Local Model Privacy-Preserving Study for Federated Learning

Kaiyun Pan[1], Daojing He[1(✉)], and Chuan Xu[2]

[1] Software Engineering Institute, East China Normal University, Shanghai, China
51184501038@stu.ecnu.edu.cn, djhe@sei.ecnu.edu.cn
[2] Inria Sophia Antipolis, Valbonne, France
chuan.xu@inria.fr

Abstract. In federated learning framework, data are kept locally by clients, which provides naturally a certain level of privacy. However, we show in this paper that a curious onlooker can still infer some sensitive information of clients by looking at the exchanged messages. More precisely, for the linear regression task, the onlooker can decode the exact local model of each client in a constant number of rounds under both cross-device and cross-silo federated learning settings. We improve one of the learning algorithms and experimentally show that it makes the onlooker harder to decode the local model of clients.

Keywords: Federated learning · Privacy-preserving · Distributed optimization · Differential privacy

1 Introduction

Nowadays, data privacy draws public attention in the approach of machine learning and statistics. Under the distributed network, many mobile devices can generate amounts of rich data and store them locally every day. With the improvement of computing and storage capabilities on these devices, it introduces the concern on the transmission of private sensitive data. Therefore, training statistical global models on remote devices rather than overpowered data server and making the storage of data locally become urgent problems to be solved. This is where the concept of federated learning (FL) comes from [1]. For privacy concerns, clients may not be willing to share their original data to data centers and keeping all private data on a central data processing server to train statistical models.

In conventional federated learning optimization algorithms, there is an assumption with a synchronous updating scheme running on entities communicating round by round in a distributed network. There is one server and fixed selected clients with fixed local dataset in this network, which the server sends the current global model parameters such as the model weight to selected clients for efficiency. In convex setting, each selected client calculates the updating

J. Garcia-Alfaro et al. (Eds.): SecureComm 2021, LNICST 398, pp. 287–307, 2021.
https://doi.org/10.1007/978-3-030-90019-9_15

gradient-descent step using the receiving model parameters from the server and sends the update to the server. The server averages on all receiving updates from selected clients and gets the updating global model parameter, and the process repeats. All selected clients and the server agree with a common optimal global model parameters.

The objective of the training is to solve the problem below:

$$\min_{w \in \mathbb{R}^d} l(w) \quad \text{where} \quad l(w) = \frac{1}{n} \sum_{i=1}^{n} l_i(w), \tag{1}$$

where n is the number of clients, w is the model parameters and $l_i(w)$ is the loss function of client i with respect to its local dataset. For a machine learning problem, the objective can be non-convex neural network and convex functions.

One primary advantage of federated learning is that the raw data are locally stored on the clients which can avoid the hidden onlooker to eavesdrop the personal raw data on the communication channels between clients and server. With the increasing attention of data privacy-preserving of personal information, data security and privacy-preserving analysis become important hotspots in myriad domains. Whereas FL has a prominent advantage on protecting personal information for clients, the sensitive information still could be deduced somehow by onlookers with analyzing the differences of the trained and uploaded relevant parameters sent by the clients. For example, paper [2] showed FL can disclose some important personal data from the parameter updates for distributed optimization and the transmission of gradients, the work in [3] demonstrated the client-level privacy leakage from federated learning by the attack from a malicious server.

In the literature, the traditional method to prevent privacy leakage is differential privacy (DP) [4]. Methods of DP based on FL were taken into account the trade-off between the convergence performance and privacy during the training process. The work in [6] proposed DP based FL algorithm focusing on the clients' privacy-preserving which utilized secure multiparty computation (SMC) for avoiding differential attacks. Whereas, the above works did not take into consideration the risk about local model privacy of clients from hidden onlookers during the uploading process. Motivated by this issue under the FL framework, we give an attempt to improve one FL algorithm and focus on how to protect the local model privacy for clients if there exists the onlooker based on our assumptions. There has been work done to show that sensitive information was hidden inside the model. The work in [7] introduced a model inversion attack for a linear classifier study, in which sensitive information of clients might be learned by the adversarial access to an ML model. Paper [8] proposed a novel class of model inversion attack, which showed how the adversarial queries recovered facial images only given the related names and access to ML model.

In our work, we consider two federated learning settings in paper [9]. Under the cross-device FL setting, a great deal of clients who are mobile or IOT devices can collaboratively train a model, the scale of clients is up to 10^{10}, which is massively parallel. The primary problems on the setting might be the communication efficiency. It's been applied in many domains, for example, Apple

applied cross-device FL in IOS 13, applications like the voice recognition for "Hey Siri" [10], Google applied it extensively to features on Pixel phones [11] and the Gboard mobile keyboard [12]. Under cross-silo FL setting, clients train a model on siloed data who are different organizations, such as financial or medical data centers, while the scale of clients is normally 2–100 clients. The primary problem in the setting is about the communication or the computation. Cross-silo FL has been applied in rich domains like smart manufacturing [13] and reinsurance financial risk prediction [14]. For the data distribution under these settings, data are generated and stored locally on clients without the right to read data from others.

We give the counter-examples to show in both settings to show the known FL algorithms we studied can not protect the local model privacy when solving a simple linear regression task. Surprisingly, the onlooker can decode the exact local model in $O(1)$ time by just looking at exchanged messages. We improve one of the learning algorithms and experimentally show that it makes the onlooker harder to decode for the local model of clients.

The reminder of this paper is organized as follows: In Sect. 2, we introduce the preliminaries and background of FL. In Sect. 3, we introduce two kinds of FL settings, give the decoding process for the onlooker and propose our private method. In Sect. 4, we design a new onlooker to decode the model privacy and demonstrate our numerical experiments. In Sect. 5, we give our concluding remarks.

2 Preliminaries and Background

This section introduces some preliminaries and the related work about our study.

2.1 Graph Theory

Consider a network of N nodes represented by a directed graph $G = (V, E)$. Node set is $V = \{1, 2, ..., n\}$. Edge set is $E \subset V \times V$, whose elements are $(i, j) \in E$ if and only if there is a communication link from node j to i, i.e., node j can send messages to node i. We assume no self-edges, i.e., $(i, i) \notin E$ for all $i \in V$. Parameter $p_{i,j} > 0$ represents the weight associated with each edge (i, j). The out-neighbor set of node i, i.e., the set of nodes that can receive messages from node i, denoted as $N_i^{out} = \{j \in V | (j, i) \in E\}, j \in N_i^{out}$. Similarly, the in-neighbor of node i which the set of nodes can send messages to node i, denoted as $N_i^{in} = \{j \in V | (i, j) \in E\}, j \in N_i^{in}$. Node i's out-degree is denoted as $D_i^{out} = |N_i^{out}|$ and its in-degree is denoted as $D_i^{in} = |N_i^{in}|$. Our work focuses on strongly connected graphs [15–17] which are defined as follows.

Definition 1. *A directed graph is strongly connected if for any $i, j \in V$, there is at least one directed path from i to j in G [18].*

2.2 Differential Privacy (DP)

The differential privacy was first proposed by Dwork et al. [19] in 2006 and has been widely studied, (ϵ, δ)–differential privacy was first addressed in [20].

Definition 2. *Differential privacy. A randomized algorithm $\mathcal{M}$ with domain $\mathbb{N}^{|x|}$ is (ϵ, δ)–differential private if for all $\mathcal{S} \subseteq \mathcal{M}$ and for all $x, y \in \mathbb{N}^{|x|}$ such that $\|x - y\|_1 \leq 1$:*

$$\Pr[\mathcal{M}(x) \in \mathcal{S}] \leq \exp(\epsilon) \Pr[\mathcal{M}(y) \in \mathcal{S}] + \delta,$$

where the probability space is over the coin flips of the mechanism $\mathcal{M}$, x is the dataset which we will query on,$\|x - y\|_1 \leq 1$ is a measure of how many records differ between x and y. If $\delta = 0$, we say that $\mathcal{M}$ is ϵ–differential private (DP).

2.3 Related Work

With the improvement of computing and storage capabilities of mobile devices, and the fact that the rich data which trained on data center is often private–sensitive, McMahan et al. [1] introduces the concept of *Federated learning*, in which keeping the trained local data on the mobile devices and learning a shared global machine learning model by collecting the updates locally calculated.

Achieving average consensus is an important problem in distributed computing and has been widely studied in distributed networks. Kempe et al. [21] and Bénézit et al. [22] introduced the conventional push–sum algorithm to achieve average consensus for nodes interacting on a directed graph. In the convex setting, Nedic et al. [23] proposed consensus–based gradient descent (CBGD) algorithm for distributed optimization. The work in [15] took efforts to study the converging rate to consensus. In this research direction, the relationship between consistency and convergence was worthy of attention [24]. Olshevsky et al. [25] proposed push–sum gradient descent (PSGD) algorithms while clients only sent partial model parameters to its neighbors for privacy which was the first setting on directed time–varying graphs. Paper [26,27,34] attached great importance to distributed optimization and estimation in machine learning. Paper [28,33,35] proposed a distributed subgradient optimization algorithm which was very close to the work done in [25].

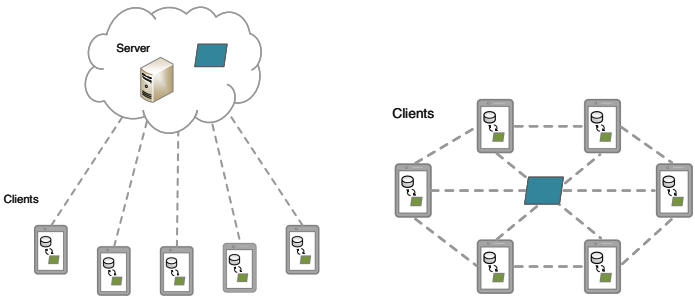
During the federated learning process, even sensitive data held on the mobile devices, risks of privacy leakage are still available among the transmitted channels. One approach for privacy–preserving for sensitive data is adding noises on them. Differential privacy provides privacy guarantee, which ensures that the output from a query on the dataset does not change obviously whether one single data point for the inputting dataset is absent or not [4]. The work in [5] first gave an attempt to propose a novel DP algorithm which not only satisfied the privacy requirement but also kept the provable learning guarantees in convex settings. Nabi et al.[29] proposed an optimization algorithm applied a differential privacy mechanism into FL. Wei et al. [30] presented a novel approach by applying differential privacy before aggregating step, i.e., noising before

model aggregation FL (NbAFL). DP could give the guarantees of privacy while sacrificing the accuracy, the trade–off between privacy and accuracy should be considered. In our work, we will show that our proposed algorithm could protect more privacy which means making the onlooker harder to decode the local model without sacrificing accuracy of results.

3 Model Privacy

In this section, we introduce the conception of local model privacy. We take into account the topology of a network as a distributed network with n clients. The communication channels are connected among n clients in a strongly connected directed graph $G = (V, E)$. Client i can send(receive) messages to (from) client $j, j \in N_i^{out}(N_i^{in})$. In this directed graph, every client i has a weight parameter vector $w_i \in \mathbb{R}^d$ related to the (global) model. Let $l_i(w) : \mathbb{R}^b \to \mathbb{R}$ be the local(loss) function of client i and w_i^* be the optimal local model, i.e., $w_i^* = \arg\min l_i(w)$. During the federated training process, clients can communicate with neighbors and update their w_i approximate to the optimal global parameter $w^* = \arg\min \sum_{i=1}^{N} l_i(w)$.

In our work, we assume that there is a strong onlooker who can eavesdrop all transmissions on channels during the training process, and know the training algorithm $\mathcal{A}$ and even the structure of the local (loss) function. Let $\mathcal{M}_i(\mathcal{A}, t)$ be the set of all the observations the onlooker collected by listening to the channels of the client i till round t when the algorithm is running on the network. Based on $\mathcal{M}_i(\mathcal{A}, t)$, the onlooker runs its decoding algorithm $\mathcal{D}$ to infer the client i's local model w_i^*. Let $w_i^{\mathcal{D}}(t)$ be the decoded model obtained by the onlooker, i.e., $w_i^{\mathcal{D}}(t) = \mathcal{D}(\mathcal{M}_i(\mathcal{A}, t))$. When the strong onlooker has the decoded model $w_i^{\mathcal{D}(t)}$ equaling to w_i^*, we say that the algorithm can not protect the local model privacy. In the following, we show that in both cross–device and cross–silo FL settings, the model privacy could not be protected when clients learn a linear regression model.



(a) Cross–device setting. (b) Cross–silo setting.

Fig. 1. FL networks

3.1 Cross–Device Federated Learning

In order to achieve more efficient communication during the process of training global model, paper [1] proposed FederatedAveraging (FedAvg) algorithm shown in Algorithm 1. FedAvg is one of the cross–device FL algorithms controlled by three key parameters: C, the fraction of clients that perform calculation on each round; B, the local minibatch size used for clients at updating step; and E, the number of training passes (epochs) on local data of clients on each round. FedAvg calculates the updating gradient–descent step with E times instead only one time on selected clients at each round before averaging step. The server chooses C–fraction of clients on each round to do the SGD step over all data maintained on these clients [1]. At each round, the selected clients execute local SGD updates for E epochs before sending the updated model parameters back to the server, then the server averages all the model parameters sent by clients and sends back the updating parameters to clients, the process repeats at each round.

Note that the structure of cross–device FL distributed network is shown in Fig. 1(a) in which it has a server and clients, the communication among them is not peer–to–peer.

Algorithm 1. FederatedAveraging (FedAvg) algorithm.

//The K clients are indexed by k and η is the learning rate.
Server executes:
initialize w_0
for each round $t = 1, 2, \ldots$ **do**
$m \leftarrow \max(C \cdot K, 1)$
$S_t \leftarrow$ (random set of m clients)
for each client $k \in S_t$ **in parallel do**
$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
for batch $b \in \mathcal{B}$ **do**
$w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

Observation 1. *When clients train a linear regression task using full local batch for gradient updates, FedAvg can not protect the local model privacy.*

Proof. Here, we give a simple instance to demonstrate Observation 1. We assume there is an onlooker who knows the structure of the loss function, the learning rate η, and how FedAvg works.

We consider a simple one–dimensional linear regression model ($d = 1$) and the loss function is least quadratic. Thus the loss function is shown as following:

$$l_i(w) = a_i(w - w_i^*)^2 + b_i \tag{2}$$

and

$$\nabla l_i(w) = 2a_i w - 2a_i w_i^*. \tag{3}$$

So at round t, the onlooker who listens on the channels between client i and server can collect the messages w_t and w_t^i, w_t represents the model parameter on server and w_t^i represents the updated model on client i.

We set the notation $e \in \{1, ..., E\}$ and $w_t^i(e)$ represents the model that updated of the eth local epoch on client i at round t, note that $w_t^i = w_t^i(E)$. At the phase of ClientUpdate in Algorithm 1, we get the following updating equation on client i at round t:

$$\begin{aligned} w_t^i(e) &= w_t^i(e-1) - \eta \nabla l_i(w_t^i(e-1)) \\ &= (1 - 2a_i\eta)^e w_t + 2\eta a_i w_i^* \sum_{j=0}^{e-1} (1 - 2a_i\eta)^j \\ &= x_i^e w_t + w_i^*(1 - x_i^e), \end{aligned} \tag{4}$$

where $x_i = 1 - 2a_i\eta$ and only x_i and w_i^* are unknown.

The decoding algorithm $\mathcal{D}$ works as follows.

Once the onlooker gets the messages $w_t, w_t^i(E)$ at round t, he rebuilds the following linear equation in two unknowns:

$$w_t^i(E) = x_i^E w_t + w_i^*(1 - x_i^E), \tag{5}$$

where $w_t^i(E), w_t$, E are known.

At round $t+1$, the onlooker rebuilds another linear equation in two unknowns if $i \in S_{t+1}$:

$$w_{t+1}^i(E) = x_i^E w_{t+1} + w_i^*(1 - x_i^E). \tag{6}$$

By solving the system of linear equations in two unknowns (5) and (6), at round $t+1$, the onlooker gets:

$$x_i^E = \frac{d_{t+1}^i(E)}{d_{t+1}} \tag{7}$$

and

$$w_i^{\mathcal{D}} = w_i^* = \frac{w_t^i(E) - x_i^E w_t}{1 - x_i^E}, \tag{8}$$

where $d_{t+1}^i(E) = w_{t+1}^i(E) - w_t^i(E), d_{t+1} = w_{t+1} - w_t$.

The above analysis shows that the onlooker can succeed in decoding the sensitive local model of client i, i.e., w_i^*. Therefore, when clients train a linear regression task, FedAvg which is under cross–device FL setting can not protect the local model privacy.

3.2 Cross–Silo Federated Learning

Consensus–Based Gradient Descent Algorithm (CBGD). We first study the widely used CBGD algorithm[23] that was first proposed for decentralized

optimization where no centralized node exists that can be easily applied in a cross–silo setting. This algorithm can avoid the communication bottleneck introduced in FedAvg, and can be highly adaptive to the communication network among silos. One feature of CBGD algorithm is that clients can interact and send (receive) their updating models to (from) their neighbors in an undirected graph. The algorithm can be realized synchronous distributed optimization based on consensus (but requiring convex and separable functions) [17,31].

The CBGD algorithms follow the following general procedures at round t, for each client i on a strongly connected graph shown in Fig. 1(b) seen as undirected.

Algorithm 2. Consensus–based gradient descent algorithm by client $i \in V$ at round t.

1. **for** $t \in \{0, ..., T\}$ **do**
2. broadcasts model $w_i(t)$ to its neighbors in N_i and receives models $w_j(t)$ from every $j \in N_i$
3. calculates a weighted average over all the received models and its own model $w_i(t)$: $z_i(t) = \sum_{j \in N_i \cup \{i\}} p_{i,j} w_j(t)$,
   notes that the weight matrix P is doubly stochastic
4. updates its model: $w_i(t+1) = z_i(t) - \eta_t \nabla l_i(\tilde{w}(t))$,
   where $\tilde{w}(t)$ could be $w_i(t)$ or $z_i(t)$ depending on the algorithm

Observation 2. *When clients train a linear regression task, consensus–based gradient descent algorithm can not protect the local model privacy.*

Proof. We consider the same linear regression model as the former subsection and the knowledge of the local loss function is the form as (2) and (3). Therefore, the onlooker who listens for the channels of client i collects the data $\mathcal{M}_i(\mathcal{A}, t) = \mathcal{M}_i(\mathcal{A}, t-1) \cup \{w_j | j \in N_i \cup \{i\}\}$. Since the onlooker knows how the algorithm works including the consensus matrix P and learning rate η, he can obtain $z_i(t)$ and $\tilde{w}(t)$, then obtain $w_i(t+1)$.

The process of decoding algorithm $\mathcal{D}$ works as follows.

Once the onlooker gets the knowledge of $w_i(t+1)$ at round $t+1$, he can rebuild the following linear equation:

$$w_i(t+1) - z_i(t) = -2\eta_t(a_i\tilde{w}(t) - a_i w_i^*), \tag{9}$$

where only a_i and w_i^* are unknown. At round $t+2$, doing the same as before, he can get another linear equation:

$$w_i(t+2) - z_i(t+1) = -2\eta_{t+1}(a_i\tilde{w}(t+1) - a_i w_i^*). \tag{10}$$

Thus, the onlooker can get two unknown linear equations from (9) and (10), then recalculate the solutions as:

$$a_i = \left(\frac{d_i(t+2)}{2\eta_{t+1}} - \frac{d_i(t+1)}{2\eta_t}\right) \times \frac{1}{\tilde{w}(t) - \tilde{w}(t+1)}$$

and

$$w_i^{\mathcal{D}}(t+2) = w_i^* = \tilde{w}(t) + \frac{d_i(t+1)}{2a_i\eta_t},$$

where $d_i(t+1) = w_i(t+1) - z_i(t)$.

Therefore, starting from round 0, the onlooker under assumptions could succeed in decoding the local optimum w_i^* of client i at round 2, i.e., when clients train a linear regression task, the cross–silo FL algorithm can not protect local model privacy.

Push–Sum Gradient Descent Algorithm (PSGD). In CBGD algorithm, we observe that the entire model parameters sent by the clients to neighbors gave an important hint for the decoding process of the onlooker. In order to avoid the privacy leakage, we turn to study PSGD algorithm [25] in which clients only send partial models to their neighbors in directed graphs (also works in undirected graphs). PSGD algorithm aims to solve the minimize problem shown in (1) in a time–varying series of uniformly strongly connected directed graphs with a collection of n nodes, which have access to the local dataset to calculate the corresponding loss functions (convex functions). Each node knows its out–degree and the state of the algorithm by rounds, while unknown about the number of clients and the graph series as the condition to implement the algorithm. Paper [25] shows that this algorithm can achieve distributed optimization with the convergence rate as $O(\log t/\sqrt{t})$. Nevertheless, we find the privacy vulnerability risk during the communication among all nodes in the distributed networks.

We reference the knowledge of graph theory shown in Sect. 2.1, $p_{j,i}(t)$ represents the weight put on the client i's model when i sends its model parameters to its neighbor $j, j \in N_i^{out}(t)$ at round t, and $\sum_{j\in N_i^{out}(t)\cup\{i\}} p_{j,i}(t) = 1$, i.e., P is column–stochastic.

For initialization, client i has $x_i(0) = w_i(0) \in \mathbb{R}^d$, scalar variable $y_i(0) = 1$, weights $p_{j,i}(t), \forall j \in N_i^{out}(t) \cup \{i\}, t \in \{0, ..., T\}$. The algorithm works as follows at round t, for each client i.

Algorithm 3. Push–sum gradient descent (PSGD) algorithm by client $i \in V$ at round t.

1. **for** $t \in \{0, ..., T\}$ **do**
2. computes $p_{j,i}(t)x_i(t)$ and $p_{j,i}(t)y_i(t)$, sends them to all client $j \in N_i^{out}(t)$
3. receives $p_{i,j}(t)x_j(t)$ and $p_{i,j}(t)y_j(t)$ from every client $j \in N_i^{in}(t)$ and sums them as follows:

$$z_i(t+1) = \sum_{j\in\ N_i^{in}(t)\cup\{i\}} p_{i,j}(t)x_j(t), \quad y_i(t+1) = \sum_{j\in\ N_i^{in}(t)\cup\{i\}} p_{i,j}(t)y_j(t)$$

4. updates local parameter: $w_i(t+1) \leftarrow \frac{z_i(t+1)}{y_i(t+1)}$
5. executes one step of gradient descent: $x_i(t+1) \leftarrow z_i(t+1) - \eta_{t+1}\nabla l_i(w_i(t+1))$

Observation 3. *When clients train a linear regression task, push–sum gradient descent algorithm can not protect the local model privacy.*

Proof. With the assumptions that the onlooker knows how Algorithm 3 works, the learning rate $\eta(t)$, the structure of the loss function of clients and $y_i(0) = 1$, we give the decoding procedures for the onlooker to decode $x_i(t), z_i(t), w_i(t+1)$ during the execution of Algorithm 3.

We consider a simple one–dimension linear regression task ($d = 1$), and the knowledge of the local loss function is the form as (2) and (3).

At round 0, the onlooker who eavesdrops the channels of i collects messages such as $\mathcal{M}_x^{in}(0) = \{p_{i,j}(0)x_j(0), \forall j \in N_i^{in}(0)\}$, $\mathcal{M}_y^{in}(0) = \{p_{i,j}(0)y_j(0), \forall j \in N_i^{in}(0)\}$, $\mathcal{M}_x^{out}(0) = \{p_{j,i}(0)x_i(0), \forall j \in N_i^{out}(0)\}$, $\mathcal{M}_y^{out}(0) = \{p_{j,i}(0)y_i(0), \forall j \in N_i^{out}(0)\}$. Because he knows $y_i(0) = 1$, he can randomly pick one pair of corresponding data from $\mathcal{M}_x^{out}(0), \mathcal{M}_y^{out}(0)$ for calculating $x_i(0) = \frac{p_{j,i}(0)x_i(0)}{p_{j,i}(0)y_i(0)}y_i(0)$. Due to weight matrix P is column–stochastic, i.e., $\sum_{j \in N_i^{out}(t) \cup \{i\}} p_{j,i}(t) = 1$, he can obtain the values of $z_i(1), y_i(1)$:

$$\begin{aligned} z_i(1) &= \sum_{j \in N_i^{in}(0)} p_{i,j}(0)x_j(0) + p_{i,i}(0)x_i(0) \\ &= \sum_{j \in N_i^{in}(0)} p_{i,j}(0)x_j(0) + (1 - \sum_{j \in N_i^{out}(0)} p_{j,i}(0))x_i(0) \\ &= \sum_{\mathbf{m} \in \mathcal{M}_x^{in}(0)} \mathbf{m} - \sum_{\mathbf{m} \in \mathcal{M}_x^{out}(0)} \mathbf{m} + x_i(0) \end{aligned} \tag{11}$$

and

$$\begin{aligned} y_i(1) &= \sum_{j \in N_i^{in}(0) \cup \{i\}} p_{i,j}(0)y_j(0) \\ &= \sum_{\mathbf{m} \in \mathcal{M}_y^{in}(0)} \mathbf{m} - \sum_{\mathbf{m} \in \mathcal{M}_y^{out}(0)} \mathbf{m} + y_i(0). \end{aligned} \tag{12}$$

Therefore, the onlooker can decode the value of $w_i(1)$ from updating step in Algorithm 3:

$$w_i(1) = \frac{z_i(1)}{y_i(1)}.$$

Then the onlooker can rebuild the following linear equation according to the gradient–descent step in line 5 from Algorithm 3 and the structure of loss function:

$$x_i(1) = z_i(1) - 2\eta_1 a_i(w_i(1) - w_i^*), \tag{13}$$

where $x_i(1)$ can be obtained by the onlooker at round 1 by doing the same process for decoding the value of $x_i(0)$ at round 0.

As the process repeats, the onlooker can obtain $z_i(2), y_i(2), w_i(2)$ by listening the channels of client i to rebuild the following linear equation:

$$x_i(2) = z_i(2) - 2\eta_2 a_i(w_i(2) - w_i^*), \tag{14}$$

where the onlooker can obtain $x_i(2)$ by repeating the same procedures as before at round 2.

Thus the onlooker can get the two unknown linear equations from (13) and (14) to recalculate the solutions as:

$$a_i = (\frac{d_i(2)}{2\eta_2} - \frac{d_i(1)}{2\eta_1}) \times \frac{1}{w_i(1) - w_i(2)})$$

and

$$w_i^{\mathcal{D}} = w_i^* = \frac{d_i(1)}{2a_i\eta_1 + w_i(1)},$$

where $d_i(t) = x_i(t) - z_i(t)$.

Thus, the onlooker can obtain the local models of clients only by 3 rounds from round 0, i.e., the PSGD algorithm can not protect local model privacy.

3.3 Private Push–Sum Gradient Descent Algorithm (PPSGD)

Algorithm 4. Private push–sum gradient descent by client $i \in V$ at round t.

// Client i has initial value $x_i(0) = w_i(0) \in \mathbb{R}^d$ and scalar value $0 < \beta < \alpha < \frac{1}{2}$.

1. randomly generates $y_i(0)$ from a distribution on a non–zero positive range
2. **for** $t \in \{0, ..., T\}$ **do**
3. **if** $|N_i^{out}(t)| > 0$ **then**
4. chooses $p_{i,i}$ from a distribution on range $[\beta, \alpha]$
5. $p_{j,i} \leftarrow \frac{1-p_{i,i}(t)}{|N_i^{out}(t)|}, \forall j \in N_i^{out}(t)$
6. **else** $p_{i,i} \leftarrow 1$
7. broadcasts $p_{j,i}(t)x_i(t), p_{j,i}(t)y_i(t)$ to clients $j \in N_i^{out}(t)$ and receives $p_{i,j}(t)x_j(t)$, $p_{i,j}(t)y_j(t)$ from every client $j \in N_i^{in}(t)$ and sums them as follows:

$$z_i(t+1) = \sum_{j \in N_i^{in}(t) \cup \{i\}} p_{i,j}(t)x_j(t), \quad y_i(t+1) = \sum_{j \in N_i^{in}(t) \cup \{i\}} p_{i,j}(t)y_j(t)$$

8. updates local parameter: $w_i(t+1) \leftarrow \frac{z_i(t+1)}{y_i(t+1)}$
9. executes gradient descent step: $x_i(t+1) \leftarrow z_i(t+1) - \eta_{t+1}\nabla l_i(w_i(t+1))$

According to the analysis about the privacy leakage from the three observations, we focus on improving Algorithm 3 to make the onlooker harder to decode the local model privacy. Since the onlooker can deduce the local models by eavesdropping all the transmission without adding noises on channels under the three FL algorithms, we find the condition of $y_i(0) = 1$ in Algorithm 3 can give the onlooker very important information to decode local model privacy. In order to puzzle the onlooker, we propose our private method shown in Algorithm 4 by randomly generating $y_i(0)$ at the client side in line 1 and designing a new strategy for weight matrix P in line 3–6. The work in [25] showed the weight

strategy in time–varying graphs that the weights put on the messages of client i were the same as $p_{j,i}(t) = \frac{1}{|N_i^{out}(t)|+1}, \forall j \in N_i^{out}(t) \cup \{i\}$. However, the design for the weight matrix P in [25] can be deduced easily for the onlooker who can listen the messages on the channels with the knowledge of the degree $|N_i^{out}(t)|$ of each client at each round, so that to help the onlooker to obtain the values of $x_i(t), y_i(t), z_i(t+1), w_i(t+1)$ at round t, then even decoding the local model. The weight strategy in our proposed private method can hardly make the onlooker to decode the value of $p_{j,i}(t)$ even he knows the degree $|N_i^{out}(t)|$ of client i, i.e., making the onlooker harder to decode the local privacy. We experimentally show the performance of PPSGD in the next section.

4 Experiments and Relevant Analysis

In this section, we implement on our distributed network shown in Fig. 2 to show the convergence of our PPSGD, the performance of the onlooker designed in Sect. 4.2 on decoding from the time–series data by listening the channels of client i, the classification accuracy for the onlooker from different distributions of $y_i(0)$ on our PPSGD and we also run our algorithm on a real dataset to see the corresponding results.

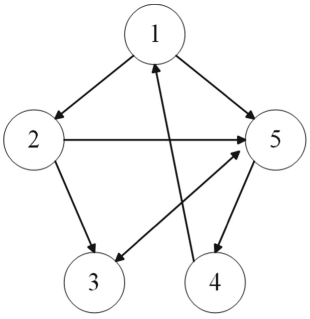


Fig. 2. A strongly connected directed graph with 5 nodes.

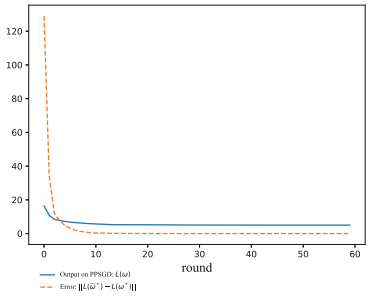


Fig. 3. Convergence on PPSGD.

We consider a network of 5 nodes whose goal is to distributively solve the following minimization problem which is the same idea with (1):

$$\min \quad L(w) \triangleq \sum_{i=1}^{n} l_i(w) \quad \text{over} \quad w \in \mathbb{R}^d, \tag{15}$$

where we assume node i has already trained its local model on the local dataset which is a convex function $l_i : \mathbb{R}^d \to \mathbb{R}^1$, i.e., the loss function. Under the assumption that the set of optimal solutions $w^* = \arg\min_{w \in \mathbb{R}^d} L(w)$ is not empty. We

apply the PPSGD by which all clients maintain variables $w_i(t)$ converging to the same point in w^* over time.

Thus, our baseline is the original local models (loss functions) trained by the clients are shown in following:

$$\begin{cases} l_1(w) = w^2 - 4w + 4 \\ l_2(w) = w^2 - 9w + 9 \\ l_3(w) = w^2 - 4w + 7 \\ l_4(w) = w^2 - 4w + 4.75 \\ l_5(w) = w^2 + w + 0.25 \end{cases} \tag{16}$$

and

$$\min \quad L(w) \triangleq \sum_{i=1}^{5} l_i(w) = 5(w-2)^2 + 5. \tag{17}$$

Hence, we can see the optimum of the global model $L(w)$ is 5 when $w = 2$ which means each client converges to the optimal solution $w^* = 2$.

4.1 The Convergence of PPSGD Algorithm

We use the programming language Python to implement our PPSGD algorithm to see the convergence results based on the baseline. Figure 3 shows the evolution of $L(w)$ and the error between the real global optimum $L(w^*)$ and the output $L(\tilde{w}^*)$ from our PPSGD. We observe that the output converges to the exact optimal value 5 and the error is equal to 0 after around 20 rounds. Thus, our proposed algorithm can still guarantee the convergence without sacrificing the accuracy of the optimal solution and achieve distributed optimization. Actually the convergence of Algorithm 4 can be proven, but here we move the result to the future work.

4.2 New Design for Onlooker

Now we consider an onlooker with an easier task: is the local model different from our baseline when the onlooker only has the ability of listening to the transmitted messages on the channels of client i. Baseline represents the original setting of our distributed network such as the local dataset maintained by clients set in (16) and the structure of the network shown in Fig. 2. The task means if we do modification on one client's local dataset (local model) as a new case, could the onlooker notice that we did such modification according to all transmitted observations collected from our baseline and the new cases. This could be the same spirit of the differential privacy shown in Sect. 2.2.

Shortly, now we assume the onlooker can train a machine learning classifier model from the two datasets (one is collected from original network, i.e., our baseline with label 0, another dataset is collected from the case that we do a little change on one client's local model with label 1). If he uses the model he

learned on these two datasets, we want to see that he still can't distinguish whether local model has been changed from the outputs of his learning model, which could confirm our method could make the onlooker harder to deduce the local model of clients.

We can see that all the observations collected by the onlooker have time–series property. Here, we let the training model trained by the onlooker be the inception model [32] by importing package fast–ai[1] in the implementation, then he can predict the data from which classes according to the output of the learning model.

The performance of the onlooker is related to how we define the training dataset and testing dataset collected by the onlooker. Moreover, we define the weak onlooker and the strong onlooker. The details are shown in the following experimental subsection.

4.3 Classification on Time–Series Data

As the new onlooker we designed in Sect. 4.2 under the above setting shown in (16) and (17), i.e., our baseline, we assume all time–series observations transmitted on the channels belonging to class 0, and all the observations transmitted on the channels belonging to class 1 under the case that we did change on the local model of client 5.

Then we change the local model of client 5 increasingly to see the classification results by weak onlooker and strong onlooker under Algorithm 3 and Algorithm 4. The change on the local model of client 5 is shown in Table 1 and Table 2.

Weak Onlooker. Table 1 shows the form of the testing dataset and training dataset for the weak onlooker's classification model. We generate new local models by changing the local model of client 5 from our baseline (Case 0). And it shows the variation on the optimal global model $L(w^*)$ and the optimal solution w^* by changing the local model $l_5(w)$.

For the training dataset, we respectively run 100 times of Algorithm 3 and Algorithm 4 with 100 rounds for our baseline (Case 0) to get 100 data with label 0, then we respectively run 20 times of these 2 algorithms with 100 rounds for Case 1–5 to get totally 100 data with label 1. For the testing dataset, we run 100 times of the 2 algorithms with 100 rounds for Case 0 to get 100 data with label 0, then we get another 100 data with label 1 for Case 6 with the same way. Therefore, we finish the collections of the dataset for the onlooker.

To visualize the classification, we use the UMAP[2] to do the dimensional reduction and the data point in the leftmost column of Fig. 4 represents all the observations from the channels got from one single execution(time) of the

[1] https://github.com/tcapelle/timeseries_fastai.

[2] UMAP is a general purpose manifold learning and dimension reduction algorithm: https://umap-learn.readthedocs.io/en/latest/basic_usage.html.

Table 1. Weak onlooker

Samples	Case	Local function of client 5	w^*	$L(w^*)$	Label
Training dataset (200 data)					
100	0 (baseline)	$w^2 + w + 0.25$	2	5	**0**
20	1	$w^2 + 0.4w + 1.75$	2.06	5.28	**1**
20	2	$w^2 - 0.2w + 3.25$	2.12	5.528	**1**
20	3	$w^2 - 0.8w + 4.75$	2.18	5.74	**1**
20	4	$w^2 - 1.4w + 6.25$	2.24	5.91	**1**
20	5	$w^2 - 2w + 7.75$	2.3	6.05	**1**
Testing dataset (200 data)					
100	0	$w^2 + w + 0.25$	2	5	**0**
100	6	$w + 0.5w + 1.2625$	2.05	5	**1**

Table 2. Strong onlooker

Samples	Case	Local function of client 5	w^*	$L(w^*)$	Label
Training dataset (200 data)					
100	0 (baseline)	$w^2 + w + 0.25$	2	5	**0**
100	6	$w^2 + 0.5w + 1.2625$	2.05	5	**1**
Testing dataset (200 data)					
100	0	$w^2 + w + 0.25$	2	5	**0**
100	6	$w + 0.5w + 1.2625$	2.05	5	**1**

algorithms we use, such as there are 200 data points which each of them has the length of 1600 values in our experiment due to there are 8 links in Fig. 2.

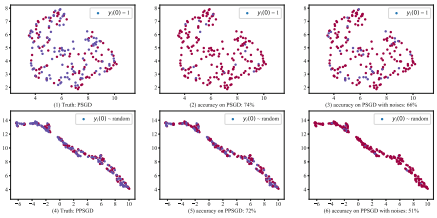


Fig. 4. Classification accuracy by weak onlooker.

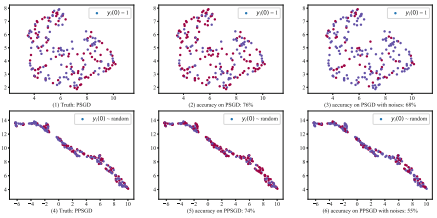


Fig. 5. Classification accuracy by strong onlooker.

The ground truth of the classification is shown in Fig. 4(1) and (4) where the red points belong to class 0, violet points belong to class 1 on Algorithm 3 and Algorithm 4. From the horizontal direction in Fig. 4, the rightmost column plots (3) and (6) represent the classification accuracy results by adding the Gaussian

noises on the transmitted observations with the accuracy as 66%, 51%, respectively. And the middle two column plots (2) and (5) represent the classification accuracy results under the two algorithms with the accuracy as 74%, 72%. From the vertical direction, the upper three plots (1), (2), (3) are the classification results under Algorithm 3, and the lower three plots (4), (5), (6) represent the classification results under Algorithm 4.

Thus, we observe that the classification accuracy under Algorithm 3 is higher than it under our proposed Algorithm 4, which means our private method can make the onlooker harder to decode the local model without harming the accuracy of the result from the cross–silo FL algorithms, i.e., the global optimum. Adding Gaussian noises can somehow protect privacy but also sacrifice the accuracy of the result.

Strong Onlooker. In the similar way, we define the dataset for the strong onlooker shown in Table 2, in which the training dataset and testing dataset are from the same two cases. Figure 5 shows the classification accuracy by the strong onlooker. We observe the similar conclusions with the weak onlooker, comparing to Fig. 4, it shows that the classification accuracy by the strong onlooker is higher than the results from the weak onlooker, i.e., the weak onlooker is harder to decode the local model privacy of clients.

4.4 The Influence on PPSGD from the Distribution of $y_i(0)$

Table 3. The distribution of $y_i(0)$

No.	Distribution of $y_i(0)$	Exp	Var	Algorithm
1	$y_i(0) = 1$	\	\	PSGD
2	$y_i(0) = 1$ Adding noises $\sim N(0, 0.065)$	\	\	PSGD
3	$y_i(0) \sim U(0.5, 1.5)$	1	0.083	PPSGD
4	$y_i(0) \sim U(0.3, 1.7)$	1	0.163	PPSGD
5	$y_i(0) \sim U(0.1, 1.9)$	1	0.27	PPSGD
6	$y_i(0) \sim Exp(1)$	1	1	PPSGD
7	$y_i(0) \sim Lognormal(-1, 2)$	1	6.39	PPSGD

To observe the influence of $y_i(0)$ for the classification accuracy from the strong onlooker's model on the test dataset, we choose to test 20000 data on a strong onlooker's model to do classification. Then we choose to randomly generate $y_i(0)$ from three kinds of distributions and add Gaussian noises to data when $y_i(0) = 1$ to see the corresponding classification results, the details are shown in Tab. 3.

Here, we only focus on the transmitted data on the channels related to client 1, and for one single execution only with 3 rounds which means the shape of the

training and test dataset is (20000, 18), note that we set the number of epochs as 10 in the learning model.

Figure 6(a) and (b) show that the relation of the training loss and accuracy is inverse. The higher classification accuracy means the onlooker can decode better to more accurately distinguish two classes, instead easier to decode the local privacy. PPSGD can make the onlooker harder to decode more privacy of local model and can't harm the accuracy results from the algorithm than the traditional privacy–preserving way like adding noises (see the red dashed lines and green lines with square marker).

From the results under the cases we randomly generate $y_i(0)$ from different distributions, we observe that when the expectation of $y_i(0)$ is equal to 1, the larger the variance of $y_i(0)$ is, the smaller the classification accuracy is, instead the more secure to protect local privacy, and harder for the onlooker to distinguish two classes (see the lines except red lines).

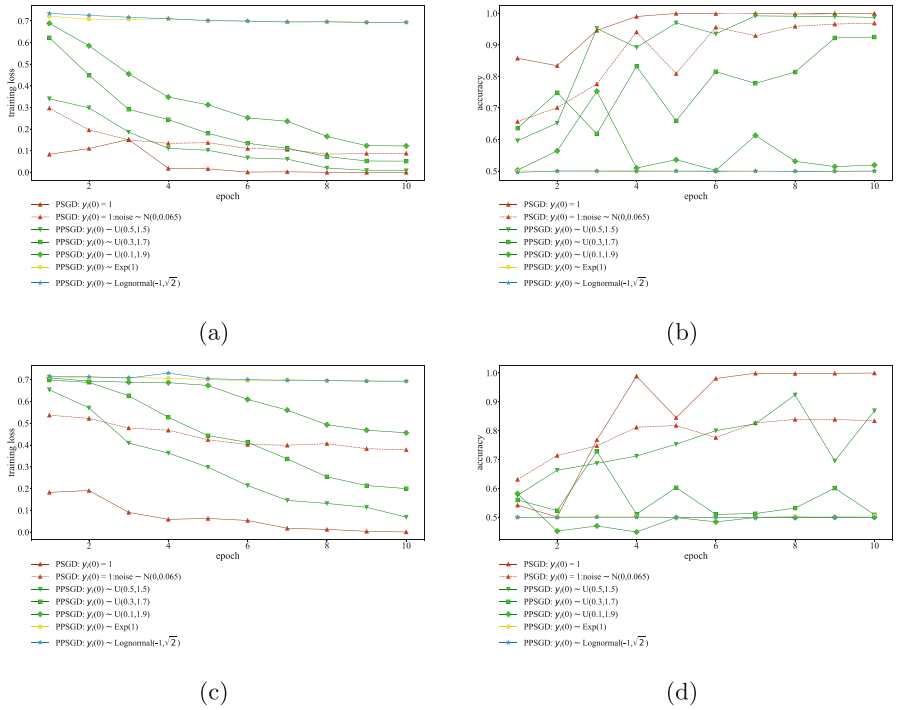


Fig. 6. (a), (b) represent the evolution of training loss and accuracy of the training model of the onlooker with 10 epochs, respectively; (c), (d) represent the evolution of training loss and accuracy of the training model of the onlooker with 10 epochs by removing $y_i(t)$, respectively. (Color figure online)

Then we do the same experiment but removing the information about $y_i(t)$ from the dataset, we want to see whether the onlooker considers the element

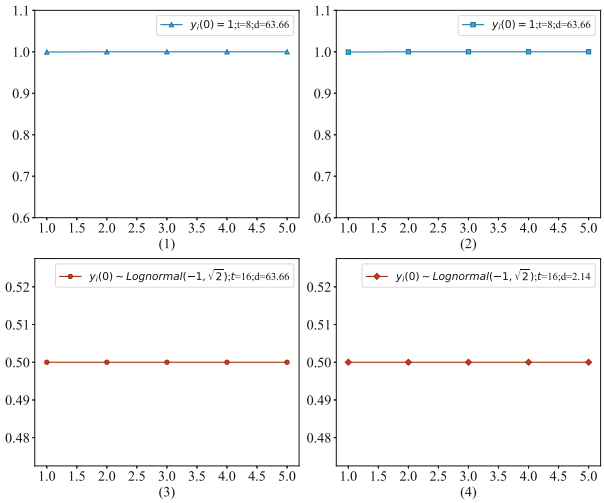


Fig. 7. Classification results on multivariate data.

about $y_i(t)$ for its decoding classifier algorithm in its learning process. The results are shown in Fig. 6(c) and (d). Comparing to Fig. 6(a) and (b), we infer that there is a relation between $x_i(t)$ and $y_i(t)$ in the learning model of the onlooker. And removing the information about $y_i(t)$ from the dataset makes the onlooker harder to do classification correctly. While it won't influence the results under the case $y_i(0) = 1$ (PSGD) (see the red lines with triangle marker in Fig. 6(b) and Fig. 6(d)) which shows the onlooker doesn't need to consider $y_i(t)$ in its learning algorithm and still could accurately distinguish the two classes.

4.5 Classification on Multivariate Data

To show the performance of the onlooker on multivariate data, we change the network as a ring with 10 nodes. We choose the Boston Housing Dataset[3] to simulate clients to do multi–feature linear regression on this dataset with 5 epochs. Shortly, in the same way, to see the performance of the model trained by the onlooker on multivariate data, we respectively do experiments under different cases.

Figure 7 shows that the classification results by the onlooker under PSGD and PPSGD, d represents the variation of the local optimum on one client's local model, t represents the number of rounds at each execution. Obviously, it shows that our private method makes the onlooker harder to decode the local model privacy of clients.

[3] Boston House Dataset:https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_bost-on.html.

5 Conclusion

In our work, we first focus on studying FL algorithms under cross–device and cross–silo FL settings, and we propose the concept of local model privacy. We demonstrate that these algorithms can not protect the local model privacy for clients when they train a linear regression task with the assumption of existing a curious onlooker. We improve one cross–silo FL algorithm, experimentally show the relevant performance of our private method and it makes the onlooker harder to decode the local model privacy of clients. In the future work, we will focus on study the performance on PPSGD in more complex ML problems, such as neural network.

Acknowledgment. Most of the work was finished during the master internship of the first author in Inria Sophia Antipolis, France. We would like to thank Prof. Giovanni Neglia and Dr. Chuan Xu for their ideas and suggestions. And this research is supported by the National Key R&D Program of China (2017YFB0801701 and 2017YFB0802805), the National Natural Science Foundation of China (Grants: U1936120, U1636216), Joint Fund of Ministry of Education of China for Equipment Preresearch (No. 6141A020333), the Fundamental Research Funds for the Central Universities, and the Basic Research Program of State Grid Shanghai Municipal Electric Power Company (52094019007F). Daojing He is the corresponding author of this article.

References

1. McMahan, B., Moore, E., Ramage, D., et al.: Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics. PMLR, pp. 1273–1282 (2017)
2. Ma, C., Li, J., Ding, M., Shu, F., et al.: On safeguarding privacy and security in the framework of federated learning. IEEE Network **34**(4), 242–248 (2020)
3. Wang, Z., Song, M., Zhang, Z., Song, Y., Qi, H.: Beyond inferring class representatives: user-level privacy leakage from federated learning. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, pp. 2512–2520. IEEE (2019)
4. Dwork, C., Roth, A.: The algorithmic foundations of differential privacy. Found. Trends Theor. Comput. Sci. **9**(3–4), 211–407 (2014)
5. Li, J., Khodak, M., Caldas, S., Talwalkar, A.: Differentially Private Meta-Learning. arXiv preprint arXiv:1909.05830 (2019)
6. Truex, S., Baracaldo, N., Anwar, A., Steinke, T., et al.: A hybrid approach to privacy-preserving federated learning. In: Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security, pp. 1–11. ACM (2019)
7. Fredrikson, M., Lantz, E., Jha, S., et al.: Privacy in pharmacogenetics: an end-to-end case study of personalized warfarin dosing. In: 23rd USENIX Security Symposium (USENIX Security 2014), pp. 17–32. USENIX (2014)
8. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1322–1333. ACM (2015)

9. Kairouz, P., McMahan, H.B., Avent, B., et al.: Advances and open problems in federated learning. arXiv preprint arXiv:1912.04977 (2019)
10. Apple: Designing for privacy (video and slide deck). Apple WWDC (2019). https://developer.apple.com/videos/play/wwdc2019/708
11. ai.google: Under the hood of the Pixel 2: How AI is supercharging hardware (2018). https://ai.google/stories/ai-in-hardware
12. Hard, A., Rao, K., Mathews, R., et al.: Federated learning for mobile keyboard prediction. arXiv preprint arXiv:1811.03604 (2018)
13. Musketeer: The MUSKETEER cross-domain platform will validate progress in the two industrial scenarios: SMART MANUFACTURING and HEALTH CARE (2019). http://musketeer.eu/project
14. WeBank: WeBank and Swiss resigned cooperation MOU (2019). https://finance.yahoo.com/news/webank-swiss-signed-cooperation-mou-112300218.html
15. Blondel, V.D., Hendrickx, J.M., Olshevsky, A., et al.: Convergence in multiagent coordination, consensus, and flocking. In: Proceedings of the 44th IEEE Conference on Decision and Control, pp. 2996–3000. IEEE (2005)
16. Jadbabaie, A., Lin, J., Morse, A.S.: Coordination of groups of mobile autonomous agents using nearest neighbor rules. IEEE Trans. Autom. Control **48**(6), 988–1001 (2003)
17. Tsitsiklis, J., Bertsekas, D., Athans, M.: Distributed asynchronous deterministic and stochastic gradient optimization algorithms. IEEE Trans. Autom. Control **31**(9), 803–812 (1986)
18. Gao, H., Wang, Y.: Dynamics Based Privacy Protection for Average Consensus on Directed Graphs. arXiv preprint arXiv:1812.02255 (2018)
19. Dwork, C.: Differential privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 1–12. Springer, Heidelberg (2006). https://doi.org/10.1007/11787006_1
20. Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., Naor, M.: Our data, ourselves: privacy via distributed noise generation. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 486–503. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_29
21. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: 44th Annual IEEE Symposium on Foundations of Computer Science, Proceedings, pp. 482–491. IEEE (2003)
22. Bénézit, F., Blondel, V., Thiran, P., Tsitsiklis, J., Vetterli, M.: Weighted gossip: distributed averaging using non-doubly stochastic matrices. In: 2010 IEEE International Symposium on Information Theory, pp. 1753–1757. IEEE (2010)
23. Nedic, A., Ozdaglar, A.: Distributed subgradient methods for multi-agent optimization. IEEE Trans. Autom. Control **54**(1), 48–61 (2009)
24. Boyd, S., Ghosh, A., Prabhakar, B., Shah, D.: Randomized gossip algorithms. IEEE Trans. Inf. Theory **52**(6), 2508–2530 (2006)
25. Nedić, A., Olshevsky, A.: Distributed optimization over time-varying directed graphs. IEEE Trans. Autom. Control **60**(3), 601–615 (2015)
26. Balcan, M.F., Blum, A., Fine, S., et al.: Distributed learning, communication complexity and privacy. In: Conference on Learning Theory. JMLR Workshop and Conference Proceedings, pp. 26-1 (2012)
27. Shamir, O., Srebro, N.: Distributed stochastic optimization and learning. In: 2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 850–857. IEEE (2014)

28. Tsianos, K.I., Lawlor, S., Rabbat, M.G.: Push-sum distributed dual averaging for convex optimization. In: 2012 IEEE 51st IEEE Conference on Decision and Control (CDC), pp. 5453–5458. IEEE (2012)
29. Geyer, R.C., Klein, T., Nabi, M.: Differentially private federated learning: a client level perspective. arXiv preprint arXiv:1712.07557 (2017)
30. Wei, K., Li, J., Ding, M., et al.: Federated learning with differential privacy: algorithms and performance analysis. IEEE Trans. Inf. Forensics Secur. **15**, 3454–3469 (2020)
31. Nagumey, A.: Book review: parallel and distributed computation: numerical methods. Int. J. Supercomput. Appl. **3**(4), 73–74 (1989)
32. Fawaz, H.I., Lucas, B., Forestier, G., et al.: Inceptiontime: finding alexnet for time series classification. Data Min. Knowl. Disc. **34**(6), 1936–1962 (2020)
33. Tsianos, K.I.: The Role of the Network in Distributed Optimization Algorithms: Convergence Rates, Scalability, Communication/Computation Tradeoffs and Communication Delays. McGill University Libraries (2013)
34. Fercoq, O., Qu, Z., Richtárik, P., Takáč, M.: Fast distributed coordinate descent for non-strongly convex losses. In: 2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP), pp. 1–6. IEEE (2014)
35. Tsianos, K.I., Lawlor, S., Rabbat, M.G.: Consensus-based distributed optimization: practical issues and applications in large-scale machine learning. In: 2012 50th Annual Allerton Conference on Communication, Control, and Computing (allerton), pp. 1543–1550. IEEE (2012)