Federated Block Coordinate Descent Scheme for Learning Global and Personalized Models

Ruiyuan Wu[1], Anna Scaglione[2], Hoi-To Wai[3], Nurullah Karakoc[2], Kari Hreinsson[2], Wing-Kin Ma[1]

[1]Department of Electronic Engineering, The Chinese University of Hong Kong, Hong Kong SAR, China
[2]School of Electrical Computer and Energy Engineering, Arizona State University, USA
[3]Dept. Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong SAR, China
[1]{rywu, wkma}@ee.cuhk.edu.hk, [2]{Anna.Scaglione, nkarakoc, khreinss}@asu.edu, [3]htwai@se.cuhk.edu.hk

Abstract

In federated learning, models are learned from users' data that are held private in their edge devices, by aggregating them in the service provider's "cloud" to obtain a global model. Such global model is of great commercial value in, e.g., improving the customers' experience. In this paper we focus on two possible areas of improvement of the state of the art. First, we take the difference between user habits into account and propose a quadratic penalty-based formulation, for efficient learning of the global model that allows to personalize local models. Second, we address the latency issue associated with the heterogeneous training time on edge devices, by exploiting a hierarchical structure modeling communication not only between the cloud and edge devices, but also within the cloud. Specifically, we devise a tailored block coordinate descent-based computation scheme, accompanied with communication protocols for both the synchronous and asynchronous cloud settings. We characterize the theoretical convergence rate of the algorithm, and provide a variant that performs empirically better. We also prove that the asynchronous protocol, inspired by multi-agent consensus technique, has the potential for large gains in latency compared to a synchronous setting when the edge-device updates are intermittent. Finally, experimental results are provided that corroborate not only the theory, but also show that the system leads to faster convergence for personalized models on the edge devices, compared to the state of the art.

Introduction

Over the past few years, *federated learning*, an emerging branch of distributed learning, has attracted increasing attention (McMahan et al. 2016; Li et al. 2018; Bonawitz et al. 2019; Yang et al. 2019). It focuses on scenarios where users' data are processed for training machine learning models locally, i.e. on users' edge devices such as cell phones and wearable devices, so that the data remain private. Data privacy is, in fact, a top priority; users are willing to share trained models with reliable service providers, but not necessarily the raw data. In this context, the service provider seeks a global model—which can, in turn, enhance the performance for the users—by aggregating these local models. Such global model reflects the "wisdom of the crowd" and helps the service provider to better understand customer preferences. What distinguishes federated learning from conventional distributed learning (where the optimization often happens in stable data centers) are the following aspects (Li et al. 2020):

- the heterogeneity of the data and of the computational power available in the users' different edge devices;
- the intermittent (and costly) nature of the communication between edge devices and the cloud.

To explain it in mathematical terms, the following problem is prototypical in federated learning:

$$\min_{\boldsymbol{x}_i, \boldsymbol{z} \in \mathcal{X}} \ \sum_{i \in \mathcal{Q}} g_i(\boldsymbol{x}_i), \ \text{ s.t. } \boldsymbol{x}_i = \boldsymbol{z}, \ \forall i \in \mathcal{Q}, \tag{1}$$

where $\mathcal{Q}$ is the set of edge devices, $\boldsymbol{x}_i$ and g_i are the model parameter and cost function on the ith edge device that depend on the local data, and $\mathcal{X}$ is the feasible region. Specifically, we can write $g_i(\boldsymbol{x}_i) := \frac{1}{|\mathcal{S}_i|} \sum_{r \in \mathcal{S}_i} h(\boldsymbol{x}_i; \boldsymbol{s}_r)$, where h is the training loss function, $\mathcal{S}_i$ the index set of training data on the ith edge device, $|\mathcal{S}_i|$ the number of elements in the set $\mathcal{S}_i$, and $\boldsymbol{s}_r$ one of such samples. To solve problem (1), federated learning methods typically adopt a recursive mechanism: edge devices process their own training data to update local models $\boldsymbol{x}_i$'s, and a cloud is introduced to aggregate $\boldsymbol{x}_i$'s for a global model $\boldsymbol{z}$ and synchronize all the local models with the updated $\boldsymbol{z}$. This process is considered standard in the current development (McMahan et al. 2016; Li et al. 2018). However, we notice that two facts leave some space for improvement:

(i) *Is it efficient to maintain the same model everywhere?* Data are generally non-independent and identically distributed (i.i.d.) on edge devices, since they reflect the usage habits of different users. In light of this, edge-device models that work well on data of their users' interest (which are also non-i.i.d.) may suffice—in other words, personalized models may be better at the tasks they are primarily used for. When a sole model is maintained, users may sacrifice their customer experience in order to improve the global model that is more beneficial to the service provider to boost new users' models.

(ii) *Is the synchronous cloud model effective?* The cloud consists of a cluster of servers that work in parallel, and an edge device only needs to talk with one such cloud server. When regarding the cloud as a sole computation resource,

one implicitly enforces some form of cloud synchronization, or Sync-cloud (achievable by techniques such as AllReduce (Patarasuk and Yuan 2009)); see Fig. 1(a) for an illustration. In federated learning, this synchronization may lead to significant update latency. To see this, recall that due to the capacity heterogeneity of communication and computation, the times of availability and local training time from edge devices can vary significantly. Consequently, a possible scenario is that most cloud servers are stranded by a few "slow" activated edge devices that even never talk with them.

Contributions

The focus of this paper is on addressing the above two issues. Our contributions and novelty can be summarized as follows:

- Our work proposes a tailored hierarchical communication architecture for federated learning. This structure, composed of master-slave and multi-agent networks, albeit admitting a "surprisingly" familiar look, has not been studied before.
- We propose a lightweight block coordinate descent computation scheme, equipped with judiciously designed communication protocols, which is the first work that can simultaneously achieve the model personalization and cloud server asynchronous updates—two seemly irrelevant issues that are actually closely related to federated learning (c.f. Remark 4).
- The sublinear convergence rate of the proposed algorithm is shown. Since our communication architecture contains two layers of information exchange, the analysis requires different analytical tools from the existing work.
- We provide latency analysis to demonstrate the efficacy of the asynchronous update for federated learning. Our latency analytical framework practically allows to estimate runtime from the distribution of the edge-device message arrival time, and to connect the number of cloud servers involved in each update with the runtime, which is new.
- Finally, numerical experiments on standard machine learning applications are carried out to support our claims.

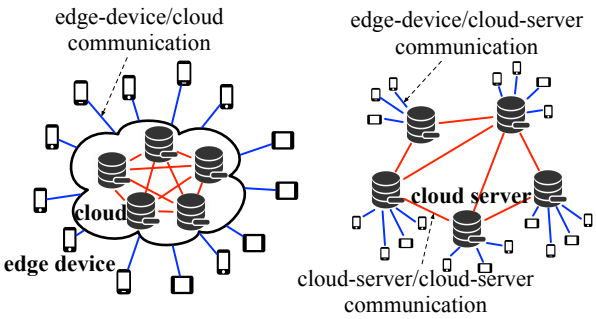
Related Work

FedAvg (McMahan et al. 2016), a simple iterative algorithm, is considered the first work in federated learning. At each round of FedAvg, the cloud sends the global model to part of the edge devices that are activated; then, the activated edge devices update its local model by a fixed epochs of stochastic gradient descent (SGD) on local cost functions; finally, the cloud aggregates the uploaded local models (via a weighted summation) as the new global model. The majority of follow-up work adopts a similar mechanism as that introduced by FedAvg (Li et al. 2020). For example, a more recent variant called FedProx (Li et al. 2018) differs from FedAvg on the edge-device update step, where it imposes an additional proximal term to the cost function, and allows the use of time-varying epochs of faster algorithms, such as accelerated SGD.

Model personalization is a natural, but sometimes ignored, issue under the federated learning scenarios. In the work of (Mohri, Sivek, and Suresh 2019), the authors argue that, due to personal preferences, models trained via FedAvg may be biased towards the interest of the majority. They propose AFL that has a sophisticated mechanism to determine the weights of edge-device models involved in the cloud aggregation, instead of the simple data size ratio used in FedAvg. Per-FedAvg (Fallah, Mokhtari, and Ozdaglar 2020) distinguishes the global model from the ones on edge devices. Their goal is to seek a good "initialization", as the global model, that can be easily upgraded locally to be optimal for each edge device with a few steps of simple updates. (Smith et al. 2017) regards model personalization as a multi-task learning problem. Their proposed MOCHA can learn separate but related models for each device, while leveraging a shared representation via multi-task learning. Attesting the importance of model personalization is the work published during the preparation and submission of this paper (Jiang et al. 2019; Arivazhagan et al. 2019; Wu, He, and Chen 2020; Hu et al. 2020) that focus exclusively on this issue. On the other hand, none of the work considers the penalty-based approach as we do.

In contrast, there is a vast amount of literature on distributed learning in asynchronous multi-agents' networks; see (Nedić, Olshevsky, and Rabbat 2018) for a recent survey. While such implementations require typically more iterations than the master-slave ones, they have no coordination overhead. Recently (Assran et al. 2019) demonstrates experimentally that the reduction on overhead yields significant benefits in terms of runtime. When tasked with a deep-learning problem on a large distributed database the asynchronous multi-agent algorithm runs faster than its master-slave counterpart, because relaxed coordination requirements in turn help complete each update without lags, compared to the traditional master-slave or incremental architectures. As of the submission of this paper, we have not seen the study of this topic in the context of federated learning, as well as no prior theory on how to characterize the performance trends versus the runtime of the algorithm, as opposed to simply focusing on number of iterations required. Though there is work mentioning asynchronous federated learning (Chen et al. 2019; Lu et al. 2019), they concentrate on the asynchronous update caused by the communication between edge devices and cloud—the cloud is still regarded as a sole point.

Problem Statement



(a) Sync-cloud architecture (b) Async-cloud architecture

Figure 1: Two communication architectures.

We detach the cloud servers from each other and consider both server/server and device/server communication. To explain, the cloud servers are connected by a (possibly) dynamic multi-agent graph and they need to talk with each other to achieve consensus, be it exact or asymptotic; the device/server communication is in a master-slave fashion and, as in general federated learning, intermittent and random. We call it the Async-cloud architecture, to distinguish it from the Sync-cloud architecture adopted by FedAvg; see Fig. 1 for their difference. The Async-cloud architecture allows model difference on cloud servers. As we will show later, such flexibility can be exploited to promote more efficient model updates.

Based on above architectures, our proposed formulation is

$$\begin{aligned} \min_{\substack{\{\boldsymbol{x}_i\}_{i\in\mathcal{Q}_n,n\in\mathcal{V}}\\ \{\boldsymbol{z}_n\}_{n\in\mathcal{V}}}} \quad & \sum_{n\in\mathcal{V}}\sum_{i\in\mathcal{Q}_n}\Big(g_i(\boldsymbol{x}_i)+\frac{\gamma_i}{2}\|\boldsymbol{x}_i-\boldsymbol{z}_n\|^2\Big) \\ \text{s.t.} \quad & \boldsymbol{x}_i\in\mathcal{X},\forall i\in\mathcal{Q}_n,n\in\mathcal{V}, \\ & \boldsymbol{z}_n=\boldsymbol{z}_m,\ \forall(n,m)\in\mathcal{E}, \end{aligned} \tag{2}$$

where $\mathcal{V}$ is the set of cloud servers, $\mathcal{Q}_n$ is the set of edge devices connected to the nth cloud server, $\boldsymbol{x}_i$'s are edge-device models, $\boldsymbol{z}_n$'s are cloud-server models, $\mathcal{E}\subseteq(\mathcal{V}\times\mathcal{V})$ indicates communication between cloud servers, and $\gamma_i>0$ is the penalty parameter. In the sequel, $\boldsymbol{z}_n$'s will be referred to as the global model and $\boldsymbol{x}_i$'s the personalized models. Our formulation distinguishes the global models on different cloud servers, and allows (but penalizes by the quadratic penalty regularizer) the deviations among personalized models.

Remark 1 *The seemingly naïve quadratic penalty has recently been revisited in different distributed learning literature. In (Zhang, Choromanska, and LeCun 2015), this penalty is used to seek better solution in deep learning tasks where the underlying optimization has many local optima. This quadratic penalty has also been investigated in adversarial scenarios, where the global model is expected to resist the attack of malicious devices (Yang, Gang, and Bajwa 2020).*

Remark 2 *(Motivation of Global and Personalized Models) Consider smartphone keyboard application. Users want accurate next-word prediction, which tailored personalized models can better deliver. However, each user produces very limited data for training and, thus, its local personalized model may fail to work for new scenarios, which is where the comprehensive knowledge from the global model helps (in our formation, this works by encouraging the personalized model to be close to its global counterpart). Also, the global model can serve as an unbiased initialization for new users.*

Federated Block Coordinate Descent

In this section, we tackle the problem defined in (2). Our development is based on the classic block coordinate descent (BCD) scheme (Bertsekas 1997):

$$\{\boldsymbol{x}_i^{(t+1)}\}_{\substack{i\in\mathcal{Q}_n\\ n\in\mathcal{V}}}=\operatorname*{argmin}_{\boldsymbol{x}_i\in\mathcal{X}} g_i(\boldsymbol{x}_i)+\frac{\gamma_i}{2}\|\boldsymbol{x}_i-\boldsymbol{z}_n^{(t)}\|^2, \tag{3a}$$

$$\{\boldsymbol{z}_n^{(t+1)}\}_{n\in\mathcal{V}}=\operatorname*{argmin}_{\{\boldsymbol{z}_n\}_{n\in\mathcal{V}}}\sum_{n\in\mathcal{V}}\sum_{i\in\mathcal{Q}_n}\frac{\gamma_i}{2}\|\boldsymbol{z}_n-\boldsymbol{x}_i^{(t+1)}\|^2 \tag{3b}$$
$$\text{s.t.}\quad \boldsymbol{z}_n=\boldsymbol{z}_m,\ \forall(n,m)\in\mathcal{E}.$$

Our subsequent effort can be interpreted as adapting the BCD update (3) to the Sync-cloud and Async-cloud architectures. Same as FedAvg, we assume that at each round only part of edge devices are available, and use $\mathcal{Q}_n^{(t)}\subseteq\mathcal{Q}_n$ to denote the set of activated edge devices at round t (i.e., edge devices that are available to do local training and are in stable communication condition). In the sequel, we will elaborate on the cloud-server and edge-device update separately.

Update on the edge device. If edge device i is not activated, i.e., $i\notin\mathcal{Q}_n^{(t)}$, we have $\boldsymbol{x}_i^{(t+1)}=\boldsymbol{x}_i^{(t)}$. Otherwise, time-varying epochs of accelerated stochastic projected gradient (ASPG) is applied. Specifically, letting t^- be the last round when the ith edge device was activated and given the initialization $\boldsymbol{x}_i^{(t,0)}=\boldsymbol{x}_i^{(t^-,K_i^{(t^-)})}$ and $\boldsymbol{x}_i^{(t,-1)}=\boldsymbol{x}_i^{(t^-,K_i^{(t^-)}-1)}$, the edge device recursively performs:

$$\boldsymbol{x}_{i,\mathsf{ex}}^{(t,k)}=\boldsymbol{x}_i^{(t,k)}+\zeta(\boldsymbol{x}_i^{(t,k)}-\boldsymbol{x}_i^{(t,k-1)}), \tag{4a}$$

$$\boldsymbol{x}_i^{(t,k+1)}= \tag{4b}$$
$$\Pi_{\mathcal{X}}\left(\boldsymbol{x}_{i,\mathsf{ex}}^{(t,k)}-\eta_x\left(\nabla\tilde{g}_i(\boldsymbol{x}_{i,\mathsf{ex}}^{(t,k)})+\gamma_i(\boldsymbol{x}_{i,\mathsf{ex}}^{(t,k)}-\boldsymbol{z}_n^{(t)})\right)\right),$$

for $k=0,\dots,K_i^{(t)}-1$, where η_x is the stepsize, $\zeta\geq0$ is the momentum weight, $\Pi_{\mathcal{X}}$ is the projection onto $\mathcal{X}$, and $\nabla\tilde{g}_i$ is the mini-batch stochastic gradient such that $\nabla\tilde{g}_i=\frac{1}{R}\sum_{r=1}^R\nabla h(\boldsymbol{x}_i;\boldsymbol{\xi}_r^{(t,k)})$ with $\boldsymbol{\xi}_r^{(t,k)}$ being a sample from the ith edge device and R being the batch size. The final update is $\boldsymbol{x}_i^{(t+1)}=\boldsymbol{x}_i^{(t,K_i^{(t)})}$. If $\zeta=0$, update (4) narrows down to the standard SPG. Empirically, the ASPG update is observed to converge faster (Beck and Teboulle 2009).

Update on the cloud server. We derive the update rule for (3b) for both Sync-cloud and Async-cloud architectures. In the synchronous case, same as FedAvg, we can denote the global models on all participating cloud servers as $\boldsymbol{z}_n=\boldsymbol{z}$ for $n\in\mathcal{V}$. Then, given initialization $\boldsymbol{z}^{(t)}$, the cloud updates

(Sync-cloud)
$$\boldsymbol{z}^{(t+1)}=\boldsymbol{z}^{(t)}-\eta_z\sum_{n\in\mathcal{V}}\sum_{i\in\mathcal{Q}_n}\gamma_i(\boldsymbol{z}^{(t)}-\boldsymbol{x}_i^{(t+1)}), \tag{5}$$

where η_z is the stepsize.

In the asynchronous setting, we propose the use of the distributed gradient descent (DGD) algorithm (Ram, Nedić, and Veeravalli 2010). Similarly, given initialization $\boldsymbol{z}_n^{(t)}$, for $n\in\mathcal{V}$, each cloud server runs

(Async-cloud)
$$\boldsymbol{w}_n^{(t)}=\sum_{m\in\mathcal{V}}a_{n,m}^{(t)}\boldsymbol{z}_m^{(t)}, \tag{6a}$$

$$\boldsymbol{z}_n^{(t+1)}=\boldsymbol{w}_n^{(t)}-\eta_z\sum_{i\in\mathcal{Q}_n}\gamma_i(\boldsymbol{w}_n^{(t)}-\boldsymbol{x}_i^{(t+1)}), \tag{6b}$$

where $a_{n,m}^{(t)}$ is a mixing coefficient decided by the server/server communication link at round t.

Algorithm 1 Pseudocode of FedBCD for problem (2)

1: **for** $t = 0, 1, \ldots, T-1$ **do**
2: **for** $n \in \mathcal{V}$ **do**
3: cloud server sends $\boldsymbol{z}_n^{(t)}$ to activated edge device $i \in \mathcal{Q}_n^{(t)}$.
4: activated edge device updates $\boldsymbol{x}_i^{(t)}$ by $K_i^{(t)}$ epochs of ASPG (4) on $g_i(\boldsymbol{x}_i) + \frac{\gamma_i}{2}\|\boldsymbol{x}_i - \boldsymbol{z}_n^{(t)}\|^2$ and obtain $\boldsymbol{x}_i^{(t+1)}$; other edge device has $\boldsymbol{x}_i^{(t+1)} = \boldsymbol{x}_i^{(t)}$.
5: cloud server receives the uploaded $\boldsymbol{x}_i^{(t+1)}$'s from activated edge devices.
6: cloud servers update $\boldsymbol{z}_n^{(t+1)}$'s by the Sync-cloud/Async-cloud update (5) or (6)

Combining (4)-(6), we obtain our FedBCD scheme. Due to the page limit, we only give the pseudocode of FedBCD in Algorithm 1, and the precise version is given in the supplementary document. In Section , we will characterize the theoretical convergence guarantee of FedBCD.

Sync-Cloud and Async-Cloud Communication Protocols for FedBCD

We introduce two communication protocols to clarify how FedBCD can be applied, and how the Async-cloud protocol can allow potentially more efficient updates.

- *Sync-cloud protocol.* This protocol is similar to that in (Bonawitz et al. 2019) and has two phases: (i) A new round t starts with the "response and update" phase. In this phase, cloud servers accept update requests from edge devices, send the global model $\boldsymbol{z}^{(t)}$ to these activated edge devices, and wait for them to perform the local update (4). For each cloud server, this phase ends when it receives $\boldsymbol{x}_i^{(t+1)}$'s from a pre-determined number of edge devices. Then, this cloud server replaces its local copy of $\boldsymbol{x}_i^{(t)}$'s with $\boldsymbol{x}_i^{(t+1)}$'s (for inactivated edge devices, it simply puts $\boldsymbol{x}_i^{(t+1)} = \boldsymbol{x}_i^{(t)}$) and sends $\sum_{i\in\mathcal{Q}_n} \boldsymbol{x}_i^{(t+1)}$ to a coordinator that keeps the latest global model $\boldsymbol{z}^{(t)}$. (ii) Once hearing from all cloud servers, this coordinator enters the "aggregation" phase, where it executes (5) to obtain $\boldsymbol{z}^{(t+1)}$ and broadcasts it back. After receiving $\boldsymbol{z}^{(t+1)}$, the cloud servers become available again for the next round of update.

- *Async-cloud protocol.* This protocol requires more careful implementation. (i) At round t, cloud servers that finished the "response and update" phase send their model $\boldsymbol{z}_n^{(t)}$'s to the coordinator. (ii) The coordinator waits until it hears from the first B cloud servers $\mathcal{V}^{(t)}$ [in Fig. 2(b) it is $|\mathcal{V}^{(t)}| = B = 2$], and enters "aggregation" phase I. Then, this coordinator calculates $\boldsymbol{w}^{(t)} = \sum_{n\in\mathcal{V}^{(t)}} \boldsymbol{z}_n^{(t)}/B$ and sends $\boldsymbol{w}^{(t)}$ to cloud servers in $\mathcal{V}^{(t)}$. (iii) The cloud servers in $\mathcal{V}^{(t)}$ enter "aggregation" phase II, where they use received $\boldsymbol{w}^{(t)}$ as $\boldsymbol{w}_n^{(t)}$ and update $\boldsymbol{z}_n^{(t+1)}$ by (6); the other cloud servers simply have $\boldsymbol{z}_n^{(t+1)} = \boldsymbol{z}_n^{(t)}$. Back to our FedBCD scheme, this protocol corresponds to the following setting: $\boldsymbol{a}_{n,m}^{(t)} = 1/B$ for $n, m \in \mathcal{V}^{(t)}$, $\boldsymbol{a}_{n,n}^{(t)} = 1$ and $\boldsymbol{a}_{n,m}^{(t)} = 0$ for $n, m \notin \mathcal{V}^{(t)}$; $\mathcal{Q}_n^{(t)} = \emptyset$ and $\eta_{z_n}^{(t)} = 0$ for $n \notin \mathcal{V}^{(t)}$.

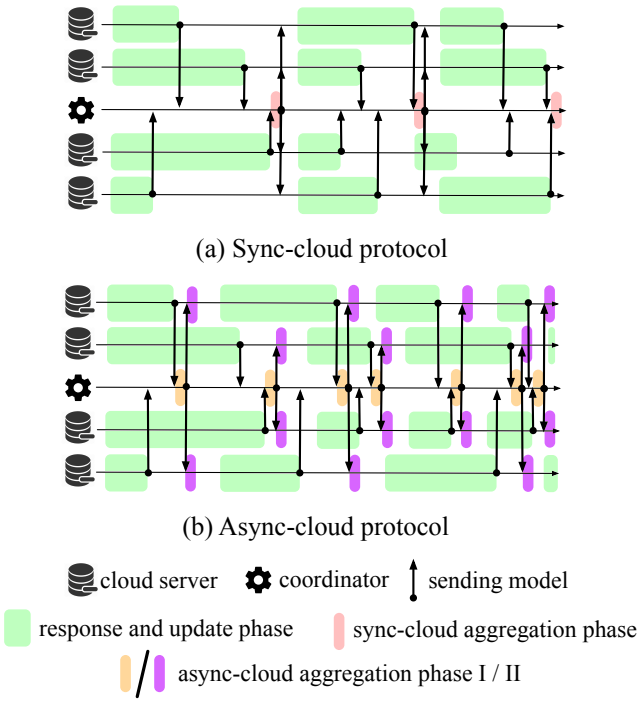


Figure 2: Protocols under Sync-/Async-cloud architectures.

In the Sync-cloud protocol, the model aggregation happens when all cloud servers finish the "response and update" phase; while in the second protocol, cloud servers work asynchronously, and the model aggregation is carried out in a "first come first serve" manner. By constructing the neighborhoods of cloud servers that mix their model dynamically based on those who are ready first, the Async-cloud protocol, as analyzed below, is expected to provide more efficient update; see Fig. 2 for an illustration.

Remark 3 *Some readers may argue that the use of the coordinator in both protocols is not favorable for multi-agent communication. However, unlike in the conventional setting, herein the coordinator only involves lightweight computations. We believe that with the advent of software-defined networking, it is fairly natural to assume that specialized servers communications, such as multi-agent message exchanges, can be orchestrated in an effective way to support specific applications. Methods that completely eliminate coordination require re-normalizing the weights (Assran et al. 2019); typically the algorithms take longer to converge.*

Latency analysis. Here we provide a methodology to gain analytical insight, by assuming a certain distribution for the inter-update time. Let $\tau_n^{(t)}, n \in \mathcal{V}$ be random time taken by cloud server n to accrue $\mathcal{Q}_n^{(t)}$ edge-devices updates; assume $\tau_n^{(t)}$ are i.i.d. with mean and standard deviation μ_τ, σ_τ respectively. Each $\tau_n^{(t)}$ equals the sum of update request arrival time and the computation time associated to $\mathcal{Q}_n^{(t)}$ edge-devices. With $|\mathcal{V}| = N$, denote by $\tau_{(k)}^{(t)}$, $k = 1, \ldots, N$ the ordered statistics of the samples $\tau_n^{(t)}$ in increasing order, i.e. $\tau_{(1)}^{(t)} \le \tau_{(2)}^{(t)} \le \ldots \le \tau_{(N)}^{(t)} = \max_{n\in\mathcal{V}} \tau_n^{(t)}$. Let the latency of

the Sync-cloud and Async-cloud protocols at round t be $\tau_{\rm SC}^{(t)}$ and $\tau_{\rm AC}^{(t)}$. It follows that $\tau_{\rm SC}^{(t)} = \tau_{(N)}^{(t)}$ and $\tau_{\rm AC}^{(t)} = \tau_{(B)}^{(t)}$. Hence the reduction in the average duration per round is:

$$r_{B,N} := \mathbb{E}[\tau_{(B)}^{(t)}]/\mathbb{E}[\tau_{(N)}^{(t)}] \equiv \mathbb{E}[\tau_{\rm AC}^{(t)}]/\mathbb{E}[\tau_{\rm SC}^{(t)}]. \quad (7)$$

Let $\beta = B/N$ denote the percentage of network nodes involved in each update, and consider the fact that $r_{B,N} \leq r_{B,N-1}$. For large N we can leverage a classic result (Mosteller 2006), showing the asymptotic normality of ordered statistics for large sample size. In particular by denoting $F_{\tau_n}^{-1}(u)$ the quantile function of the random variable τ_n, we have

$$\tau_{(\beta N)}^{(t)} \sim \mathcal{N}\left(F_{\tau_n}^{-1}(\beta), \frac{\beta(1-\beta)}{N\left[f_{\tau_n}\left(F_{\tau_n}^{-1}(\beta)\right)\right]^2}\right) \quad \Rightarrow$$

$$r_{\beta N,N} \lessapprox \frac{F_{\tau_n}^{-1}(\beta)}{F_{\tau_n}^{-1}(1-1/N)}, \text{ for } N \gg 1. \quad (8)$$

For $\tau_n^{(t)}$ with sample space $[0, +\infty)$, $F_{\tau_n}^{-1}(1-1/N) \to +\infty$, there is an infinite gain in latency asymptotically. For typical distributions, though, the rate is slow. For example, for $\tau_n^{(t)}$ following a Weibull distribution, with shape parameter k (irrespective of the scale parameter λ), we have $r_{\beta N,N} \lessapprox -(\ln(1-\beta))^{\frac{1}{k}}(\log(N))^{-\frac{1}{k}}$.

For a random time distribution with finite support $[0, \overline{\tau}]$ with $\overline{\tau} < +\infty$, instead, the reduction saturates to $F_{\tau_n}^{-1}(\beta)/\overline{\tau}$. Also notice that, in general, the smaller is the percentage of network nodes in each update β the slower is the update progress per round, yielding a trade-off between reducing latency and increasing the update progress, which results in an optimum choice of β. Finally, it should be pointed out that we only provide a straightforward solution to illustrate the potential of the Async-cloud architecture. Multi-agent consensus techniques such as DGD have been well studied in distributed learning (Assran et al. 2019), opening the door to further advances to promote more efficient updates.

Remark 4 *Model personalization and asynchronous updates may seem to be two independent topics at first glance. However, we emphasize that both of them inherently match the heterogeneous nature of federated learning—they go hand in hand, due to the idiosyncratic behavior of the edge devices manifesting itself in both the non i.i.d. data and the non-uniform times at which updates are carried out. In view of this, only forcing equal models but applying asynchronous updates (or the reverse) is actually a half-measure.*

An Intuitive Variant of FedBCD

In FedBCD, edge devices update local models only when they are in stable communication condition, which could lead to slow local model update. On the other hand, it is reasonable to assume that edge devices can also train local models offline. For instance, think about a phone that is charged and idle. This is a good time for the phone to do local training, even if it is not connected to Wi-Fi. Thus, we have $\mathcal{Q}_n^{(t)} \subseteq \tilde{\mathcal{Q}}_n^{(t)} \subseteq \mathcal{Q}_n$, where $\tilde{\mathcal{Q}}_n^{(t)}$ is the set of edge devices that are available for local training but in bad communication condition. We can then take the intuitive approach suggested in (Zhang, Choromanska, and LeCun 2015) by tackling the cost function and quadratic penalty in (2) separately. Simply speaking, at round t, edge devices in $\tilde{\mathcal{Q}}_n^{(t)}$ run local training using cost functions g_i's. Then, the subset of edge devices in $\mathcal{Q}_n^{(t)}$ sends update requests to the cloud for the global model to adjust their local models. The cloud-server updates remain unchanged. We call the resulting scheme FedBCD-I, and summarize its pseudocode in Algorithm 2. FedBCD-I can adopt both the Sync-cloud and Async-cloud protocols introduced for FedBCD with minor modification. The only difference is that, herein edge devices send update requests only when they finish their round of local training. If an edge device fails to communicate with the cloud for a pre-determined number of rounds, it will stop local training and wait for the earliest opportunity to contact the cloud. A detailed algorithm and protocol description is provided in the supplementary document.

Algorithm 2 Pseudocode of FedBCD-I for problem (2)

for $t = 0, 1, \ldots, T-1$ **do**
  for $i \in \tilde{\mathcal{Q}}_n^{(t)}, n \in \mathcal{V}$ **do**
    edge device updates $\boldsymbol{x}_i^{(t)}$ by $K_i^{(t)}$ epochs of ASPG on $g_i(\boldsymbol{x}_i)$ and obtain $\boldsymbol{x}_i^{(t+1)}$.
  for $n \in \mathcal{V}$ **do**
    cloud server sends $\boldsymbol{z}_n^{(t)}$ to activated edge device $i \in \mathcal{Q}_n^{(t)} \subseteq \tilde{\mathcal{Q}}_n^{(t)}$.
    activated edge device updates $\boldsymbol{x}_i^{(t+1)}$ by $K_i^{(t)}$ epochs of $\boldsymbol{x}_i^{(t+1)} \leftarrow \Pi_{\mathcal{X}}(\boldsymbol{x}_i^{(t+1)} - \gamma_i(\boldsymbol{x}_i^{(t+1)} - \boldsymbol{z}_n^{(t)}))$.
    cloud server receives the uploaded $\boldsymbol{x}_i^{(t+1)}$'s from activated edge devices.
  cloud servers update their models $\boldsymbol{z}_n^{(t+1)}$'s by Sync-cloud/Async-cloud update with information only from $\mathcal{Q}_n^{(t)}$; i.e., replacing $\mathcal{Q}_n$ in (5) or (6) with $\mathcal{Q}_n^{(t)}$.

Theoretical Convergence Analysis

We investigate the behavior of FedBCD in this section. Our result applies to both the Sync-cloud and Async-cloud settings. Let us start with the following assumptions:

Assumption 1 *The server/server and device/server communication satisfies that*
(i) Each edge device is activated at least once every p iterations, where $p < \infty$.
(ii) In the Async-cloud update (6)*, the communication graph within cloud servers is possibly time-varying and satisfies, for some $q < \infty$ that, $(\mathcal{V}, \bigcup_{t=1,\ldots,q} \mathcal{E}^{(t_0+t)})$ is strongly connected for all t_0.*

Assumption 2 *In Problem* (2) *the conditions below are met:*
(i) The feasible set $\mathcal{X}$ is convex and compact.
(ii) For all $i \in \mathcal{Q}_n, n \in \mathcal{V}$, function g_i is bounded on $\mathcal{X}$, and is L_i-Lipschitz smooth on $\tilde{\mathcal{X}}$, where $\tilde{\mathcal{X}} = \text{conv}[\bigcup_{0\leq\zeta\leq 1}\{\boldsymbol{x} + \zeta(\boldsymbol{x}-\boldsymbol{y}) \mid |\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}\}]$ is the feasible set extended by the momentum update. Also, its gradient ∇g_i is bounded on $\mathcal{X}$.

Assumption 3 *The mixing coefficient $\{a_{n,m}^{(t)}\}_{t=0,1,\ldots,}$ in Async-cloud update* (6) *satisfies the conditions below:*

(i) There exists a scalar $c > 0$ such that $a_{n,m}^{(t)} \geq c$ if $(n,m) \in \mathcal{E}^{(t)}$ and $a_{n,m}^{(t)} = 0$ otherwise.
(ii) (doubly stochastic) $\sum_{m\in\mathcal{V}} a_{n,m}^{(t)} = \sum_{n\in\mathcal{V}} a_{n,m}^{(t)} = 1$.

Assumption 4 *The stepsizes and momentum weight satisfy*
(i) $\eta_x \leq \min_{i\in\mathcal{Q}_n, n\in\mathcal{V}}\{1/(L_i+\gamma_i)\}$.
(ii) $\eta_z = d/\sqrt{T} \leq 1/(\max_{i\in\mathcal{Q}_n,n\in\mathcal{V}}\{\gamma_i\}\cdot\max_{n\in\mathcal{V}}\{|\mathcal{Q}_n|\})$, for a positive constant d and total communication round T.
(iii)$\zeta \leq \min_{i\in\mathcal{Q}_n,n\in\mathcal{V}}\{\omega/\sqrt{1+\rho_i\eta_x}\}$, for some constants $\omega \in (0,1)$ and ρ_i such that $\rho_i \leq L_i + \gamma_i$.

Assumption 5 *For any i, let $\nabla h(\boldsymbol{x}_i;\boldsymbol{\xi}_i)$ be a stochastic gradient of $g_i(\boldsymbol{x}_i)$, where $\boldsymbol{\xi}_i$ is a random sample from the ith edge device [recall that in (1) we denote $g_i(\boldsymbol{x}_i) := \frac{1}{|\mathcal{S}_i|}\sum_{r\in\mathcal{S}_i} h(\boldsymbol{x}_i;\boldsymbol{s}_r)$]. It holds for any $\boldsymbol{x}_i$ and $\boldsymbol{\xi}_i$ that $\mathbb{E}[\nabla g_i(\boldsymbol{x}_i) - \nabla h(\boldsymbol{x}_i;\boldsymbol{\xi}_i)] = \mathbf{0}$ and $\mathbb{E}[\|\nabla g_i(\boldsymbol{x}_i) - \nabla h(\boldsymbol{x}_i;\boldsymbol{\xi}_i)\|^2] \leq \sigma^2$.*

Let us briefly examine our assumptions: Assumption 1 makes sure that no edge device or cloud server is isolated; Assumptions 2-3 are common in constrained optimization and average consensus algorithms; Assumption 4 requires that the stepsize and the momentum weight are bounded; Assumption 5 is typical for stochastic gradient descent. Careful readers may notice that the Async-cloud protocol in Section violates Assumption 4 by considering time-varying η_z. This issue can be fixed by minor modification; the resulting version is, however, more complicated in terms of implementation and is detailed in the supplementary document.

We characterize the convergence of FedBCD to a stationary point. Our convergence metric is the constrained version of the one in (Assran et al. 2019).

Theorem 1 *Suppose that Assumptions 1-5 hold, communication round T is sufficiently large (see the supplementary document for a clear definition), and batch size R is used to calculate the stochastic gradients (c.f.the detailed discussion of the stochastic gradient below (4))*
(i) Sequences $\{\boldsymbol{x}_i^{(t)}\}$, $\{\boldsymbol{z}_n^{(t)}\}$ generated by FedBCD satisfy

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\langle\nabla_{\boldsymbol{x}_i} f_i(\boldsymbol{x}_i^{(t+1)},\bar{\boldsymbol{z}}^{(t)}),\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i^{(t+1)}\right\rangle\right] \geq -\frac{A_1}{T^{1/4}} - \frac{A_2}{R^{1/4}},\ \forall\hat{\boldsymbol{x}}\in\mathcal{X},\ i\in\mathcal{Q}_n, n\in\mathcal{V},$$

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\sum_{n\in\mathcal{V}}\sum_{i\in\mathcal{Q}_n}\gamma_i(\boldsymbol{z}^{(t)} - \boldsymbol{x}_i^{(t+1)})\right\|^2\right] \leq \frac{B_1}{\sqrt{T}} + \frac{B_2}{\sqrt{R}},$$

$$\frac{1}{T}\sum_{t=0}^{T-1}\|\bar{\boldsymbol{z}}^{(t)} - \boldsymbol{z}_n^{(t)}\| \leq \frac{C}{\sqrt{T}},\ \forall n\in\mathcal{V},$$

where A_1, A_2, B_1, B_2, C are positive constants (determined by parameters in Assumption 1-5) whose accurate forms are in the supplementary document, and $\bar{\boldsymbol{z}}^{(t)} = \frac{1}{|\mathcal{V}|}\sum_{n\in\mathcal{V}}\boldsymbol{z}_n^{(t)}$.
(ii) If we use the exact gradients during updates, the above results hold with all terms related to the batch size R removed.

The proof of Theorem 1 is in the supplementary document. We see that the batch size R and communication round T determine the convergence rate, which is the case in projection-based stochastic algorithms (Ghadimi, Lan, and Zhang 2016). Note that on edge devices the data size is not large and thus exact gradients are possibly computable. For this case, our result reveals a sub-linear convergence rate of $\mathcal{O}(1/\sqrt{T})$, which is consistent with the existing decentralized non-convex algorithms (Li et al. 2018; Assran et al. 2019).

Remark 5 *In our analysis, we discuss the convergence of both the Sync-cloud and Async-cloud settings in a unified way. If we only consider the Sync-cloud setting, FedBCD will become a special case of a centralized computation scheme (Wu, Wai, and Ma 2020). By modifying the assumptions to make FedBCD fit this computation scheme, we will be able to obtain a $\mathcal{O}(1/T)$ sub-linear convergence rate, which is standard for centralized non-convex algorithms. On the other hand, it should be pointed out that the Async-cloud setting of FedBCD is not seen in the literature, and requires careful and non-trivial analysis.*

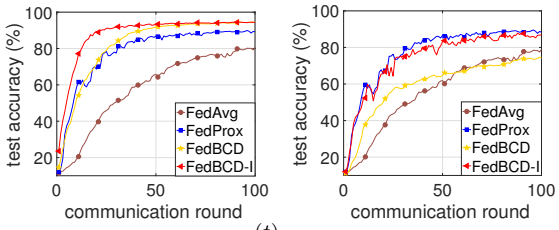
Numerical Experiments

We train two models on PyTorch: a three-layer neural network for the classification of the MNIST dataset (LeCun et al. 1998) and a deeper ResNet-20 model (He et al. 2016) for the CIFAR-10 dataset. We assume that there are 10 cloud servers, each connected to 10 edge devices. To emulate the data heterogeneity, we restrict the "diversity" of training data; e.g., "diversity" being 3 means that each edge device owns data of three labels. We also use a small $|\mathcal{Q}_n^{(t)}|$ to emulate occurrences of intermittent communication. The other settings are: the edge-device learning rate is $\eta_x = 0.005$, the momentum weight ζ is 0.9, the edge-device update epoch is sampled from $[1,5]$ every round, $\mathcal{X}$ is an elementwise $[-2,2]$-box constraint, the batch size is $R = 32$; for FedBCD/FedBCD-I, the cloud-server learning rate is $\eta_z = 0.5$; for FedBCD, the penalty parameter is $\gamma_i = 1$; for FedBCD-I, we use $\gamma_i = 0.2$, $|\tilde{\mathcal{Q}}_n^{(t)}| = 8$, and edge devices suspend local training after finishing 4 offline rounds; for FedProx, ASPG is used in edge-device update, and the penalty parameter is $\mu = 5$. The above parameters selection was by trial and error.

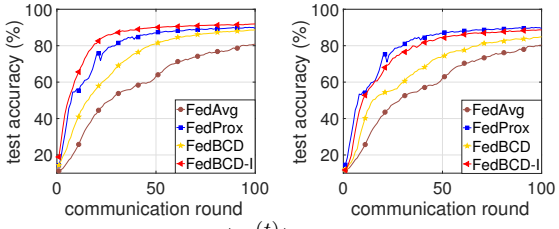
Model Deviation vs. Model Consensus

We compare FedBCD/FedBCD-I (which allow model deviations) with FedAvg/FedProx (which enforce model consensus). Two performance measures are considered: the personalized performance, where edge devices test local models on test data that have the same diversity as their training data; and the global performance, where the cloud tests the global model on the whole test data set. To be fair, in this experiment all algorithms use the Sync-cloud protocol (see the supplementary document for the Sync-cloud protocol for FedAvg/FedProx), and the two types of performance are recorded every round (i.e. the latency per round is not considered). It can be seen in Fig. 3 that, the personalized and global performance of FedAvg/FedProx is almost the same. This makes sense, since therein the same model is maintained everywhere. In comparison, FedBCD/FedBCD-I, by allowing local models to deviate from the global one, provide better personalized performance. We also see that FedBCD

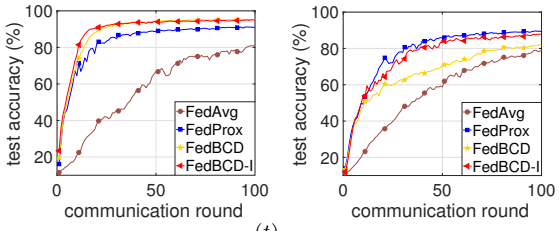
• (left column) average **personalized**, (right column) **global**



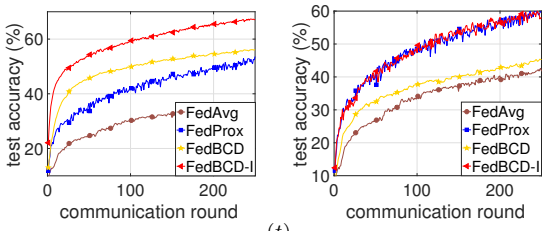
(a) MNIST-$|\mathcal{Q}_n^{(t)}| = 3$, diversity: 3



(b) MNIST-$|\mathcal{Q}_n^{(t)}| = 3$, diversity: 6



(c) MNIST-$|\mathcal{Q}_n^{(t)}| = 6$, diversity: 3



(d) CIFAR10-$|\mathcal{Q}_n^{(t)}| = 5$, diversity: 3

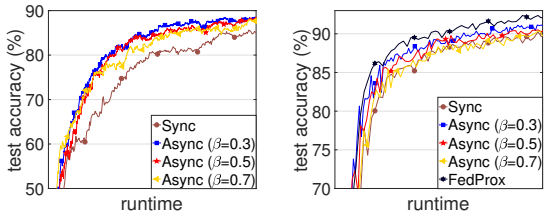
Figure 3: Test accuracy of models by different algorithms with different diversity and activation.

suffers from slow update progress when the $|\mathcal{Q}_n^{(t)}|$ is low and is only faster than FedAvg (this is not surprising, see Section). In contrast, its intuitive variant FedBCD-I turns out to be more competitive; particularly, we see that in the more challenging CIFAR-10 task, FedBCD-I has global performance comparable to FedProx, while giving much better personalized performance.

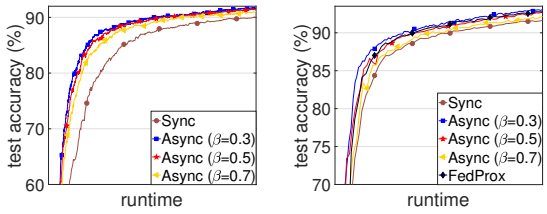
Sync-Cloud vs. Async-Cloud

Following the latency analysis in Section , we demonstrate the possible efficiency gain of the Async-cloud update. To emulate the distributed scenario of federate learning, we simulate each edge device/cloud server and the coordinator in Python using TCP sockets for inter-process communication, and test it on four 60 core servers with Intel Xeon E7-4870 v2 CPUs and 256 GB of memory each. In the experiment, we

• **global** performance:



(a) FedBCD (b) FedBCD-I



(c) FedBCD (d) FedBCD-I

Figure 4: Efficiency of the algorithms, on MNIST with $|\mathcal{Q}_n^{(t)}| = 3$ and "diversity "(a-b) 3, (c-d) 6

.

generate numerically the random latency $\tau_n^{(t)}$ for the "respond and update" phase (green phases in Figure 2) and assume that server aggregation computation (non-green phases in Figure 2) is negligible, i.e. the overall runtime is dominated by $\tau_n^{(t)}$. The detailed settings, and the statistics used in our simulations are given in the supplementary document. The results are shown in Fig. 4. We first notice that, the higher "diversity" leads to a smoother improvement of the global performance. This is reasonable, since a more "biased" local model is expected to bring more "noise" to the global model training. Besides, we can see that the Async-cloud update is clearly more efficient. As mentioned before, based on the size of β, there is a trade-off between the duration and the update progress of each round. In this experiment, we see that $\beta = 0.3$ or 0.5 seems to strike a good balance and leads to appealing performance. We also compare FedBCD-I with FedProx. Note that FedProx focuses on improving the global performance, and is generally faster than our methods for the global performance. However, by using the asynchronous update, our scheme could outperform FedProx even for the global performance in terms of runtime; see Fig. 4(d).

Conclusion

This work has two main contributions: first, we recognize that machine learning models on edge devices reflect the user habits, and can thus be different; second, we spotlight that the cloud is a cluster of powerful servers, and their intra-communication can be exploited for more efficient updates. We study these two aspects and provide a solution that is proven promising theoretically and empirically.

Acknowledgments

This work was supported by the Army Research Office, USA, under Project ID ARO #W911NF-20-1-0153. The work by Ruiyuan Wu and Wing-Kin Ma was supported by a General Research Fund (GRF) of the Research Grant Council (RGC), Hong Kong, under Project ID CUHK 14208819.

References

Arivazhagan, M. G.; Aggarwal, V.; Singh, A. K.; and Choudhary, S. 2019. Federated learning with personalization layers. *arXiv preprint arXiv:1912.00818*.

Assran, M.; Loizou, N.; Ballas, N.; and Rabbat, M. 2019. Stochastic gradient push for distributed deep learning. In *Proc. Int. Conf. Mach. Learn.*, volume 97, 344–353. PMLR.

Beck, A.; and Teboulle, M. 2009. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sci.* 2(1): 183–202.

Bertsekas, D. P. 1997. Nonlinear programming. *J. Oper. Res. Soc.* 48(3): 334–334.

Bonawitz, K.; Eichner, H.; Grieskamp, W.; Huba, D.; Ingerman, A.; Ivanov, V.; Kiddon, C.; Konecny, J.; Mazzocchi, S.; McMahan, H. B.; et al. 2019. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*.

Chen, Y.; Ning, Y.; Slawski, M.; and Rangwala, H. 2019. Asynchronous online federated learning for edge devices with non-IID data. *arXiv preprint arXiv:1911.02134*.

Fallah, A.; Mokhtari, A.; and Ozdaglar, A. 2020. Personalized federated learning: A meta-learning approach. *arXiv preprint arXiv:2002.07948*.

Ghadimi, S.; Lan, G.; and Zhang, H. 2016. Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization. *Math. Program.* 155(1-2): 267–305.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 770–778.

Hu, R.; Guo, Y.; Li, H.; Pei, Q.; and Gong, Y. 2020. Personalized federated learning with differential privacy. *IEEE Internet Things J.* 7(10): 9530–9539.

Jiang, Y.; Konečnỳ, J.; Rush, K.; and Kannan, S. 2019. Improving federated learning personalization via model agnostic meta learning. *arXiv preprint arXiv:1909.12488*.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceed. IEEE* 86(11): 2278–2324.

Li, T.; Sahu, A. K.; Talwalkar, A.; and Smith, V. 2020. Federated learning: Challenges, methods, and future directions. *IEEE Signal Process. Mag.* 37(3): 50–60.

Li, T.; Sahu, A. K.; Zaheer, M.; Sanjabi, M.; Talwalkar, A.; and Smith, V. 2018. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*.

Lu, Y.; Huang, X.; Dai, Y.; Maharjan, S.; and Zhang, Y. 2019. Differentially private asynchronous federated learning for mobile edge computing in urban informatics. *IEEE Trans. Industr. Inform.* 16(3): 2134–2143.

McMahan, H. B.; Moore, E.; Ramage, D.; Hampson, S.; et al. 2016. Communication-efficient learning of deep networks from decentralized data. In *Proc. Int. Conf. Artif. Intell. Stat. (AISTATS)*.

Mohri, M.; Sivek, G.; and Suresh, A. T. 2019. Agnostic federated learning. *arXiv preprint arXiv:1902.00146*.

Mosteller, F. 2006. On some useful inefficient statistics. In *Selected Papers of Frederick Mosteller*, 69–100. Springer.

Nedić, A.; Olshevsky, A.; and Rabbat, M. G. 2018. Network topology and communication-computation tradeoffs in decentralized optimization. *Proc. IEEE* 106(5): 953–976.

Patarasuk, P.; and Yuan, X. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel Distrib. Comput.* 69(2): 117–124.

Ram, S. S.; Nedić, A.; and Veeravalli, V. V. 2010. Distributed stochastic subgradient projection algorithms for convex optimization. *J. Optim. Theory Appl.* 147(3): 516–545.

Smith, V.; Chiang, C.-K.; Sanjabi, M.; and Talwalkar, A. S. 2017. Federated multi-task learning. *Adv. Neural. Inf. Process. Syst.* 30: 4424–4434.

Wu, Q.; He, K.; and Chen, X. 2020. Personalized federated learning for intelligent IoT applications: A cloud-edge based framework. *IEEE Comput. Graph Appl.* .

Wu, R.; Wai, H.-T.; and Ma, W.-K. 2020. Hybrid inexact BCD for coupled structured matrix factorization in hyperspectral super-resolution. *IEEE Trans. Signal Process.* 68: 1728–1743.

Yang, Q.; Liu, Y.; Chen, T.; and Tong, Y. 2019. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.* 10(2): 1–19.

Yang, Z.; Gang, A.; and Bajwa, W. U. 2020. Adversary-resilient distributed and decentralized statistical inference and machine learning: An overview of recent advances under the Byzantine threat model. *IEEE Signal Process. Mag.* 37(3): 146–159.

Zhang, S.; Choromanska, A. E.; and LeCun, Y. 2015. Deep learning with elastic averaging SGD. In *Adv. Neural Inf. Process. Syst.*, 685–693.