synchronization step. The "thundering herd" problem also adds difficulty to implementing large-scale synchronous systems. For asynchronous optimizers, the obstacles are the update covering problem (i.e., later updates rewrite parameters and cover earlier updates) and the delay error problem. With more workers, asynchronous training becomes less stable. However, the impact of these problems has not been clearly analyzed yet. Thus, we only list the number of machines/workers used in experiments to reflect the potential scalability. According to Table 2, SGD and ASGD have shown good scalability.

Asynchronization. SGD, SVRG, and Adam are designed for centralized machine learning with only one machine. They require a costly synchronization step to ensure convergence when being applied to parallel training. By allowing the asynchronous update of the global model, the time-consuming synchronization step can be avoided. For example, Hogwild!, ASGD, EASGD, and DC-ASGD are designed for asynchronous machine learning.

Delay Solution. The unstable network condition and the limited resources in client-based training can easily cause update delay. In the asynchronous training scheme, the global model may have already been updated by others when the gradient arrives at the server. The delayed gradient thus becomes less accurate for the current global model. Directly applying delayed gradients to the global model can slow down the convergence speed due to the delay error. Hogwild! and ASGD have no solution for delayed gradients. They just ignore the delay error. EASGD does not need a delay solution because its global model is a moving average instead of being updated by gradients. DC-ASGD compensates for delayed gradients by using Taylor series expansion. Therefore, regarding delay solution, EASGD and DC-ASGD outperform Hogwild! and ASGD.

3 SERVER-BASED DISTRIBUTED TRAINING

Although centralized machine learning has shown good performance in many kinds of tasks, it cannot catch up with the growing demand of processing more data and training larger models.

Under this circumstance, server-based distributed training techniques have been developed. The concept of server-based training means that operations and executions of the machine learning model are all done on servers. In this section, we first introduce the motivations and then discuss distributed parallelism categories and distributed optimization algorithms.

3.1 Motivations

Training Acceleration. The original centralized machine learning scheme can only use the computation power of a single machine, which indicates that it may require a long time to train a good model when dealing with large amounts of data. A potential solution is to distribute data on different machines and let them process data samples simultaneously. As the heavy calculation is distributed to multiple machines and executed parallel, the training process is significantly accelerated.

Large Model Support. Large-scale deep learning has shown its effectiveness in many areas and has gained rapid development. However, for those large models that have billions and trillions of parameters to be optimized, the resources on a single machine can hardly support the learning task. Thus, researchers have studied the feasibility of dividing the large model into different parts and training them in parallel with multiple machines.

3.2 Task Definition

In Section 2.3, we mentioned that the objective function is $\arg\min_w L(w; D_{train})$. There are three key components in the formula: (1) Loss Function L, (2) Model parameters w, and (3) Dataset D_{train}. For these three components, we determine which of them is to be divided:

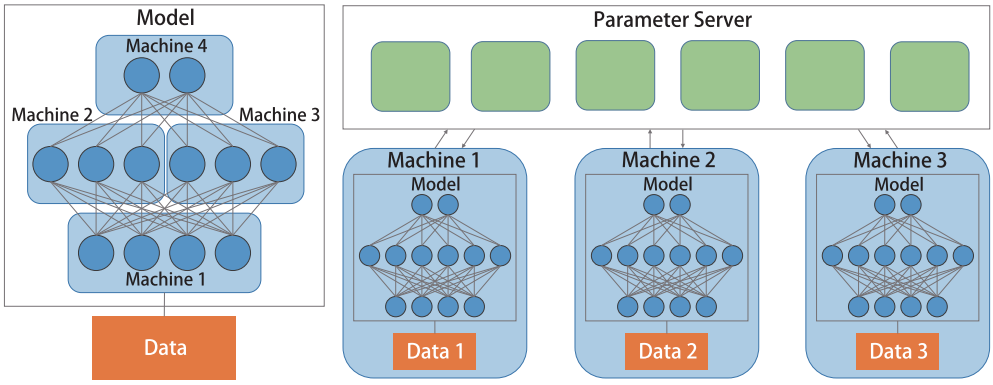
- Loss Function L: Since L usually takes only a little storage, it is not necessary to divide it.
- Dataset D_{train}: If D_{train} is large, we can divide and store it on several machines. After machines generate model updates through local training, the updates are transferred and aggregated to generate a new global model. This is known as distributed training with data parallelism.
- Model w: If w is large and contains huge amounts of parameters, we can let each machine process only one part of w. Intermediate results over parts of the model are transferred between machines. This is known as distributed training with model parallelism.

Note that both data parallelism and model parallelism let the whole dataset flow through the whole model, which guarantees the effectiveness of training. We normally do not divide the dataset and the model at the same time because this may cause incomplete training and result in performance loss.

3.3 Parallel Training Categories

As shown in Figure 2, there are two complementary architectures for distributed machine learning, namely data parallelism and model parallelism.

Data Parallelism. In most occasions, a large dataset that contains various kinds of data samples is very helpful for training a well-performed model. Sometimes the dataset is so large that it cannot be stored on a single machine. It is also possible that the huge amount of data results in a prohibitively slow training process. According to [77], distributed machine learning with data parallelism has emerged to solve the data storage problem and accelerate the training. In this scheme, the whole dataset is divided into sub-sets and distributed on machines. Each machine keeps a copy of the model and trains it based on the locally available part of data. After several iterations of training, the local models may become quite different from each other. The information is gathered



(a) Model Parallelism (b) Data Parallelism

Fig. 2. Architectures of model parallelism and data parallelism for server-based distributed training.

to generate an updated global model. This process is called data aggregation. Then, if the performance of the new global model is still not satisfying, another round of training is started. With data parallelism, more data is processed simultaneously, which means it speeds up the training.

Model Parallelism. In some special machine learning tasks, the model can be so large that it is too slow and even not able to be trained and run on a single machine. This problem is particularly serious in deep learning tasks. Thus, large-scale distributed deep networks are proposed in [24] trying to deal with it. Model parallelism methods are adopted and used to train large models with billions of parameters. In this scheme, each machine keeps a small part of the whole model. During training, the data flows through machines in order to be processed by the local sub-models. On most occasions, every round of training needs the cooperation of all machines. Therefore, this process should be done sequentially since the inputs of some machines depend on the outputs of others. In this situation, using a scheduler to manage the training process may be helpful for solving the dependency between machines. Compared with data parallelism, model parallelism is more complex and also harder to implement due to the strong cooperation among machines.

3.4 Parallel Communication Frameworks

Parallel communication frameworks help realize parallel training. Without a well-designed communication scheme, the limitation on the network bandwidth may become a troublesome bottleneck for the whole system. The frameworks can be divided into the following three kinds: (1) **MapReduce/AllReduce**, (2) **Parameter Server**, and (3) **Data Flow**. We briefly introduce their design in Sections 3.4.1 to 3.4.3. Then we give a simple discussion on this topic in Section 3.4.4.

3.4.1 MapReduce/AllReduce.

Design of MapReduce. In MapReduce, the map operation distributes data and tasks to workers and the reduce operation aggregates all results. The general process of MapReduce is shown in Figure 3. To accomplish the distributed computation task, several mappers and reducers are set on available nodes. Mappers read data from the storage, perform the mapping in parallel, and generate intermediate results. Reducers then aggregate intermediate results and generate the final result.

Design of AllReduce. One problem of MapReduce is that it takes huge communication costs to transfer intermediate results to the reducers. To deal with this problem, AllReduce is proposed and integrated into the Message Passing Interface (MPI). In AllReduce, all worker nodes also work as reducers. Parts of the intermediate results are transferred between workers if necessary. With

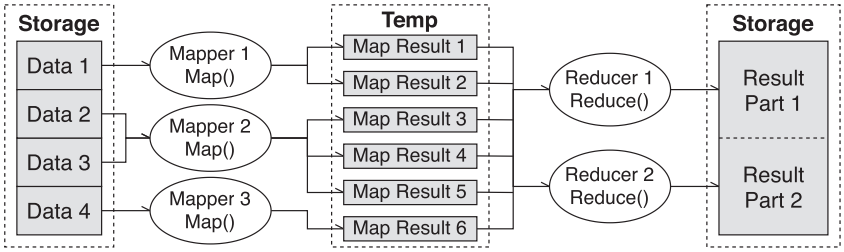


Fig. 3. Process of MapReduce.

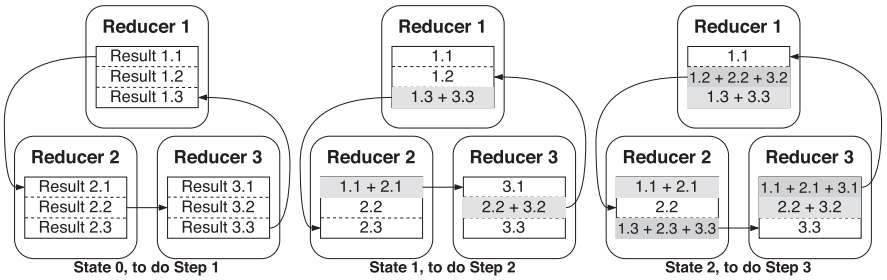


Fig. 4. Process of Ring AllReduce.

this design, the amount of transferred data decreases. The network burden is also relieved since all the workers' bandwidths are used during reducing. AllReduce has been realized using many different topologies [76]. Here we use ring topology (Figure 4) as an example to explain AllReduce. For k (here $k = 3$) nodes, we first use $(k - 1)$ steps to make each node keep $\frac{1}{k}$ of the final aggregated result. Then, with another $(k - 1)$ ring communication steps, the result on each node can be completed.

Pros/Cons. In machine learning, MapReduce/AllReduce is commonly used to make multiple machines cooperate with each other. One advantage of MapReduce/AllReduce is that it is easy to deploy. MapReduce has been applied to server-based distributed training in [33, 72]. Here, mapping is to retrieve the global model and generate model updates. Reducing is to aggregate model updates and update the global model. Baidu brought the Ring AllReduce technique to deep learning in 2017 [34]. AllReduce has now been supported in TensorFlow. In addition, we can use a third-party library, called Horovod [95], to simplify the implementation of AllReduce in deep learning. One problem of MapReduce/AllReduce is that it can be easily blocked by stragglers as synchronization is needed.

3.4.2 Parameter Server.

Design of Parameter Server. The parameter server (PS) can be either a single server or a server cluster that takes charge of the task arrangement but doesn't do the task by itself. The tasks are actually done by the workers. Each worker only has to communicate with the central server to pull or push data and has no need to be aware of other participators. This means one worker's computation task is independent of others' so that asynchronous working is possible. PS is more robust than MapReduce/AllReduce since it supports asynchronous communication and thus won't suffer from the straggler problem. An example of PS is shown in Figure 2(b).

Development of Parameter Server. The idea of PS came from the parallel Latent Dirichlet Allocation (LDA) architecture [107]. This first-generation PS used Memcached as the storage of

parameters and managed the synchronization of workers. The lack of flexibility and performance is its main disadvantage. After that, YahooLDA [4] and Distbelief [24] followed this idea and improved PS's design for specific applications. Petuum [40] was a more general platform based on YahooLDA, but it placed more constraints on worker threading models. These works are all considered as the second-generation PS. To build a more robust system, the third-generation PS was proposed and implemented in [61]. PS has been applied to distributed deep learning tasks by Google in [24].

Pros/Cons. Here, we first introduce the advantages of the third-generation PS: (1) *Efficient Communication:* Communication has been optimized for learning tasks to reduce overhead, and asynchronous communication is supported. (2) *Flexible Consistency:* The system allows three consistency models, including sequential consistency, eventual consistency, and bounded delay consistency. (3) *Elastic Scalability:* New servers can join without rebooting the whole system. (4) *Fault Tolerance and Durability:* Chain replication is used to back up data entries on servers. The vector clock design enables a failed node to be quickly recovered to its original working status. Compared to AllReduce, one disadvantage of PS is that all transferred data must pass through the server. This can bring a huge communication burden to the server. On the other side, the support for asynchronous training and the elastic scalability are great advantages of PS. Although the PS architecture was originally designed for server-based machine learning, some of its features such as efficient communication are also strongly needed in client-based machine learning.

3.4.3 *Dataflow.*

Design of Dataflow. For distributed machine learning with model parallelism, a specially designed scheme called dataflow can be applied. Unlike the above-mentioned two schemes in which each node has similar functions for the whole task, in dataflow, different parts of the model are distributed on different machines, so their jobs vary from one to another. The whole computation process is organized using a directed acyclic graph. Nodes are units of the model, and edges describe how data flows. If data flows between two units that are stored on different machines, communication will take place. An example of this scheme is shown in Figure 2(a).

Pros/Cons. The disadvantage of this scheme is that the failure of any machine can cause the graph to be incomplete and the system can no longer run. If we use redundancy to solve this problem, backups for every machine are necessary. This may result in an expensive cost. Thus, the dataflow scheme is more suitable for cooperation among several powerful and stable machines.

3.4.4 *Discussion.*

From these techniques, we can see that server-based distributed training focuses on the cooperation between powerful machines. Its core idea is using data parallelism or model parallelism to solve large-scale learning tasks. The different communication schemes are designed for dealing with the low bandwidth of the local area network compared with the shared memory. However, in client-based training, the challenges are mainly due to the limited resources and the unstable network, which is quite different from those of server-based distributed training. Even if some techniques such as data parallelism and parameter server can be referred to for designing client-based training, we still have to pay attention to the special limitations in its scenario.

3.5 Distributed Optimization Algorithms

This kind of algorithm focuses on how to better manage a large scale of workers and how to use their resources to solve a huge complex task in a distributed way. A typical solution is to transform the original hard problem into much easier sub-problems. In this section, we introduce several distributed optimization algorithms. We list their key features in Table 3.

Table 3. Distributed Optimization Algorithms for Server-Based Distributed Training

Name	Math Tool	Objective	Convergence Speed	Experimental Scale
ADMM	Dual Decomposition, Method of Multipliers	Closed, Proper, Convex	Reach optimal as iteration number $k \to \infty$	Tested on 30GB data with 80 workers
DANE	Approximate Newton Method	Strongly Convex, Smooth	Linear convergence for quadratic objectives	Tested on CoverType [25], MNIST, and ASTRO-PH [45] with 64 workers
CoCoA	Uses dual variables to do efficient merge	Convex, Smooth	Convergence rate is $\frac{m-1+\Theta}{m}$ (m: number of workers, Θ: convergence rate of the local optimization method)	CoverType with 4 workers RCV1 with 8 workers ImageNet with 32 workers
CoCoA+	Uses dual variables to do efficient additive merge	Convex	Linear convergence for smooth convex objectives. Independent of the number of workers	CoverType with 16 workers RCV1 with 16 workers Epsilon with 100 workers
DiSCO	Inexact Damped Newton Method	Strongly Convex, Smooth, Self-Concordant	Linear convergence Communication rounds $t \approx O((md)^{\frac{1}{4}} \log(\frac{1}{\epsilon}))$ (m: number of workers, d: number of features)	Tested on CoverType, RCV1, and News20 [87] with 64 workers
Hydra	Randomized Coordinate Descent Method	Strongly Convex, Smooth	Roughly linear convergence Reach ϵ-accurate with possibility at least $(1-\rho)$ after $O(\frac{d}{m} \log(\frac{1}{\epsilon\rho}))$ iterations (m: number of workers, d: number of features)	Speedup tested on 3TB data with 512 workers. Convergence tested on ASTRO-PH with 32 workers

ADMM. Alternating Direction Method of Multipliers (ADMM) [15] makes use of both the decomposability of Dual Ascent and the superior convergence properties of Method of Multipliers. Machines distributedly use augmented Lagrangian methods to solve local sub-problems based on local data and alternately compute some shared variables to solve the global problem. The idea of this algorithm can be traced back to the mid-1970s and was used in distributed SVM training [30] in 2010. Although ADMM may take a large number of iterations to converge to high accuracy, it can usually reach modest accuracy within tens of iterations in practice.

DANE. Distributed Approximate NEwton (DANE) [96] is an approximate Newton-like method. In every iteration, each worker separately takes an approximate Newton step by implicitly using its local Hessian and makes two rounds of communication. In contrast to ADMM, DANE can benefit from the fact that sub-problems are often similar in applications of machine learning. DANE performs well on smooth and strongly convex problems. It is proved that DANE can achieve linear convergence if the learning rate is close to 1 and the approximation for Hessian is good.

CoCoA. Communication-efficient distributed dual Coordinate Ascent (CoCoA) was first proposed in 2014 [43]. By making use of convex duality, after dividing the task into approximate local sub-problems, we can choose whether to solve the primal sub-problem or the dual problem. The main advantage of CoCoA is its flexibility. It allows machines to choose a local optimization method and train the model to arbitrary accuracy. What's more, the trade-off between local computation and communication can also be tuned easily by setting a specific parameter. Experiments show that CoCoA can achieve up to a 50× speedup when dealing with problems like SVM, logistic regression, and lasso. Note that CoCoA [43] was improved to a more general case CoCoA' [106] in 2017 (CoCoA [43] and CoCoA+ [66] are predecessors of CoCoA' [106].). The convergence for non-smooth or non-strongly convex objectives has been analyzed for CoCoA'.

CoCoA+. CoCoA+ [66] also makes use of the primal-dual problem to get optimization. Compared with CoCoA, CoCoA+ additionally studies and proves the convergence on non-smooth loss

Table 4. Applicability of Distributed Optimization Algorithms to Client-Based Training

Name	Local Complexity	Comm. Overhead (per worker per round)	Scalability	Other Pro(s)/Con(s)
ADMM	A serially solvable convex problem	AllReduce(model)	Not theoretically analyzed Performs well in experiments	Tested with numerical experiments Implemented in C using MPI [108] ✓
DANE	Does mirror descent	3 · Sizeof(model)	Convergence rate is independent of the number of workers ✓	Performs well only for quadratic objectives ✗
CoCoA	Depends on local dual optimization method	2 · Numof(features) ✓	Slows down as the number of workers grows ✗	Allows steering the trade-off between communication and local computation ✓
CoCoA+	Depends on local dual optimization method	2 · Numof(features) ✓	Convergence rate is independent of the number of workers ✓	Allows steering the trade-off between communication and local computation ✓
DiSCO	Computes gradients and Hessians	Numof(features)^2 ✓	Slows down as the number of workers grows ✗	Requires self-concordant objectives✗
Hydra	Does coordinate decent	Not clearly analyzed. ✗	Speeds up as the number of workers grows ✓	Partitions dataset by features, requires redistribution of data ✗

functions. Linear convergence is proved for convex smooth objectives, while sub-linear convergence is proved for convex non-smooth objectives. Furthermore, to get rid of the slowdown caused by averaging updates, CoCoA+ chooses to add all updates. Experiments show that CoCoA+ only slows down a little as the number of workers increases, and it is faster than CoCoA for a large number of workers.

DiSCO. Distributed Self-Concordant Optimization (DiSCO) [124] is a Newton-type method. Compared with DANE, DISCO uses a distributed preconditioned conjugate gradient method to compute inexact Newton steps in each iteration and gets a superior communication efficiency. One significant advantage of DiSCO is that compared with other algorithms, it has fewer parameters to be paid attention to and adjusted. According to the experiments, when the number of workers increases to 16 and 64, DiSCO significantly outperforms ADMM and DANE on the convergence speed. DiSCO-S [67] is an improved version of DiSCO. It uses an approximated Hessian as its preconditioning matrix and uses the Woodbury Formula to deal with the linear system more efficiently.

Hydra. HYbriD cooRdinAte (Hydra) [88] is a randomized coordinate descent method. This kind of method is becoming popular in many learning tasks such as boosting and large-scale regression. In Hydra's design, the original data are partitioned and assigned to one node from the cluster of machines. Each node independently updates a random subset of its data based on a designed closed-form formula in each iteration. The updates are all parallelized.

3.6 Applicability of Distributed Optimization Algorithms to Client-Based Training

Compared with server-based distributed training, client-based distributed training is harder to implement due to the large number of workers and the relatively limited resources. Thus, some distributed optimization algorithms designed for server-based training may no longer be applicable to client-based training. In what follows, we discuss the applicability of existing distributed optimization algorithms to client-based training. As shown in Table 4, we investigate applicability mainly from three aspects: (1) **Local Complexity**, (2) **Communication Overhead**,

and (3) **Scalability**. They are key concerns in the context of client-based training. We also list some other pros and cons of these distributed optimization algorithms.

Local Complexity. Limited by the relatively poor computation resources (CPU and memory) available on mobile devices, client-based training is very sensitive to the local complexity of algorithms. However, in server-based training, local complexity has not been paid much attention to as the worker nodes are all regarded as powerful machines. We can hardly find accurate local complexity information in the above-mentioned distributed optimization work. Here, in Table 4, we just list the local computation methods that may reflect the local complexity to some degree.

In ADMM, DANE, DiSCO, and Hydra, the local optimization step is fixed. However, CoCoA and CoCoA+ allow multiple choices on the local solver (please refer to [106] for detailed suggestions), which means we can choose an appropriate local solver according to workers' available resources. This design is useful in client-based training since it improves the algorithm's compatibility for heterogeneous mobile devices. Thus, regarding the local complexity, CoCoA and CoCoA+ outperform the others.

Communication Overhead. Communication overhead is another important issue in client-based training. Considering the complex network condition of mobile devices, high communication overhead not only brings expensive data transfer cost but also increases the risk of transfer failure. Communication overhead can be further divided into the number of communication rounds and the amount of data transferred in each round. The number of communication rounds is reflected by the convergence speed listed in Table 3. Except for ADMM, which requires a large number of rounds to converge, the other algorithms can all achieve linear convergence.

The amount of transferred data per worker per round is listed in the Communication Overhead column in Table 4. In ADMM, the communication step is accomplished by using AllReduce to update the model parameters. In DANE, the amount of data transferred per round is triple the size of the model. Since the size of the model is not accurately provided, we cannot get the accurate communication overhead of ADMM and DANE. In CoCoA and CoCoA+, the amount of data transferred by each worker in each round is twice the number of the features. They are quite communication efficient when dealing with tasks with small feature space. In DiSCO, the amount of data transferred in each round is square the number of features, which may also be acceptable. Although several communication schemes have been provided in Hydra, the amount of transferred data has not been clearly analyzed. Regarding communication overhead, CoCoA, CoCoA+, and DiSCO outperform the other algorithms.

Scalability. Good scalability means we can assign the task to more workers and make use of more parallel resources. We compare the scalability of distributed optimization algorithms by checking if their convergence speed is affected by the number of workers. Results are listed in Table 4.

In ADMM, although the scaled version of the algorithm is given, the scalability is only discussed for the scale of the datasets. Whether the scale of workers has an impact on ADMM is not theoretically analyzed. We could only say that the performance of ADMM is not greatly changed as the number of workers grows according to the comparison experiments between DANE and ADMM done in [96]. In DiSCO and CoCoA, according to Table 3, the increase in the number of workers causes the slowdown of the convergence speed. This indicates that the scalability of DiSCO and CoCoA is not so good. In DANE and CoCoA+, the convergence rate is independent of the number of workers. In Hydra, according to Table 3, since it can reach ϵ-accurate with possibility at least $(1-\rho)$ after $O(\frac{d}{m}\log(\frac{1}{\epsilon\rho}))$ iterations, the number of iterations actually decreases as the number of workers m increases. This means Hydra speeds up as the scale of workers grows. Regarding scalability, DANE, CoCoA+, and Hydra outperform the other algorithms.

Other Pros/Cons. Besides the above three key aspects, which are mostly concerned with client-based training, these distributed algorithms also have some other pros/cons that should be incorporated for evaluating their applicability to client-based training. Details are presented in Table 4.

DANE and DiSCO require strict assumptions on the objective function. Hydra partitions training data by features, which means it may require a data redistribution step. Thus, DANE, DiSCO, and Hydra have additional cons. ADMM has been implemented using MPI in [108] and thus is easier to be applied. CoCoA and CoCoA+ allow steering the trade-off between communication and local computation, which gives us the freedom to change the training strategy according to available resources. For this part, we could say that ADMM, CoCoA, and CoCoA+ outperform the others.

Another problem is that all these distributed optimization algorithms require a synchronous update aggregation process in each communication round. Since the heterogeneity of mobile devices can easily cause the straggler problem, the synchronization may significantly degrade the system efficiency. This disadvantage should be carefully considered when applying distributed optimization algorithms to client-based training. Considering that some optimizers introduced in Section 2.5 have already supported the asynchronous scheme, we can refer to them and design new asynchronous distributed optimization algorithms for client-based training.

4 CLIENT-BASED INFERENCE

In contrast to server-based machine learning where all operations over the model (including training and inference) are conducted on servers, client-based machine learning intends to pull down some computation tasks to clients so that the quality of service can be improved in terms of response latency, personalization, security privacy, and so on. Based on the execution phase, we divide client-based machine learning into client-based inference and client-based training. Client-based inference focuses on the local execution of trained machine learning models. In this section, we discuss the necessity and the feasibility of client-based inference.

4.1 Motivations

Motivations for client-based inference can be concluded as follows: (1) client-based inference can reduce service latency and preserve user privacy, (2) client-based inference helps lower the cost of cloud platforms for service producers, (3) it is feasible to deploy client-based inference thanks to the rapid development of mobile chipsets, and (4) it is convenient to deploy client-based inference using the off-the-shelf mobile machine learning frameworks.

4.2 Challenges

One major challenge is that **(1) inference tasks using complex or large-scale models can hardly be solved on mobile devices.** Even though the power of mobile chipsets has grown quickly, it is still far away from that of high-performance servers. The CPU power and the memory capacity of mobile devices may have trouble supporting complex models, especially deep learning models.

Another challenge is that **(2) the trade-off between inference accuracy and resource consumption needs to be carefully decided.** This is a fine-grained resource control problem. As client-based inference aims at improving the quality of service, we should never let client-based inference take too many resources or drain the battery. Considering that multiple models may be run simultaneously to provide different services, the trade-off between inference accuracy and resource consumption must be carefully decided to prevent resource contention.

4.3 Current Advances

In the past few years, deep learning has become the main trend of machine learning. Mobile phones, as the most popular device in people's daily lives, become the most promising platform for deep learning. Ravi [85] has proposed a method to generate compact on-device neural networks, which brings convenience to the deployment of client-based deep learning inference. Deep-learning-based apps such as [57, 73, 122] have been built. General mobile deep learning frameworks [7, 10, 28, 35, 44, 110] have been developed. Here, we introduce several mobile deep learning applications to show the current development of client-based inference.

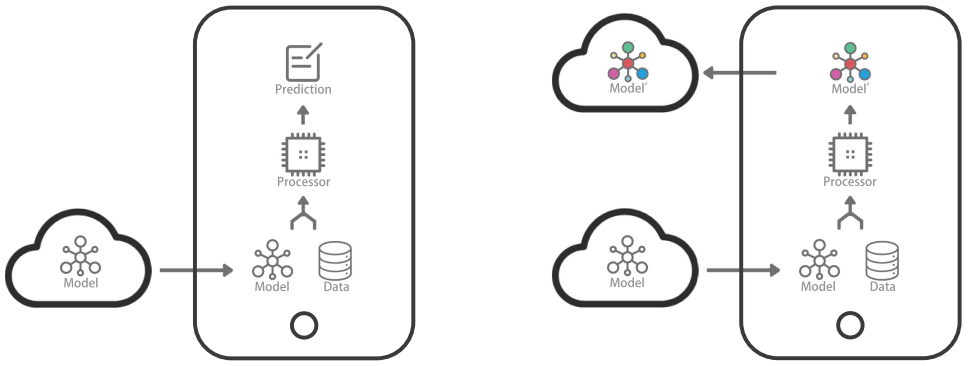
Mobile Computer Vision (CV). According to [117], photo beautify and face recognition together cover over half of all mobile deep learning applications. For CV applications, server-based inference can bring high privacy risk as the raw sensitive images and videos have to be transferred through the network. To solve this problem, DeepEye [68] and DeepMon [41] were proposed to support the on-device execution of deep vision models. Another problem is that the huge latency incurred by server-based inference may be unacceptable for applications such as mobile augmented reality. As a solution, DeepDecision [84] designs an on-device small deep learning model tiny-YOLO and automatically decides the inference to be done locally or remotely. By using optional client-based inference, the real-time capability of the system is guaranteed.

Mobile Natural Language Processing (NLP). Another hot topic is to solve NLP tasks such as sentiment analysis, translation, and question answering on mobile devices. To better preserve users' privacy as well as reduce the service latency, DeQA [18] adapts existing question answering systems to suit on-device running. Client-based inference is also applied to voice assistant and voice input applications to ensure that they can still function normally even if the device is offline. Siri, a voice assistant developed by Apple, uses on-device deep learning [19, 104] to improve the text-to-speech synthesis process. Google makes use of client-based inference to realize personalized speech recognition [69]. Georgiev et al. [32] use mobile GPU to accelerate mobile audio sensing.

Other Applications. Client-based inference also plays an important role in other applications that require real-time responses or involve sensitive data processing. Sicong et al. [102] designed UbiEar to do smartphone-based acoustic event sensing and notification. This work greatly improves the quality of life for Deaf or Hard-of-Hearing (DHH) people. Fang et al. [29] designed NestDNN to dynamically select the structure of on-device deep learning CV models based on the trade-off between accuracy and current available resources. In addition to these published works, client-based inference is also applied to various kinds of applications (e.g., recommendation, movement tracking, and identity recognition) without being noticed [117]. It is certain that people's daily lives have already been deeply permeated and improved by client-based inference techniques.

5 CLIENT-BASED TRAINING

Client-based training is mainly motivated by the strong need of making the best use of user data generated on mobile devices and protecting users' privacy at the same time. Since we can hardly find any work or research papers about client-based local training, for the rest of this survey, client-based distributed training is referred to as client-based training for convenience. In contrast to traditional machine learning that requires centralized datasets, client-based training uses mobile devices to solve machine learning problems according to local user data. The server aggregates all intermediate results and gets the trained model. Generally speaking, client-based training aims to transfer some computation tasks from centralized servers to decentralized mobile devices. It can not only offload servers' burden but also make use of the growing processing power on mobile devices. Furthermore, considering that data is processed locally in client-based training, the user



(a) Client-Based Inference (b) Client-Based Training

Fig. 5. Illustrations of inference and distributed training in client-based machine learning.

data that is either too much to be uploaded or too sensitive to be uploaded can now participate in machine learning tasks. This gives the possibility to improve the model's accuracy and accelerate the training process. More importantly, besides the benefits mentioned above, client-based training can even better preserve users' privacy. In this section, we first introduce and explain the motivations of client-based training in Section 5.1. We formally define the task in Section 5.2. Then we discuss the general constraints in Section 5.3. The main challenges faced by client-based training will be claimed in Section 5.4. Finally, we use federated learning and split learning as examples to show the current development of client-based training in Section 5.5 and Section 5.6.

5.1 Motivations

We conclude that motivations for client-based training are as follows: (1) client-based training keeps all advantages of client-based inference as client-based inference can be viewed as part of it; (2) in client-based training, the on-device model has a much shorter update cycle compared with server-based training and thus may perform better; (3) client-based training is able to preserve user privacy and make full use of on-device sensitive user data at the same time; and (4) some work in this direction has shown the feasibility and effectiveness of client-based training.

One problem faced by server-based training is that the cost (communication cost and computation cost) is high. In server-based training, user data is generated on users' devices and then collected. However, in many cases, the data is so fine-grained and much that it incurs huge communication overhead when being transferred to the server. In addition, processing huge amounts of data and running large-scale machine learning on the server is not only time-consuming but also costly. Even if the model is finally trained to good performance on the server, the long training cycle will result in the delay of the model, which can cause a decrease in its performance as users' behavior patterns may have already changed. Another problem in server-based training is the privacy risk. Specifically, the server in both centralized and distributed machine learning frameworks requires direct access to training data and thus needs to store raw user data, which inevitably suffers outsider and insider attacks [79–81]. For example, a malicious hacker may invade the datacenter, compromise part of the server, and leak private databases. Further, if the server is untrusted, it may share user data with unauthorized entities or even trade for profits. To address these problems and disadvantages of server-based training, attempts at client-based training have been made.

The difference between client-based inference and client-based training is shown in Figure 5. From the figure we can see that the client-based inference process just returns a prediction result

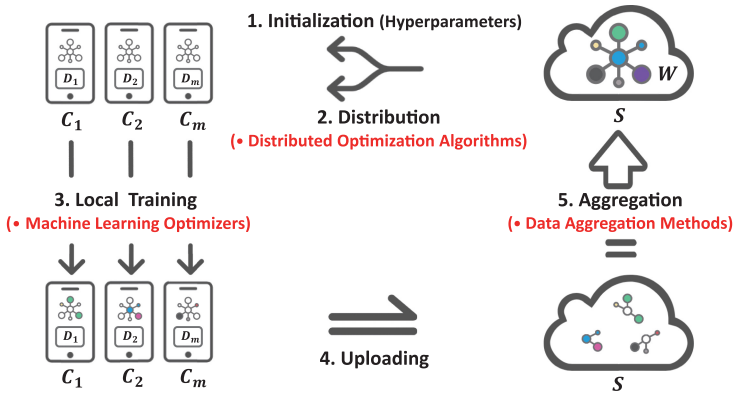


Fig. 6. Workflow of client-based training.

that may be used to produce a service for the user locally. It has nothing to do for the server. However, if some part of the training process can be additionally done on-device, the generated trained model can not only become personalized but also provide valuable information for the global model stored on the server if the update of the local model is uploaded. This is exactly the idea of client-based training. As an example of client-based training, federated learning has already shown its effectiveness, which will be introduced later in Section 5.5. Besides federated learning, some other works that are closely related to client-based training have also been proposed. He et al. [38] designed a framework for decentralized on-device linear learning. Koloskova et al. [51, 52] further studied decentralized deep learning with compressed communication. They together show us the feasibility of on-device client-based training.

5.2 Task Definition

We first introduce some notations and definitions in client-based training: (1) A real matrix $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$ is the model learned from decentralized data. (2) A list $\{C_1, C_2, \ldots, C_m\}$ contains all m clients that are run on different mobile devices. They are also known as the workers in client-based training. (3) A list $\{D_1, D_2, \ldots, D_m\}$ contains all local datasets used by the clients. (4) A server S communicates with clients and arranges their tasks. S can either be a single computer or a server cluster whose structure is transparent to the clients.

5.2.1 Workflow. As shown in Figure 6, a general workflow of client-based training consists of five phases: (1) *Initialization:* The workers $\{C_1, C_2, \ldots, C_m\}$ run on mobile devices download hyperparameters from the server S and get themselves initialized. (2) *Distribution:* The server S arranges the tasks and sends them to the workers $\{C_1, C_2, \ldots, C_m\}$. The model to be trained should also be distributed to the workers in this step. (3) *Local Training:* The workers $\{C_1, C_2, \ldots, C_m\}$ train the local models based on the local datasets $\{D_1, D_2, \ldots, D_m\}$. Updates will be generated after a certain number of local training iterations. (4) *Uploading:* The workers $\{C_1, C_2, \ldots, C_m\}$ upload the updates to the server S. (5) *Aggregation:* The server S aggregates all updates to generate a final update and applies it to the global model $\mathbf{W}$. A single round of learning ends here. Go back to step 2 to start a new round of learning. A similar workflow has been adopted in the federated learning protocol [12].

5.2.2 System Modules. To accomplish client-based training, the system is made up of three modules: local training module, communication module, and aggregation module. We show how they cooperate with each other and what kinds of techniques can be used to improve them.

The local training module is implemented on the worker. It works in step 3. Besides the data preprocessing part, the core component of the local training module is the local machine learning optimizer. Considering that the resources available for client-based training are very limited, the local optimizer must be chosen and designed carefully. It should be run on mobile devices without a significant bad effect on users' daily usage experience. For the local training module, the input is the initialized model and the output is the update. The update can be generated based on either the new model that has been trained on the local dataset or the difference between the two models.

The Communication module takes part in step 1, step 2, and step 4. It is implemented on both the server and the worker, which means the server and the worker have to cooperate with each other for communication. The communication module is responsible for two jobs: task arrangement and data transfer. For the server, it should be concerned with the task arrangement part whose main purpose is to divide the original complicated problem into easier sub-problems. This can be done through distributed optimization algorithms. The input is the original problem and the output is a set of sub-problems. From the worker's perspective, data transfer is the main challenge as mobile network resources are limited and expensive. Compression methods can be used to reduce the communication cost. Here, the input is the original data and the output is the compressed data.

The aggregation module is implemented on the server. It works in step 5. Since the number of updates received from workers can be very large, it is inefficient and even impossible to apply all updates to the global model one by one. Thus, the main job of the data aggregation module is to extract and make the best use of the information contained in the updates. It takes all received updates as inputs. Its output can be either a final update that can be directly applied to the global model or a new model that replaces the old one.

5.3 Constraints

The only difference between server-based training and client-based training is where the training process is done. However, this change introduces additional constraints for the whole system due to the limitations on mobile devices: (1) compared with PCs and servers, the hardware resources available on mobile devices are poor and limited; (2) the network condition for mobile devices is unstable, and the communication cost is very high; and (3) unlike servers, which are owned and managed by data centers, mobile devices are not totally under control, and thus the reliability and the security of the whole decentralized training process are not guaranteed.

The first limitation is similar to the first challenge of client-based inference as both of them require plenty of computation. The second limitation is introduced considering that client-based training additionally needs the communication process to accomplish distributed training. The third limitation indicates that client-based training is more vulnerable to attacks. In what follows, we show how these three limitations lead to six concrete constraints of client-based training.

Simple On-Device Task. The machine learning sub-problems solved on mobile devices should be easy. This constraint is desired according to the first limitation. Although the development of mobile devices has kept accelerating in recent years, their computation power is still far less than that of personal computers, not to mention the server clusters. The key reason for this phenomenon is that mobile devices need to be portable. In this situation, chipsets are specially designed for mobile devices to balance between the need for lower energy consumption and the need for greater power. Thus, unlike the CPUs used on personal computers that can easily reach 100W in power with stable power supply, the peak power of CPUs used on mobile devices is usually under 10W. Considering that mobile devices have to spare some resources and energy to support their basic functions, the part that can be used for supporting mobile machine learning becomes even less. Moreover, the mobile devices usually have no additional cooler and just use passive cooling

methods. This also limits the maximum power of them. Considering the facts given above, the necessity of the simple on-device task constraint has been clear. First, mobile devices cannot handle complex machine learning tasks because the power is limited. Second, since only passive cooling methods are used, a hard machine learning task is very likely to cause the mobile device to become hot. This must not happen as it has a huge negative effect on user experience. Therefore, the machine learning sub-problems solved on mobile devices should be easy enough so that the users are unaware of its running at all.

Short Training Time. The time spent on one round of local on-device training should be short. We have mentioned that the hardware resources available on mobile devices are poor and limited compared with personal computers. Thus, the systems run on mobile devices are specially designed. Background apps may be paused by the system to save resources for the foreground app. Considering that most users just use an app for a little while at one time, it is better to finish the training task during the time that the app is running in the foreground. Otherwise, if the training has not been finished when the app is paused or quit, the generated update may not be sent to the server in time and may become useless at the next start due to the long delay.

No Uploading. Local private user data should not be uploaded to the server in principle. For now, although the 4G network has already been deployed widely and the 5G network is coming, the cost of data on mobile devices is still expensive. What's more, local data on mobile devices may contain sensitive user information, which means it is not suitable for being uploaded to the server. Thus, to best preserve user privacy, the local raw user data should not be uploaded in principle. However, considering that uploading a small subset of local data is feasible and may be helpful for the aggregation process on the server, it can be permitted under special situations with guaranteed privacy. Note that *No Uploading* is one important cause of the significant difference between server-based distributed training and client-based training. Without uploading data, the local datasets used for training are specific to their owners and can never become IID, which is a necessary condition for many machine learning algorithms. Moreover, *No Uploading* also causes the sizes of local datasets to be unbalanced and results in the information contained in different workers' updates being unbalanced. Hence, how to determine which updates are more valuable (contains more information) becomes another problem to be solved.

Low Communication Overhead. The data transferred in one round of learning should not be too much. This constraint also aims at protecting the experience of users' daily usage. If too much data has to be transferred, it may be a heavy burden for the mobile devices even if they are connected to WLANs and can cause other apps that are using the network to be blocked.

Low Communication Frequency. The communication frequency should be low and it is better to let the workers decide when to communicate. As the network condition for mobile devices is unstable, a scheme with frequent communication is unsuitable. What's more, a low communication frequency with a long training time is beneficial for reducing the communication cost. The reason we suggest letting the workers decide when to make the communication is that the high heterogeneity of workers makes the server have no idea when the sub-problems will be solved. By letting workers make the decision, we may also reduce the communication cost as workers can choose to do the communication at the time when they are connected to WLAN.

High Awareness of Data Reliability. The server should be highly aware of the data reliability of the received updates. Since the decentralized training scheme cannot guarantee the reliability of the received data, client-based training suffers from fake updates or low-quality updates. To prevent machine learning poisoning attacks, a possible solution is to let the server evaluate the data

reliability by making use of update diversity or additional information such as worker reputation. With high awareness of data reliability, the server can distinguish and reject low-quality updates and make the training process become more secure and stable.

5.4 Challenges

From the above six constraints, we can summarize the major challenges of client-based training: (1) transforming original machine learning tasks to easier sub-problems that can be solved on mobile devices effectively and efficiently; (2) dealing with the unbalanced non-IID dataset and covering its negative effects; (3) compressing the transferred data between the server and the clients; (4) lowering communication frequency and overhead; and (5) ensuring the efficiency and security of data reliability evaluation.

For challenge (1), although we can refer to distributed optimization algorithms, none of them guarantees that it performs well with millions or even billions of workers. The huge number of workers and the limited resources available to workers may cause the optimization problem to become so difficult that we must design a brand new solution for it.

For challenge (2), some existing methods (e.g., [125]) use data-sharing strategies to mitigate the negative effect of the unbalanced non-IID dataset. The problem of data-sharing strategies is that raw data transfer can be costly and risky for mobile devices. Another possible solution is to design new machine learning algorithms that are insensitive to the data distribution. For example, semi-cyclic SGD [3] can adapt to data with a block-cyclic pattern.

Regarding challenges (3) and (4), they have already been widely studied in other areas (e.g., data compression and computer network). However, considering that general methods normally may not well adapt to every scenario, specific algorithms that are closely combined with client-based training must be more effective and thus are needed urgently.

For challenge (5), to prevent model poisoning attacks [9, 31, 98], some existing studies [31] use update diversity to distinguish low-reliability updates. Some others [48, 49] choose to calculate the reputation of workers and use the reputation score to represent the data reliability. Techniques such as blockchain have been used to achieve secure reputation management. Although these methods can effectively protect the global model training process from the attacks, they incur large computational cost as the number of workers increases. Thus, the main task here is to design an algorithm that can guarantee both security and efficiency.

To better learn about these challenges and their possible solutions, we use federated learning and split learning as two examples and introduce their current advances.

5.5 Current Advances in Federated Learning

In recent years, attempts at the implementation of client-based training have occurred. Federated learning, as an example, was proposed by researchers from Google in 2015. Generally speaking, federated learning means to do machine learning tasks federatedly among a large number of mobile devices with on-device private data protected. Its core idea is to train the model locally on user devices and aggregate updates on the server without uploading raw user data, which shares a similar scheme of distributed machine learning. From 2016 to 2018, Google published several related articles to complement federated learning's framework. In 2019, applications of federated learning appeared. A detailed survey [47] about federated learning was given in December 2019. Federated learning can be viewed as an attempt at client-based training with privacy preservation as the primary goal. Existing federated learning work mainly focuses on studying how to improve the final model's performance and how to design an efficient communication scheme. In this section, problems and current advances in federated learning are introduced and discussed.

5.5.1 Model Performance.

Literature Review. The idea of federated learning can be traced back to Distributed Selective Stochastic Gradient Descent (DSSGD) [100]. It was published in October 2015 and is about distributed deep learning without sharing datasets.

Following the idea of distributed learning and privacy preservation, in November 2015, researchers from Google submitted their first attempt on federated learning [54]. Federated learning can be viewed as an improved version of DSSGD that is optimized for mobile devices. In this work, three basic properties of federated learning's scenario were given: (1) *Non-IID Data*, (2) *Unbalanced Data*, and (3) *Massively Distributed Data*. (Note that the fourth property (4) *Limited Communication* was introduced in [70] and will be discussed in Section 5.5.2.) Also, this work proposed an efficient federated optimization algorithm called Distributed Stochastic Variance Reduced Gradient (DSVRG) based on SVRG [46] and DANE [96].

Later in 2016, Konečnỳ et al. [55] complemented and improved the above work. DSVRG was renamed as Federated SVRG (FSVRG). Equations and mathematical proofs for it were also provided.

After the above works had formulated the basic training scheme of federated learning, researchers continued to study the effectiveness of federated learning and tried to improve its performance. Zhao et al. [125] analyzed the negative effect of non-IID datasets on model performance and provided a simple data-sharing strategy to deal with it. Yu et al. [121] studied why model averaging works for deep learning tasks. Eichner et al. [27] discovered that cyclic patterns in the data samples are hard to avoid in federated learning and does harm to the performance of SGD. They proposed Semi-Cyclic SGD to correct this problem when optimizing convex objectives. Mohri et al. [74] proposed a new framework of agnostic federated learning to avoid the federated model being biased towards different clients. Chen et al. proposed FedMeta [21], which combines federated learning with meta-learning. To better evaluate various kinds of federated learning algorithms, LEAF [17], a modular benchmarking framework for learning in federated settings, was proposed. It contains open-source federated datasets and is continuously updated.

Discussion. While federated leaning training algorithms have developed from DSSGD to FSVRG, the most important non-IID data problem has not been solved well yet. What's more, the cyclic pattern of data is also harmful and should be taken into consideration. In conclusion, to further improve the model performance of federated learning, we need to deal with the *cyclic unbalanced non-IID data* problem under the *non-convex objective* condition.

5.5.2 Communication Efficiency.

Literature Review. The first aspect of improving communication efficiency is reducing communication rounds. In [70], Federated Averaging (FedAvg) was proposed to deal with the fourth basic characteristic of federated learning, *Limited Communication*. By enabling multiple local training iterations and using model averaging methods, the number of communication rounds is reduced. Similarly, based on parallel restarted SGD [121], Yu et al. [120] proposed parallel restarted SGD with momentum to enlarge local training steps and thus reduce the total number of rounds.

Meanwhile, reducing the size of communication data also helps improve communication efficiency. According to [56], as on most occasions the bandwidth of the uplink is much poorer than that of the downlink, reducing the uplink communication cost is a more urgent task. Two kinds of approaches (*Structured Updates* and *Sketched Updates*) that can lower the size of the updates are implemented and tested in federated learning in [56]. Besides the general compression methods, Caldas et al. [16] proposed Federated Dropout. With Federated Dropout, each worker trains a smaller sub-model instead of the whole global model. Then the size of updates is also reduced.

Discussion. Although the above-mentioned methods do improve the communication efficiency, they are still using the two-tier server-worker communication structure. This scheme not only

brings a huge communication burden to the central server but also suffers from the instability of mobile workers' network. Perhaps we can consider using a multi-tier communication structure to further improve both communication efficiency and communication stability.

5.5.3 Security & Privacy.

Literature Review. After the development of FedAvg [70], to better ensure the security of the aggregation process in federated learning, Google proposed a practical secure aggregation method [13] for privacy-preserving machine learning. In the secure aggregation process, the secure multi-party computation technique is used to compute sums of model parameter updates. What's more, model poisoning attacks [9, 31, 98] toward federated learning have also been studied. The main purpose of model poisoning is to make the trained model output wrong answers or even attacker-chosen answers. To prevent model poisoning attacks, update diversity [31] and worker reputation [48, 49] can be used to recognize unreliable updates.

For better privacy preservation, McMahan et al. [71] applied differential privacy methods to FedAvg and this only resulted in a negligible cost in inference accuracy. Agarwal et al. [3] proposed cpSGD to achieve both differential privacy and communication efficiency in federated learning settings. Chaoyue et al. [20] proposed a secure federated sub-model learning scheme with tunable property that enables the workers to tune privacy and utility. Considering that the maximum contribution is an important parameter for differential privacy algorithms (the noise to be added to data is closely related to it), Amin et al. [5] studied how to bound user contributions in federated learning. Cheng et al. [23] proposed a novel lossless privacy-preserving tree-boosting system called SecureBoost.

Discussion. Regarding better privacy, although federated learning can naturally protect sensitive raw user data, we still need to make sure that the attacker cannot infer information from the transferred updates. For the security part, since federated learning distributes the training process to unreliable mobile devices, attacks that aim at misleading the final model can be more easily implemented. For example, we should pay more attention to the potential data poisoning attacks.

5.5.4 Applications. Federated learning has already been used in some applications and has shown satisfying performance. Google first applied federated learning to Gboard to improve its query suggestions [119]. They tested federated learning on 100 clients. Results show that federated learning does improve the performance of the deployed LSTM model. After that, they also applied federated learning to the mobile keyboard next-word prediction task [37]. According to the evaluation done on server-hosted logs data, the federated-trained CIFG model performs nearly as well as the centralized-trained CIFG model. And for evaluation done on client-owned data caches, federated learning even outperforms centralized learning. Google has also applied federated learning to learn Out-of-Vocabulary (OOV) words [22]. This work conducted both simulated federated learning on a non-IID dataset and real-word federated learning on data hosted on user mobile devices. Results show that the federated learning method can learn OOV words effectively. Google introduced their system design for federated learning at scale in [12], which mainly focus on how to design an elastic parameter server to support a large number of clients in real-word settings.

Besides Google, many other researchers were also exploring federated learning. Smith et al. [105] combined federated learning with multi-task learning. Intel [99] used federated learning to do multi-institutional deep learning for brain tumor segmentation without sharing patient data. Liu et al. [64] proposed Federated Transfer Learning (FTL). This work shows that federated learning is also suitable for being applied to machine learning tasks in the scenario of cooperation among banks where sensitive information mustn't be shared. Sattler et al. [93] proposed Sparse Ternary Compression (STC). STC is more robust to non-IID datasets and works as a substitute for FedAvg.

5.5.5 Surveys. Google has published a very detailed survey paper [47] to summarize all works and research related to federated learning. Advances of federated learning and open problems in this area have also been discussed. Yang et al. provided a survey [118] about federated learning's concept and applications. In this survey paper, federated learning is extended and classified into Horizontal Federated Learning (HFL) and Vertical Federated Learning (VFL). Li et al. [62] write a survey to discuss challenges, methods, and future directions of federated learning.

5.6 Current Advances on Split Learning

Just like federated learning can be viewed as a special case of data-parallel distributed learning where datasets are distributed on mobile workers, split learning can be considered as a special case of model-parallel distributed learning where the model is distributed on the server and mobile workers. Here, we continue to introduce current advances in split learning.

5.6.1 Model Performance.

Literature Review. The idea of split learning first appeared in [36] and was proposed by a research team from MIT. It is motivated by the need to use multiple agents to collaboratively train a deep neural network without transferring raw sensitive user data, which is very similar to the motivation of federated learning. To achieve this goal, Split Neural Network (SplitNN) was designed. In SplitNN, only the bottom parts of the model are trained on a worker who owns the raw data. The gradients together with the labels for the training data are transmitted to the server, which accomplishes the rest of the training process for the top parts of the model. Considering that the labels may also reveal sensitive user information, U-shaped SplitNN is designed. In U-shaped SplitNN, the server only processes the middle layers of the model. The bottom layers and the top layers are stored on workers to process the raw data and the corresponding labels. Experiments have been done to demonstrate the effectiveness of SplitNN.

Later, the concept of split learning was formally proposed in [113]. SplitNN, as a method of split learning, has been improved in this work. Several possible privacy-preserving structures of SplitNN have been provided and discussed. We will introduce this part later in Section 5.6.3. By comparing SplitNN with large-scale SGD and FedAvg on CIFAR 10 and CIFAR 100, it is shown that SplitNN can achieve higher validation accuracy with much less computation.

Discussion. In SplitNN, although a training algorithm using multiple workers has been designed, the whole training process is still run sequentially. According to the algorithm, a worker must fetch the latest model before it starts training. After a training iteration, this worker's updated model will be marked as the latest model. This scheme implies that only one worker can train the latest model at the same time. Although Singh et al. [103] proposed an approach called "split learning without any client weight synchronization," how it works and how it performs are not described. On the other hand, since there is no model aggregation process, multiple workers training the latest model in parallel will result in conflicts. This lack of parallel training scheme makes SplitNN fail to get benefit from parallel training acceleration. Moreover, while local raw data is not transferred in split learning, the impact of the local non-IID dataset has not been analyzed yet.

5.6.2 Communication Efficiency.

Literature Review. A detailed comparison of communication efficiency of split learning and federated learning was given in [103]. The theoretical analysis shows that split learning becomes more communication efficient as the number of clients increases and can well adapt to big models. One shortcoming of this work is that the theoretical analysis has not been validated by experiments.

Discussion. The communication efficiency of split learning has not been well studied yet. As workers need to transfer data with the server in each training iteration, the network condition (e.g.,

bandwidth and latency) can greatly influence the system efficiency. What's more, the applicability of communication compression methods needs to be further examined.

5.6.3 Security & Privacy.

Literature Review. Attacks on split learning have been studied in [2]. This work shows that it is possible to reconstruct the raw data from the worker's outputs if only a few convolution layers are trained on the workers. In addition, neither introducing additional hidden layers nor applying differential privacy to the split layer can effectively mitigate this shortcoming. Both of the two attempts cause an unacceptable decrease in the model accuracy.

Several privacy-preserving structures of SplitNN have been introduced in [113]. As multiple institutions might own different modalities of the same user's data, vertical split learning is designed to deal with this kind of data, which is partitioned by features. Extended vanilla split learning and Tor-like multi-hop split learning arrange additional workers to process the middle layers of the model. This structure helps to cover the identities of the bottom workers, who use their sensitive raw data to train the bottom layers, and thus the privacy of the bottom workers is better preserved by the anonymity. Besides the basic structure design, Vepakomma et al. [112] used two losses in one model to reduce data leakage for SplitNN. Sharma et al. [97] proposed ExpertMatcher to do model matching for split learning with only the encoded hidden representation of local raw data shared. By hiding the raw data representation, user privacy is better preserved.

Discussion. One important problem not studied yet is how to trade off between on-device model complexity and user privacy. By putting some complex layers of the model on workers, the privacy may be better preserved as it becomes harder to infer raw data. Meanwhile, split learning also suffers model poisoning attacks. Now that some parts of the model do not even exist on the server, model poisoning attacks become much easier and should be more carefully handled.

5.6.4 Applications. Split learning has been tested in the medical field in [78]. U-shaped SplitNN has been implemented to enable the collaborative machine learning between several hospitals. Experiments are done on two medical datasets: retinal fundus photos and chest X-rays. Results show that split learning outperforms non-collaborative methods greatly.

5.6.5 Surveys. Vepakomma et al. [114] have surveyed methods for deep learning without revealing raw data, including large-batch SGD, federated learning, and split learning. Key ideas, limitations, and future trends of them have been simply discussed. In addition, this survey article has introduced several cryptographic techniques that can be used to further preserve privacy in the machine learning area, including homomorphic encryption, oblivious transfer, and garbled circuits.

6 FUTURE DIRECTIONS

In this section, we introduce some potential research directions of client-based training. Most of them are motivated by the problems and challenges discussed in Section 5 that have not been studied well (e.g., Non-IID Training Sets). Others are ideas that can improve the robustness or the adaptivity of the system (e.g., General Mobile Training Framework).

Non-IID Training Sets. This problem was first introduced in federated learning. It also exists in the scenario of other client-based training methods such as split learning as we can no longer upload the original data and do a shuffle. Zhao et al. [125] have shown the negative effect of non-IID datasets on model convergence and proposed a data-sharing strategy to deal with it. However, it may be difficult for mobile devices to share a small part of the local dataset with others because the process is hard to manage and the communication cost can be high. We also have to ensure that the privacy is still preserved during the whole data transmission process. Another possible

solution is to develop new machine learning algorithms that are not sensitive to the distribution of the training set. However, this direction does not have much existing work that can be referenced and may become a new hard machine learning problem.

Aggregation Methods. For existing data aggregation methods designed for server-based distributed training (e.g., FedAvg [70], Ensemble-Compression [109], and Codistillation [6]), none of them guarantees a fast convergence. Since the number of workers is usually under 100 in server-based distributed training, no one has the experience to aggregate tens of thousands of updates in a round in client-based training. Moreover, the above-introduced methods all have disadvantages. The averaging-based methods may cause a significant decrease in the convergence speed as they reduce the scale of updates on weights. They may also not be suitable for non-convex problems. The distillation-based methods may not be able to merge tens of thousands of models because they need much additional computation to handle this complex task. Thus, how to design an effective, efficient, and robust aggregation method is an urgent problem to be solved.

Security & Privacy. Although client-based training can prevent the leakage of raw user data, Shokri et al. [101] have already shown the feasibility of the attack against machine learning models. This work demonstrated how to inference user membership only based on the trained model. Differential privacy, as a solution to this problem, requires additional computation on mobile clients and can cause a decrease in model accuracy. Secure aggregation [13] does no harm to the model performance but can hardly be deployed in large-scale client-based training due to its high complexity. On the other hand, malicious clients may adjust their update data in order to affect or control the behavior of the final model and gain benefits for themselves [9]. These facts imply that better privacy and security in the scenario of client-based training is still an open problem.

Communication with 5G. The fifth-generation mobile network (5G) brings a high-bandwidth and low-latency network to mobile devices. The training slow-down caused by network latency and the client drop-off caused by network instability can be greatly relieved by 5G. Since communication efficiency is a key concern for the implementation of client-based training, the high bandwidth, low latency, and good stability of 5G may help client-based training become more robust [65].

While current client-based training methods are commonly using a server-client communication scheme (e.g., parameter server), 5G provides us with the opportunity to extend the communication scheme to device-to-device (D2D) and vehicle-to-vehicle (V2V). Some work (e.g., [91]) has already tried to combine federated learning with V2V networks to achieve low-latency neighbor cooperation. We can also use the 5G D2D network [94] to add middle layers between the server and the clients to realize a multi-tier network structure that is more robust and scalable.

What's more, as 5G has brought stronger connectivity to mobile devices, a secure data utilization strategy becomes an urgent problem. Thus, besides being improved by 5G techniques, client-based training has also been used to deal with 5G problems such as the Network Data Analytics Function (NWDAF) [42, 75]. These facts show that combining client-based learning with 5G is beneficial to both sides and thus will be an important future direction.

Standardization & Benchmark. Up till now, there still did not exist a white paper to comprehensively define and claim the standard of client-based training. Since deploying client-based training requires a trade-off between a lot of properties (e.g., model accuracy, communication overhead, complexity, privacy, and the scale of supported clients), the comparison between different client-based training algorithms will be difficult without pre-defined standards and evaluation metrics. Moreover, as client-based training can be applied to various kinds of tasks, it is also important to collect and release corresponding benchmark datasets that satisfy the client-based training settings. We recall that the *No Uploading* constraint in client-based training forces the

Table 5. Available Datasets for Client-Based Training in LEAF

Name	#samples	#users	#samples per user		Task
			mean	stdev	
FEMNIST	805,263	3,550	226.83	88.94	Image Classification
Shakespeare	4,226,158	1,129	3,743.28	6,212.26	Sentiment Analysis
Twitter	1,600,498	660,120	2.42	4.71	Next-Character Prediction
CelebA	200,288	9,343	21.44	7.63	Image Classification
Reddit	56,587,343	1,660,820	34.07	62.9	Next-Word Prediction
Synthetic Dataset	107,553	1,000	107.55	213.22	Classification

Table 6. Mobile Machine Learning Frameworks

Name	(Server-Based) Centralized	(Server-Based) Distributed	Mobile Inference	Mobile Training
TensorFlow Lite [35]	TensorFlow ✓	TensorFlow ✓	✓	✗
Core ML [7]	Create ML ✓	✗	✓	✗
PyTorch Mobile [28]	PyTorch ✓	PyTorch ✓	✓	✗
NCNN [110]	✗	✗	✓	✗
Paddle Lite [10]	Paddle ✓	Paddle ✓	✓	✗
MNN [44]	MNN ✓	✗	✓	✓

local datasets to be unbalanced non-IID datasets that are partitioned by users. LEAF [17] can be taken as a reference. Table 5 has listed the currently available datasets in LEAF. All these datasets allow partition by user, which means they can be used to test client-based training algorithms under the non-IID condition. For now, LEAF datasets have covered only a few tasks. More benchmark datasets are needed to support the development of client-based training in different areas.

Deployment Scenarios. Since both federated learning and split learning are general client-based training frameworks that are concerned about the overall learning process and scheme, they have to be combined with concrete machine learning methods and models when being applied to real-life applications. However, as they are newly emerging techniques and haven't been tested on many tasks, the suitable deployment scenarios for any of them are still not clear. For now, client-based training has caught people's eyes because it can guarantee user data privacy by sacrificing a little model performance and training speed. That is the reason federated learning and split learning have been applied to applications whose user data is sensitive (e.g., Gboard) and the health area. Considering that client-based training is experiencing rapid development with its performance, efficiency, and robustness all being improved, it is urgent to figure out what else client-based training can do and how to better deploy it on mobile devices in various kinds of real-life scenarios.

General Mobile Training Framework. We list some commonly used mobile machine learning frameworks in Table 6. Although the computation power of mobile devices is already sufficient for training small models, many existing open-source mobile machine learning frameworks such as TensorFlow Lite [35] and PyTorch Mobile [28] still only support inference operations, which means they are actually mobile *inference* frameworks. Without a general mobile training framework, implementing client-based training on mobile applications can be inefficient and time-consuming because developers have to realize all training operations by themselves for each task. To deal with this problem, MNN [44] from Alibaba has provided an on-device training module. MNN

supports constructing a model from zero and training it totally on mobile devices. We hope that other mobile machine learning frameworks can also add support for mobile on-device training.

7 CONCLUSION

In this survey, we have provided a thorough overview of the development of machine learning in recent years, from traditional server-based machine learning to emerging client-based learning. We have discussed their purposes and demonstrated the sufficiency and necessity of client-based machine learning. Specifically, for client-based inference, we have discussed its challenges and demonstrated its current advances, especially in the fields of computer vision and natural language processing. In addition, for client-based training, we have illustrated motivations and bottlenecks, given a clear task definition, and further offered a general guideline for practitioners. As typical examples of client-based training, we have introduced the concepts of federated learning and split learning and also reviewed their current advances. We finally have pointed out some future research directions of client-based machine learning in both academia and industry. In summary, applying client-based machine learning to real-world industrial applications is still faced with many challenges and opportunities, which calls for more attention to be paid to its future development. We hope that this survey can be a good starting point.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Retrieved from https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Sharif Abuadbba, Kyuyeon Kim, Minki Kim, Chandra Thapa, Seyit A. Camtepe, Yansong Gao, Hyoungshick Kim, and Surya Nepal. 2020. Can we use split learning on 1D CNN models for privacy preserving training? *arXiv preprint arXiv:2003.12365* (2020).

[3] Naman Agarwal, Ananda Theertha Suresh, Felix Xinnan X. Yu, Sanjiv Kumar, and Brendan McMahan. 2018. cpSGD: Communication-efficient and differentially-private distributed SGD. In *Advances in Neural Information Processing Systems 31 (NeurIPS'18)*. Curran Associates, Inc., 7564–7575.

[4] Amr Ahmed, Moahmed Aly, Joseph Gonzalez, Shravan Narayanamurthy, and Alexander J. Smola. 2012. Scalable inference in latent variable models. In *Proceedings of the 5th ACM International Conference on Web Search and Data Mining (WSDM'12)*. 123–132.

[5] Kareem Amin, Alex Kulesza, Andres Munoz, and Sergei Vassilvtiskii. 2019. Bounding user contributions: A bias-variance trade-off in differential privacy. In *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, Vol. 97. 263–271.

[6] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E. Dahl, and Geoffrey E. Hinton. 2018. Large scale distributed neural network training through online distillation. *arXiv preprint arXiv:1804.03235* (2018).

[7] Apple. 2017. Core ML. Retrieved January 20, 2020, from https://developer.apple.com/documentation/coreml.

[8] Haim Avron, Alex Druinsky, and Anshul Gupta. 2015. Revisiting asynchronous linear solvers: Provable convergence rate through randomization. *Journal of the ACM (JACM)* 62, 6 (2015), 51.

[9] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2018. How to backdoor federated learning. *arXiv preprint arXiv:1807.00459* (2018).

[10] Baidu. 2019. Paddle Lite. Retrieved January 20, 2020, from https://github.com/PaddlePaddle/Paddle-Lite.

[11] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer.

[12] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konecný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046* (2019).

[13] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*. 1175–1191.

[14] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. 2018. Optimization methods for large-scale machine learning. *Siam Review* 60, 2 (2018), 223–311.

[15] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning* 3, 1 (2011), 1–122.

[16] Sebastian Caldas, Jakub Konečny, H. Brendan McMahan, and Ameet Talwalkar. 2018. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210* (2018).
[17] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečnỳ, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097* (2018).
[18] Qingqing Cao, Noah Weber, Niranjan Balasubramanian, and Aruna Balasubramanian. 2019. DeQA: On-device question answering. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'19)*. 27–40.
[19] Tim Capes, Paul Coles, Alistair Conkie, et al. 2017. Siri on-device deep learning-guided unit selection text-to-speech system. In *Proceedings of the 18th Annual Conference of the International Speech Communication Association (INTERSPEECH'17)*. 4011–4015.
[20] Niu Chaoyue, Wu Fan, Tang Shaojie, Hua Lifeng, Jia Rongfei, Lv Chengfei, Wu Zhihua, and Chen Guihai. 2020. Billion-scale federated learning on mobile clients: A submodel design with tunable privacy. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom'20)*. 405–418.
[21] Fei Chen, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. 2018. Federated meta-learning for recommendation. *arXiv preprint arXiv:1802.07876* (2018).
[22] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. 2019. Federated learning of out-of-vocabulary words. *arXiv preprint arXiv:1903.10635* (2019).
[23] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, and Qiang Yang. 2019. SecureBoost: A lossless federated learning framework. *arXiv preprint arXiv:1901.08755* (2019).
[24] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. 2012. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25 (NeurIPS'12)*. Curran Associates, Inc., 1223–1231.
[25] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. Retrieved from http://archive.ics.uci.edu/ml.
[26] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12(Jul 2011), 2121–2159.
[27] Hubert Eichner, Tomer Koren, Brendan McMahan, Nathan Srebro, and Kunal Talwar. 2019. Semi-cyclic stochastic gradient descent. In *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, Vol. 97. 1764–1773.
[28] Facebook. 2017. PyTorch. Retrieved January 20, 2020, from https://pytorch.org/.
[29] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom'18)*. 115–127.
[30] Pedro A. Forero, Alfonso Cano, and Georgios B. Giannakis. 2010. Consensus-based distributed support vector machines. *Journal of Machine Learning Research* 11, May (2010), 1663–1707.
[31] Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. 2018. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866* (2018).
[32] Petko Georgiev, Nicholas D. Lane, Cecilia Mascolo, and David Chu. 2017. Accelerating mobile audio sensing algorithms through on-chip GPU offloading. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'17)*. 306–318.
[33] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan. 2011. SystemML: Declarative machine learning on MapReduce. In *2011 IEEE 27th International Conference on Data Engineering (ICDE'11)*. 231–242.
[34] Andrew Gibiansky. 2017. Bringing HPC Techniques to Deep Learning. Retrieved June 10, 2020, from https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/.
[35] Google. 2017. TensorFlow Lite. Retrieved January 20, 2020, from https://www.tensorflow.org/lite/.
[36] Otkrist Gupta and Ramesh Raskar. 2018. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications* 116 (2018), 1–8.
[37] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604* (2018).
[38] Lie He, An Bian, and Martin Jaggi. 2018. COLA: Communication-efficient decentralized linear learning. *arXiv preprint arXiv:1808.04883* (2018).
[39] Geoffrey E. Hinton. 2007. Learning multiple layers of representation. *Trends in Cognitive Sciences* 11, 10 (2007), 428–434.
[40] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A. Gibson, Greg Ganger, and Eric P. Xing. 2013. More effective distributed ML via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems (NeurIPS'13)*. 1223–1231.

[41] Loc N. Huynh, Youngki Lee, and Rajesh Krishna Balan. 2017. DeepMon: Mobile GPU-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'17)*. 82–95.

[42] Martin Isaksson and Karl Norrman. 2020. Secure federated learning in 5G mobile networks. *arXiv preprint arXiv:2004.06700* (2020).

[43] Martin Jaggi, Virginia Smith, Martin Takác, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I. Jordan. 2014. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems (NeurIPS'14)*. 3068–3076.

[44] Xiaotang Jiang, Huan Wang, Yiliu Chen, Ziqi Wu, Lichuan Wang, Bin Zou, Yafeng Yang, Zongyang Cui, Yu Cai, Tianhang Yu, Chengfei Lv, and Zhihua Wu. 2020. MNN: A universal and efficient inference engine. In *Proceedings of Machine Learning and Systems 2020 (MLSys'20)*, Vol. 2. 1–13.

[45] Thorsten Joachims. 2006. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)*. 217–226.

[46] Rie Johnson and Tong Zhang. 2013. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems (NeurIPS'13)*. 315–323.

[47] Peter Kairouz, H. Brendan McMahan, Brendan Avent, et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[48] Jiawen Kang, Zehui Xiong, Dusit Niyato, Shengli Xie, and Junshan Zhang. 2019. Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory. *IEEE Internet of Things Journal* 6, 6 (2019), 10700–10714.

[49] Jiawen Kang, Zehui Xiong, Dusit Niyato, Yuze Zou, Yang Zhang, and Mohsen Guizani. 2020. Reliable federated learning for mobile networks. *IEEE Wireless Communications* 27, 2 (2020), 72–80.

[50] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[51] Anastasia Koloskova, Tao Lin, Sebastian U. Stich, and Martin Jaggi. 2019. Decentralized deep learning with arbitrary communication compression. *arXiv preprint arXiv:1907.09356* (2019).

[52] Anastasia Koloskova, Sebastian U. Stich, and Martin Jaggi. 2019. Decentralized stochastic optimization and gossip algorithms with compressed communication. *arXiv preprint arXiv:1902.00340* (2019).

[53] Jakub Konecnỳ. 2017. Stochastic, distributed and federated optimization for machine learning. *arXiv preprint arXiv:1707.01155* (2017).

[54] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. 2015. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575* (2015).

[55] Jakub Konečnỳ, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. 2016. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527* (2016).

[56] Jakub Konečnỳ, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[57] Nicholas D. Lane, Petko Georgiev, and Lorena Qendro. 2015. DeepEar: Robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'15)*. 283–294.

[58] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436.

[59] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (1998), 2278–2324.

[60] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. 2004. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research* 5 (Apr 2004), 361–397.

[61] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*. 583–598.

[62] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2019. Federated learning: Challenges, methods, and future directions. *arXiv preprint arXiv:1908.07873* (2019).

[63] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. 2015. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems (NeurIPS'15)*. 2737–2745.

[64] Yang Liu, Tianjian Chen, and Qiang Yang. 2018. Secure federated transfer learning. *arXiv preprint arXiv:1812.03337* (2018).

[65] Dumitrel Loghin, Shaofeng Cai, Gang Chen, Tien Tuan Anh Dinh, Feiyi Fan, Qian Lin, Janice Ng, Beng Chin Ooi, Xutao Sun, Quang-Trung Ta, et al. 2020. The disruptions of 5G on data-driven technologies and applications. *IEEE Transactions on Knowledge and Data Engineering* 32, 6 (2020), 1179–1198.

[66] Chenxin Ma, Virginia Smith, Martin Jaggi, Michael I. Jordan, Peter Richtárik, and Martin Takáč. 2015. Adding vs. averaging in distributed primal-dual optimization. *arXiv preprint arXiv:1502.03508* (2015).
[67] Chenxin Ma and Martin Takáč. 2016. Distributed inexact damped newton method: Data partitioning and load-balancing. *arXiv preprint arXiv:1603.05191* (2016).
[68] Akhil Mathur, Nicholas D. Lane, Sourav Bhattacharya, Aidan Boran, Claudio Forlivesi, and Fahim Kawsar. 2017. DeepEye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'17)*. 68–81.
[69] I. McGraw, R. Prabhavalkar, R. Alvarez, M. G. Arenas, K. Rao, D. Rybach, O. Alsharif, H. Sak, A. Gruenstein, F. Beaufays, and C. Parada. 2016. Personalized speech recognition on mobile devices. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'16)*. 5955–5959.
[70] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. 2016. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629* (2016).
[71] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2017. Learning differentially private language models without losing accuracy. *arXiv preprint arXiv:1710.06963* (2017).
[72] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, D. B. Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. 2016. MLlib: Machine learning in apache spark. *Journal of Machine Learning Research* 17, 34 (2016), 1–7.
[73] Gaurav Mittal, Kaushal B. Yagnik, Mohit Garg, and Narayanan C. Krishnan. 2016. SpotGarbage: Smartphone app to detect garbage using deep learning. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'16)*. 940–945.
[74] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. 2019. Agnostic federated learning. In *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, Vol. 97. 4615–4625.
[75] Solmaz Niknam, Harpreet S. Dhillon, and Jeffery H. Reed. 2019. Federated learning for wireless communications: Motivation, opportunities and challenges. *arXiv preprint arXiv:1908.06847* (2019).
[76] Pitch Patarasuk and Xin Yuan. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal Parallel and Distributed Computing* 69, 2 (2009), 117–124.
[77] Diego Peteiro-Barral and Bertha Guijarro-Berdiñas. 2013. A survey of methods for distributed machine learning. *Progress in Artificial Intelligence* 2, 1 (2013), 1–11.
[78] Maarten G. Poirot, Praneeth Vepakomma, Ken Chang, Jayashree Kalpathy-Cramer, Rajiv Gupta, and Ramesh Raskar. 2019. Split learning for collaborative deep learning in healthcare. *arXiv preprint arXiv:1912.12115* (2019).
[79] Raluca Ada Popa. 2014. *Building Practical Systems that Compute on Encrypted Data*. Ph.D. Dissertation. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science.
[80] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: Protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*. 85–100.
[81] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2012. CryptDB: Processing queries on an encrypted database. *Communications of the ACM* 55, 9 (2012), 103–111.
[82] Foster J. Provost and Daniel N. Hennessy. 1996. Scaling up: Distributed machine learning with cooperation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI'96)*, Vol. 1. 74–79.
[83] Ning Qian. 1999. On the momentum term in gradient descent learning algorithms. *Neural Networks* 12, 1 (1999), 145–151.
[84] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen. 2018. DeepDecision: A mobile deep learning framework for edge video analytics. In *IEEE IEEE Conference on Computer Communications (INFOCOM'18)*. 1421–1429.
[85] Sujith Ravi. 2019. Efficient on-device models using neural projections. In *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, Vol. 97. 5370–5379.
[86] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS'11)*. 693–701.
[87] Jason Rennie. 2007. 20 Newsgroups. Retrieved June 22, 2020, from http://qwone.com/ jason/20Newsgroups/.
[88] Peter Richtárik and Martin Takáč. 2016. Distributed coordinate descent method for learning with big data. *Journal of Machine Learning Research* 17, 1 (2016), 2657–2681.
[89] Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *Annals of Mathematical Statistics* 22, 3 (1951), 400–407.
[90] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.

[91] Sumudu Samarakoon, Mehdi Bennis, Walid Saad, and Merouane Debbah. 2018. Federated learning for ultra-reliable low-latency V2V communications. In *2018 IEEE Global Communications Conference (GLOBECOM'18)*. IEEE, 1–7.

[92] J. Reddi Sashank, Kale Satyen, and Kumar Sanjiv. 2018. On the convergence of Adam and beyond. In *Proceedings of the 6th International Conference on Learning Representations (ICLR'18)*.

[93] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. 2019. Robust and communication-efficient federated learning from non-IID data. *arXiv preprint arXiv:1903.02891* (2019).

[94] Stefano Savazzi, Monica Nicoli, and Vittorio Rampa. 2020. Federated learning with cooperating devices: A consensus approach for massive IoT networks. *IEEE Internet of Things Journal* 7, 5 (2020), 4641–4654.

[95] Alexander Sergeev and Mike Del Balso. 2018. Horovod: Fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[96] Ohad Shamir, Nati Srebro, and Tong Zhang. 2014. Communication-efficient distributed optimization using an approximate newton-type method. In *Proceedings of the 31st International Conference on Machine Learning (ICML'14)*, Vol. 32. 1000–1008.

[97] Vivek Sharma, Praneeth Vepakomma, Tristan Swedish, Ken Chang, Jayashree Kalpathy-Cramer, and Ramesh Raskar. 2019. ExpertMatcher: Automating ML model selection for clients using hidden representations. *arXiv preprint arXiv:1910.03731* (2019).

[98] Muhammad Shayan, Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. 2018. Biscotti: A ledger for private and secure peer-to-peer machine learning. *arXiv preprint arXiv:1811.09904* (2018).

[99] Micah J. Sheller, G. Anthony Reina, Brandon Edwards, Jason Martin, and Spyridon Bakas. 2018. Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation. In *International MICCAI Brainlesion Workshop*. 92–104.

[100] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*. 1310–1321.

[101] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy (SP'17)*. 3–18.

[102] Liu Sicong, Zhou Zimu, Du Junzhao, Shangguan Longfei, Jun Han, and Xin Wang. 2017. UbiEar: Bringing location-independent sound awareness to the hard-of-hearing people with smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (UbiComp'17)* 1, 2 (2017), 1–21.

[103] Abhishek Singh, Praneeth Vepakomma, Otkrist Gupta, and Ramesh Raskar. 2019. Detailed comparison of communication efficiency of split learning and federated learning. *arXiv preprint arXiv:1909.09145* (2019).

[104] Siri Team. 2017. Deep learning for Siri's voice: On-device deep mixture density networks for hybrid unit selection synthesis. Retrieved Nov. 13, 2020, from https://machinelearning.apple.com/research/siri-voices.

[105] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. 2017. Federated multi-task learning. In *Advances in Neural Information Processing Systems 30 (NeurIPS'17)*. Curran Associates, Inc., 4424–4434.

[106] Virginia Smith, Simone Forte, Chenxin Ma, Martin Takáč, Michael I. Jordan, and Martin Jaggi. 2017. CoCoA: A general framework for communication-efficient distributed optimization. *Journal of Machine Learning Research* 18, 1 (2017), 8590–8638.

[107] Alexander Smola and Shravan Narayanamurthy. 2010. An architecture for parallel topic models. *Proceedings of the VLDB Endowment* 3, 1–2 (2010), 703–710.

[108] Boyd Stephen, Parikh Neal, Chu Eric, Peleato Borja, and Eckstein Jonathan. 2010. MPI example for alternating direction method of multipliers. Retrieved June 26, 2020, from https://stanford.edu/ boyd/papers/admm/mpi/.

[109] Shizhao Sun, Wei Chen, Jiang Bian, Xiaoguang Liu, and Tie-Yan Liu. 2017. Ensemble-compression: A new method for parallel training of deep neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD'17)*. 187–202.

[110] Tencent. 2017. ncnn. Retrieved January 20, 2020, from https://github.com/Tencent/ncnn.

[111] T. Tieleman and Geoffrey Hinton. 2012. Neural networks for machine learning. Retrieved from https://www.cs.toronto.edu/tijmen/csc321/slides/lecture_slides_lec6.pdf.

[112] Praneeth Vepakomma, Otkrist Gupta, Abhimanyu Dubey, and Ramesh Raskar. 2019. Reducing leakage in distributed deep learning for sensitive health data. *arXiv preprint arXiv:1812.00564* (2019).

[113] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. 2018. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564* (2018).

[114] Praneeth Vepakomma, Tristan Swedish, Ramesh Raskar, Otkrist Gupta, and Abhimanyu Dubey. 2018. No peek: A survey of private distributed deep learning. *arXiv preprint arXiv:1812.03288* (2018).

[115] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. 2017. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems 30 (NeurIPS'17)*. Curran Associates, Inc., 4148–4158.

[116] Xindong Wu, Vipin Kumar, J. Ross Quinlan, et al. 2008. Top 10 algorithms in data mining. *Knowledge and Information Systems* 14, 1 (2008), 1–37.

[117] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. 2019. A first look at deep learning apps on smartphones. In *The World Wide Web Conference (WWW'19)*. 2125–2136.

[118] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 2 (2019), 12.

[119] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. 2018. Applied federated learning: Improving Google keyboard query suggestions. *arXiv preprint arXiv:1812.02903* (2018).

[120] Hao Yu, Rong Jin, and Sen Yang. 2019. On the linear speedup analysis of communication efficient momentum SGD for distributed non-convex optimization. In *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, Vol. 97. 7184–7193.

[121] Hao Yu, Sen Yang, and Shenghuo Zhu. 2019. Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'19)*, Vol. 33. 5693–5700.

[122] Xiao Zeng, Kai Cao, and Mi Zhang. 2017. MobileDeepPill: A small-footprint mobile deep learning system for recognizing unconstrained pill images. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'17)*. 56–67.

[123] Sixin Zhang, Anna E. Choromanska, and Yann LeCun. 2015. Deep learning with elastic averaging SGD. In *Advances in Neural Information Processing Systems (NeurIPS'15)*. 685–693.

[124] Yuchen Zhang and Xiao Lin. 2015. DiSCO: Distributed optimization for self-concordant empirical loss. In *International Conference on Machine Learning (ICML'15)*, Vol. 37. 362–370.

[125] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated learning with non-IID data. *arXiv preprint arXiv:1806.00582* (2018).

[126] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. 2017. Asynchronous stochastic gradient descent with delay compensation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. 4120–4129.

[127] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J. Smola. 2010. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS'10)*. 2595–2603.

Received January 2020; revised July 2020; accepted September 2020