

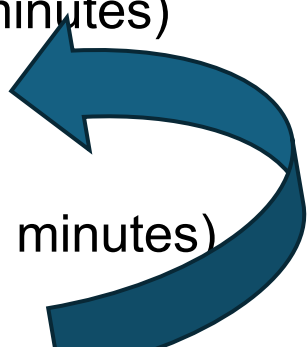
Introduction to Federated Learning and Optimization

--Starting from Distributed Learning

08/10/2023

Shuai Zhao

Content

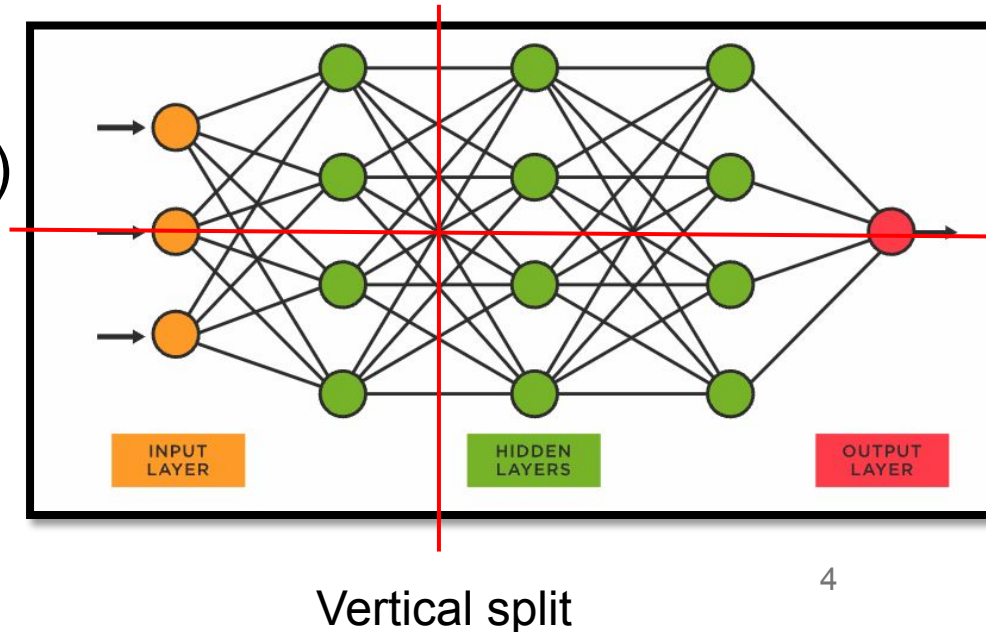
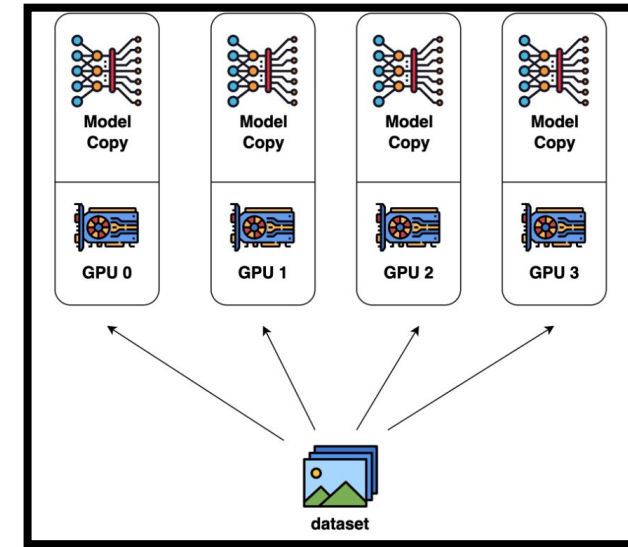
- Distributed Learning (10 minutes)
 - Why Federated Learning (FL) is important (5 minutes)
 - Relationship with Distributed Learning
 - Privacy Protection
 - Four Popular Federated Learning Methods (20 minutes)
 - FedSGD
 - Federated Averaging (FedAvg) Method
 - FedProx
 - FedNova
 - Convergence Analysis of FedAvg (10 minutes)
 - Convex setting
 - Convex non-iid setting
 - Non-Convex Setting
 - Other topics in Federated Learning (10 minutes)
 - Gradient quantization / Model Compression
 - Vertical Federated Learning
 - Client sampling
- 
- Recap the relationship between FL and Distributed Learning

Part 1: Starting from Distributed Learning

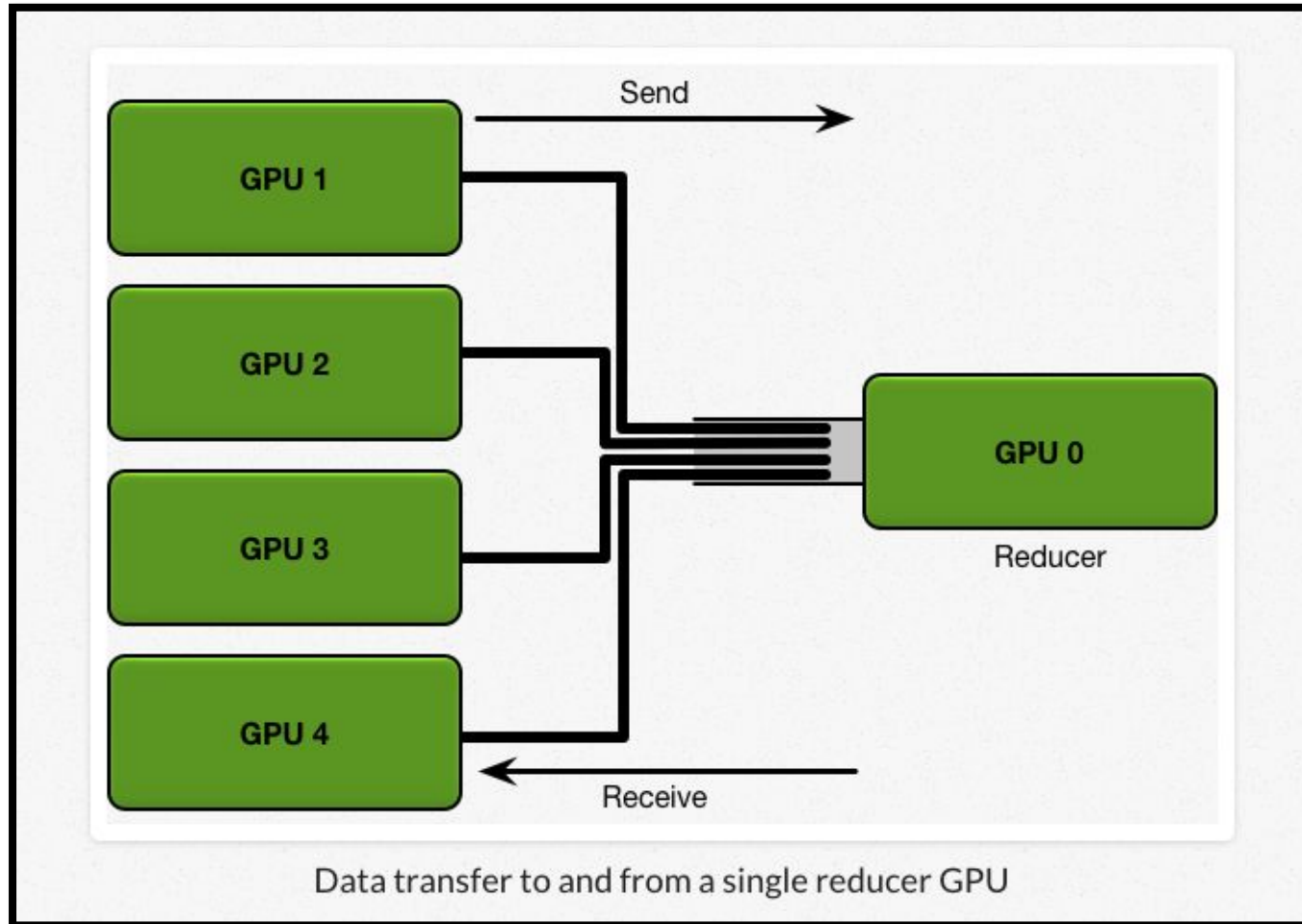
Distributed Learning

- Data Parallelism
 - Parameter Server
 - Ring Allreduce
- Model Parallelism
 - Vertical Parallelism (Pipeline Parallelism)
 - Horizontal Parallelism (Tensor Parallelism)

Horizontal split



Data Parallelism – Parameter Server



Each GPU has a full copy of the model parameters

Problem: communication overhead at GPU 0;
More GPUs used, More communication cost

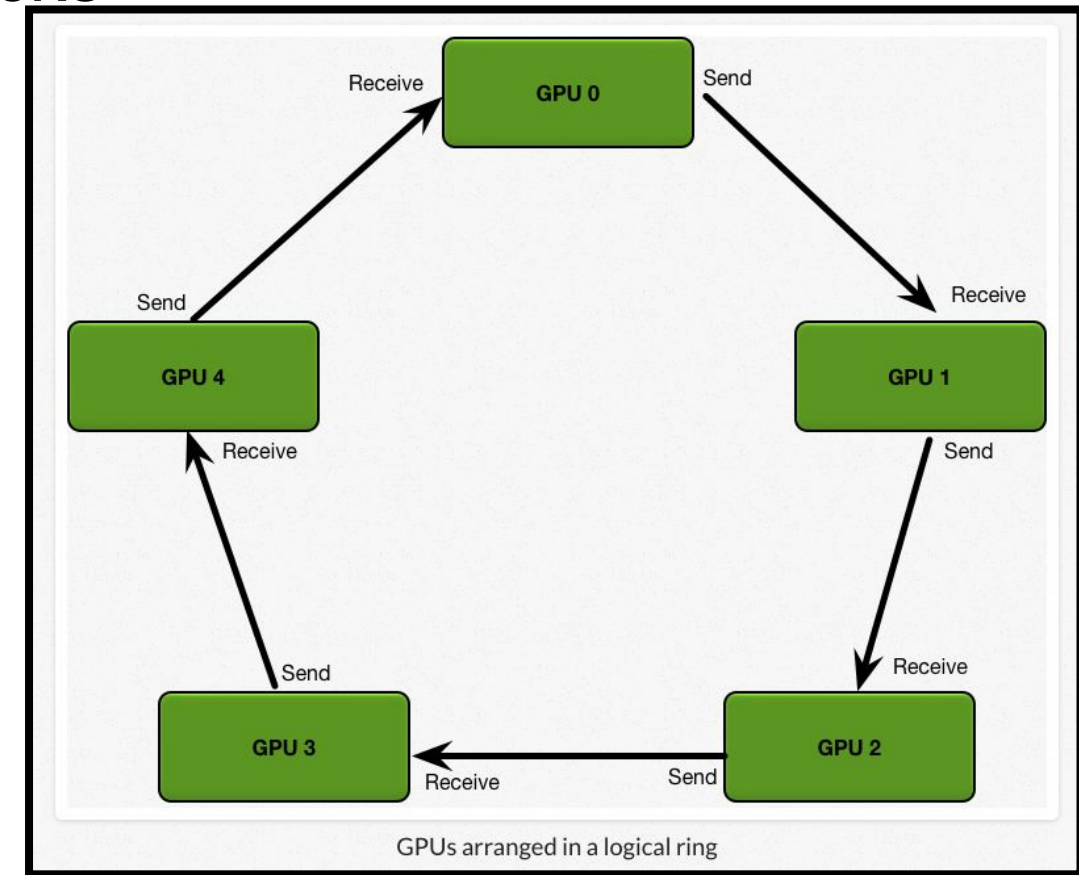
Data Transferred = $K * N$
 K is the number of client GPUs;
 N is the total number of model parameters;

Data Parallelism – Ring Allreduce (i.e., Horovod)

- Divide model parameters into K chunks
- Two steps:
 - Scatter Reduce
 - Allgather

Data Transferred = $2(K - 1) * (N / K) \approx 2*N$

Irrelevant to the number of clients K



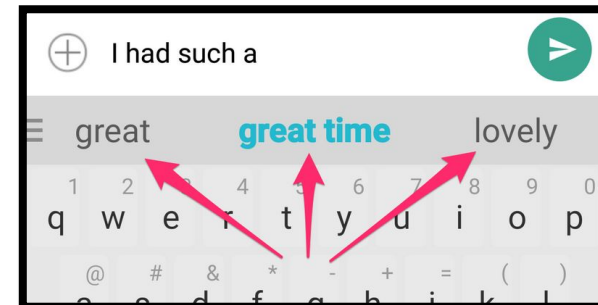
Part 2: What is FL and its relationship with DL

Federated Learning

- History:
 - FL is coined by Jakub Konečný (his advisor is Peter Richtarik) when Jakub was interning at Google under Brendan McMahan in 2016.
 - Peter Richtarik is a student of **Yurii Nesterov** (Nesterov momentum).

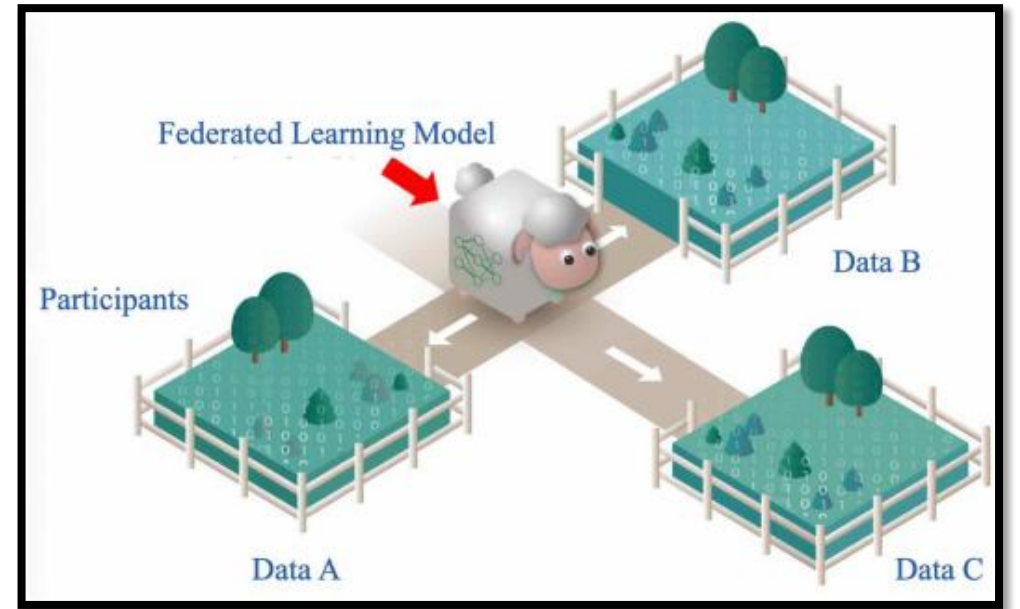
**Optimization! Optimization!!
Optimization!!!**

- First Problem Setting:
 - next-word prediction under the constraint that the data of each user must dwell at her own device.



Key Points of Federated Learning

- Federated Learning is distributed learning (data parallelism settings) with the following constraints:
 - Non-IID
 - Unbalanced
 - Data cannot leave out of the device
 - Massively distributed
 - Limited communication



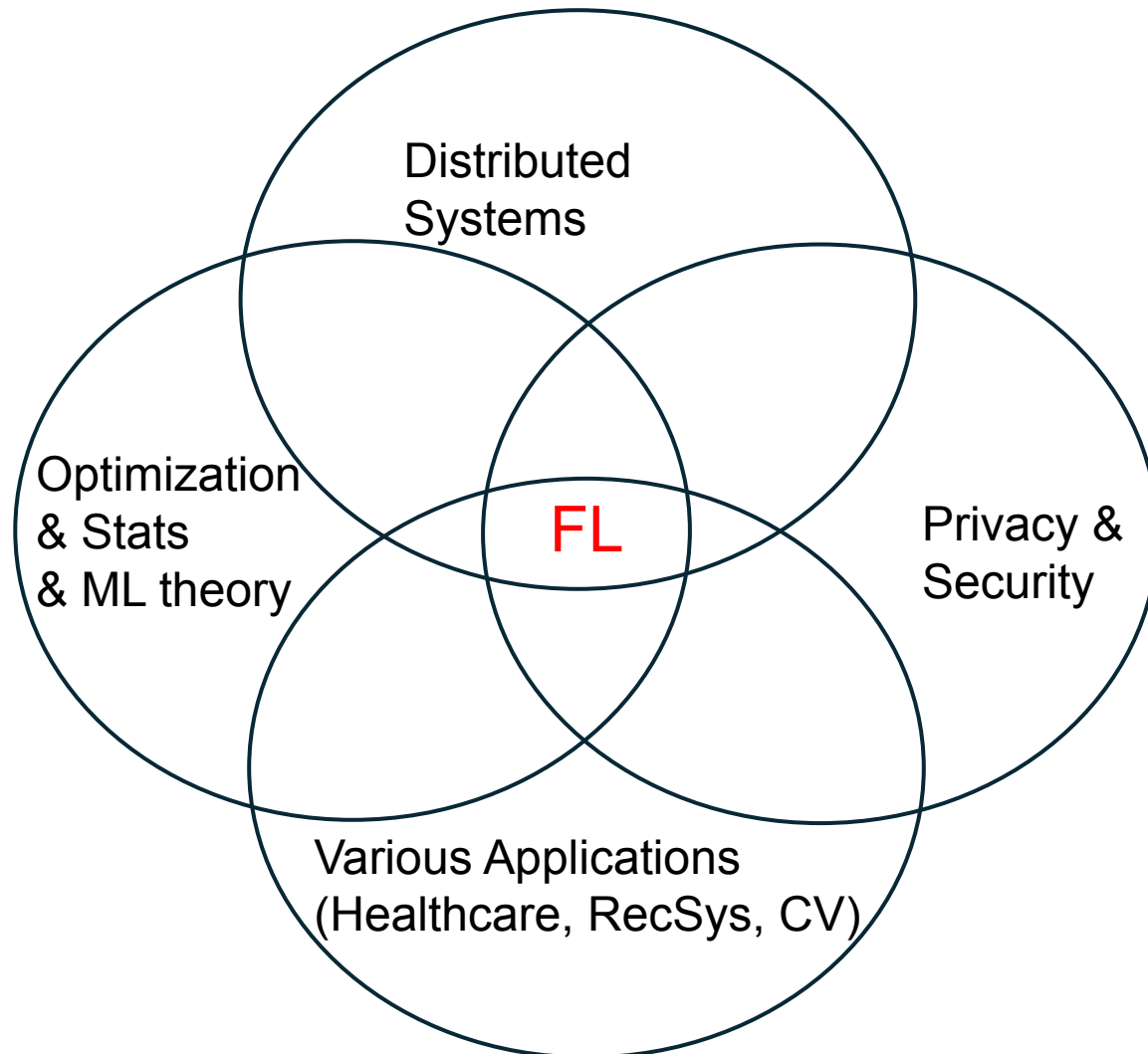
Picture from Prof. Qiang Yang

Why FL is important??

- On the application side, it protect member privacy
 - GDPR and DMA law in EU
 - CCPA law in CA state
- On the theory side, it raises a few interesting questions upon distributed learning
 - How to save communication cost?
 - Algorithm convergence analysis
 - Model compression/gradient quantization
 - How to deal with non-iid and unbalanced distribution?
 - Client sampling

In fact, FL is regarded as an extension of distributed learning.

Why Federated Learning is HOT?



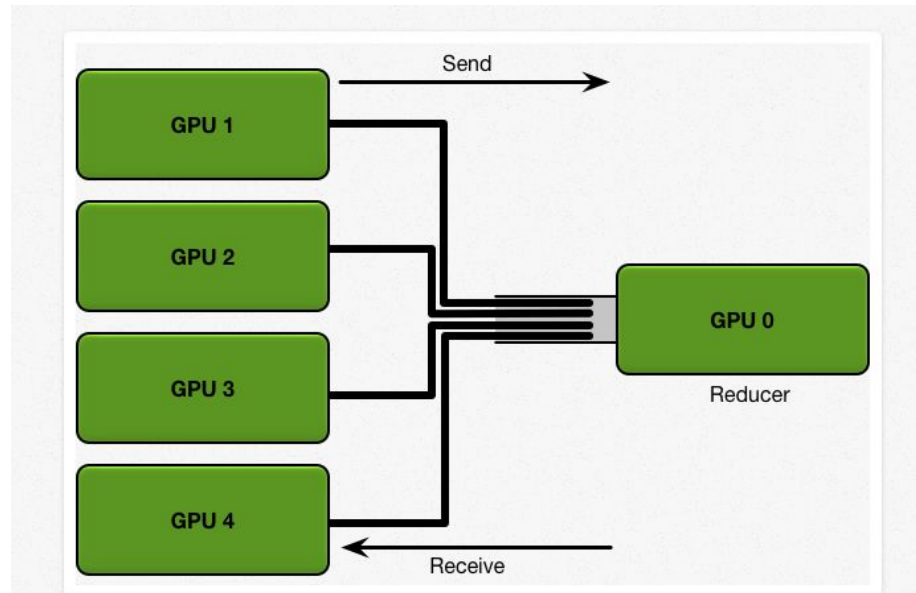
FL is overlapped by many research domains.

Part 3: Popular FL algorithms

To be clear, FL is a model training strategy, i.e., model agnostic.

The Naïve Method (FedSGD)

- Sample a fraction C of K clients
- Each sampled client does one batch of gradient update, send the new parameter to server
- Server averages all received parameters and broadcast to all clients



Data transfer to and from a single reducer GPU

Similar to the Parameter Server

Problem of FedSGD

- Too Slow!
 - Communicate is needed for each batch update.
 - Large communication cost between a client to the server
- Can we only communicate after a few epochs?



Federated Averaging

Federated Averaging (FedAvg)

Algorithm 1 FederatedAveraging. The K clients are indexed by k ; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:

```
initialize  $w_0$ 
for each round  $t = 1, 2, \dots$  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

In distributed learning, FedAvg is a new name of an old method called “local SGD” or “Parallel restarted SGD”.

$$\mathbf{w}_k^{(t+1)} = \begin{cases} \mathbf{w}_k^{(t)} - \eta_t g_k(\mathbf{w}_k^{(t)}, \xi_k^{(t)}) & \text{for } (t+1) \bmod \tau \neq 0 \\ \frac{1}{m} \sum_{j \in S^{(t)}} \left(\mathbf{w}_j^{(t)} - \eta_t g_j(\mathbf{w}_j^{(t)}, \xi_j^{(t)}) \right) \triangleq \bar{\mathbf{w}}^{(t+1)} & \text{for } (t+1) \bmod \tau = 0 \end{cases}$$

Borrow local SGD Idea From Distribution Learning to FL and give it a new name called FedAvg.

Problems of FedAvg => FedProx

- It doesn't consider data non-iid, a heuristic twist is to add a proximal term (like a regularization).

Algorithm 2 FedProx (Proposed Framework)

Input: $K, T, \mu, \gamma, w^0, N, p_k, k = 1, \dots, N$

for $t = 0, \dots, T - 1$ **do**

 Server selects a subset S_t of K devices at random (each device k is chosen with probability p_k)

 Server sends w^t to all chosen devices

 Each chosen device $k \in S_t$ finds a w_k^{t+1} which is a γ_k^t -inexact minimizer of: $w_k^{t+1} \approx$

$\arg \min_w h_k(w; w^t) = F_k(w) + \frac{\mu}{2} \|w - w^t\|^2$

 Each device $k \in S_t$ sends w_k^{t+1} back to the server

 Server aggregates the w 's as $w^{t+1} = \frac{1}{K} \sum_{k \in S_t} w_k^{t+1}$

end for

FedProx is indeed to tune the learning rate

B.1 SGD with Proximal Updates

In this case, we can write the update rule of local models as follows:

$$\mathbf{x}_i^{(t, \tau_i)} = \mathbf{x}_i^{(t, \tau_i - 1)} - \eta \left[g_i(\mathbf{x}_i^{(t, \tau_i - 1)}) + \mu \left(\mathbf{x}_i^{(t, \tau_i - 1)} - \mathbf{x}^{(t, 0)} \right) \right]. \quad (31)$$

Subtracting $\mathbf{x}_i^{(t, 0)}$ on both sides, we obtain

$$\mathbf{x}_i^{(t, \tau_i)} - \mathbf{x}^{(t, 0)} = \mathbf{x}_i^{(t, \tau_i - 1)} - \mathbf{x}^{(t, 0)} - \eta \left[g_i(\mathbf{x}_i^{(t, \tau_i - 1)}) + \mu \left(\mathbf{x}_i^{(t, \tau_i - 1)} - \mathbf{x}^{(t, 0)} \right) \right] \quad (32)$$

$$= (1 - \eta\mu) \left(\mathbf{x}_i^{(t, \tau_i - 1)} - \mathbf{x}^{(t, 0)} \right) - \eta g_i(\mathbf{x}_i^{(t, \tau_i - 1)}). \quad (33)$$

Repeating the above procedure, it follows that

$$\Delta_i^{(t)} = \mathbf{x}_i^{(t, \tau_i)} - \mathbf{x}^{(t, 0)} = -\eta \sum_{k=0}^{\tau_i - 1} (1 - \eta\mu)^{\tau_i - 1 - k} g_i(\mathbf{x}_i^{(t, k)}). \quad (34)$$

According to the definition, we have $\mathbf{a}_i = [(1 - \alpha)^{\tau_i - 1}, (1 - \alpha)^{\tau_i - 2}, \dots, (1 - \alpha), 1]$ where $\alpha = \eta\mu$.

Biased Learning Objective of FedAvg => FedNova

Algorithm 1 FederatedAveraging. The K clients are indexed by k ; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:

```
initialize  $w_0$ 
for each round  $t = 1, 2, \dots$  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

In this section, we use a simple quadratic model to illustrate the convergence problem. Suppose that the local objective functions are $F_i(\mathbf{x}) = \frac{1}{2} \|\mathbf{x} - \mathbf{e}_i\|^2$, where $\mathbf{e}_i \in \mathbb{R}^d$ is an arbitrary vector and it is the minimum \mathbf{x}_i^* of the local objective. Consider that the global objective function is defined as

$$F(\mathbf{x}) = \frac{1}{m} \sum_{i=1}^m F_i(\mathbf{x}) = \sum_{i=1}^m \frac{1}{2m} \|\mathbf{x} - \mathbf{e}_i\|^2, \quad \text{which is minimized by } \mathbf{x}^* = \frac{1}{m} \sum_{i=1}^m \mathbf{e}_i. \quad (3)$$

Below, we show that the convergence point of FedAvg can be arbitrarily away from \mathbf{x}^* .

Lemma 1 (Objective Inconsistency in FedAvg). *For the objective function in (3), if client i performs τ_i local steps per round, then FedAvg (with sufficiently small learning rate η , deterministic gradients and full client participation) will converge to*

$$\tilde{\mathbf{x}}_{\text{FedAvg}}^* = \lim_{T \rightarrow \infty} \mathbf{x}^{(T,0)} = \frac{\sum_{i=1}^m \tau_i \mathbf{e}_i}{\sum_{i=1}^m \tau_i}, \text{ which minimizes the surrogate obj.: } \tilde{F}(\mathbf{x}) = \frac{\sum_{i=1}^m \tau_i F_i(\mathbf{x})}{\sum_{i=1}^m \tau_i}.$$

FedNova: A unified equation

more general form as follows:

$$\mathbf{x}^{(t+1,0)} - \mathbf{x}^{(t,0)} = -\tau_{\text{eff}} \sum_{i=1}^m w_i \cdot \eta \mathbf{d}_i^{(t)}, \quad \text{which optimizes } \tilde{F}(\mathbf{x}) = \sum_{i=1}^m w_i F_i(\mathbf{x}). \quad (4)$$

The three key elements τ_{eff} , w_i and $\mathbf{d}_i^{(t)}$ of this update rule take different forms for different algorithms. Below, we provide detailed descriptions of these key elements.

1. **Normalized gradient $\mathbf{d}_i^{(t)}$:** The normalized gradient is defined as $\mathbf{d}_i^{(t)} = \mathbf{G}_i^{(t)} \mathbf{a}_i / \|\mathbf{a}_i\|_1$, where $\mathbf{G}_i^{(t)} = [g_i(\mathbf{x}_i^{(t,0)}), g_i(\mathbf{x}_i^{(t,1)}), \dots, g_i(\mathbf{x}_i^{(t,\tau_i)})] \in \mathbb{R}^{d \times \tau_i}$ stacks all stochastic gradients in the t -th round, and $\mathbf{a}_i \in \mathbb{R}^{\tau_i}$ is a non-negative vector and defines how stochastic gradients are locally accumulated. The normalizing factor $\|\mathbf{a}_i\|_1$ in the denominator is the ℓ_1 norm of the vector \mathbf{a}_i . By setting different \mathbf{a}_i , (4) works for most common client optimizers such as SGD with proximal updates, local momentum, and variable learning rate, and more generally, any solver whose accumulated gradient $\Delta_i^{(t)} = -\eta \mathbf{G}_i^{(t)} \mathbf{a}_i$, a linear combination of local gradients.
If the client optimizer is vanilla SGD (*i.e.*, the case of FedAvg), then $\mathbf{a}_i = [1, 1, \dots, 1] \in \mathbb{R}^{\tau_i}$ and $\|\mathbf{a}_i\|_1 = \tau_i$. As a result, the normalized gradient is just a simple average of all stochastic gradients within current round: $\mathbf{d}_i^{(t)} = \mathbf{G}_i^{(t)} \mathbf{a}_i / \tau_i = \sum_{k=0}^{\tau_i-1} g_i(\mathbf{x}_i^{(t,k)}) / \tau_i$. Later in this section, we will present more specific examples on how to set \mathbf{a}_i in other algorithms.
2. **Aggregation weights w_i :** Each client's normalized gradient \mathbf{d}_i is multiplied with weight w_i when computing the aggregated gradient $\sum_{i=1}^m w_i \mathbf{d}_i$. By definition, these weights satisfy $\sum_{i=1}^m w_i = 1$. Observe that these weights determine the surrogate objective $\tilde{F}(\mathbf{x}) = \sum_{i=1}^m w_i F_i(\mathbf{x})$, which is optimized by the general algorithm in (4) instead of the true global objective $F(\mathbf{x}) = \sum_{i=1}^m p_i F_i(\mathbf{x})$ – we will prove this formally in Theorem 1.
3. **Effective number of steps τ_{eff} :** Since client i makes τ_i local updates, the average number of local SGD steps per communication round is $\bar{\tau} = \sum_{i=1}^m \tau_i / m$. However, the server can scale up or scale down the effect of the aggregated updates by setting the parameter τ_{eff} larger or smaller than $\bar{\tau}$ (analogous to choosing a global learning rate [25, 40]). We refer to the ratio $\bar{\tau} / \tau_{\text{eff}}$ as the slowdown, and it features prominently in the convergence analysis presented in Section 4.2.

Recap the relationship between FL and distributed learning

- copy-and-paste research between federated learning and distributed learning.

kill two birds with one stone



One idea, Two papers!

Sebastian U. Stich, Don't Use Large Mini-Batches, Use Local SGD, ICLR 2020 (Purely Experiments)

Mini-batch SGD. For a sum-structured optimization problem of the form $\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{w})$ where $\mathbf{w} \in \mathbb{R}^d$ denotes the parameters of the model and $f_i: \mathbb{R}^d \rightarrow \mathbb{R}$ the loss function of the i -th training example, the mini-batch SGD update for $K \geq 1$ workers is given as

$$\mathbf{w}_{(t+1)} := \mathbf{w}_{(t)} - \gamma_{(t)} \left[\frac{1}{K} \sum_{k=1}^K \frac{1}{B} \sum_{i \in \mathcal{I}_{(t)}^k} \nabla f_i(\mathbf{w}_{(t)}) \right], \quad (1)$$

where $\gamma_{(t)} > 0$ denotes the learning rate and $\mathcal{I}_{(t)}^k \subseteq [N]$ the subset (mini-batch) of training datapoints selected by worker k (typically selected uniformly at random from the locally available datapoints on worker k). For convenience, we will assume the same batch size B per worker.

Local SGD. Motivated to better balance the available system resources (computation vs. communication), local SGD (a.k.a. local-update SGD, parallel SGD, or federated averaging) has recently attracted increased research interest (McDonald et al., 2009; Zinkevich et al., 2010; McDonald et al., 2010; Zhang et al., 2014; 2016; McMahan et al., 2017). In local SGD, each worker $k \in [K]$ evolves a local model by performing H sequential SGD updates with mini-batch size B_{loc} , before communication (synchronization by averaging) among the workers. Formally,

$$\mathbf{w}_{(t)+h+1}^k := \mathbf{w}_{(t)+h}^k - \gamma_{(t)} \left[\frac{1}{B_{\text{loc}}} \sum_{i \in \mathcal{I}_{(t)+h}^k} \nabla f_i(\mathbf{w}_{(t)+h}^k) \right], \quad \mathbf{w}_{(t+1)}^k := \frac{1}{K} \sum_{k=1}^K \mathbf{w}_{(t)+H}^k, \quad (2)$$

where $\mathbf{w}_{(t)+h}^k$ denotes the local model on machine k after t global synchronization rounds and subsequent $h \in [H]$ local steps ($\mathcal{I}_{(t)+h}^k$ is defined analogously). Mini-batch SGD is a special case

if $H = 1$ and $B_{\text{loc}} = B$.

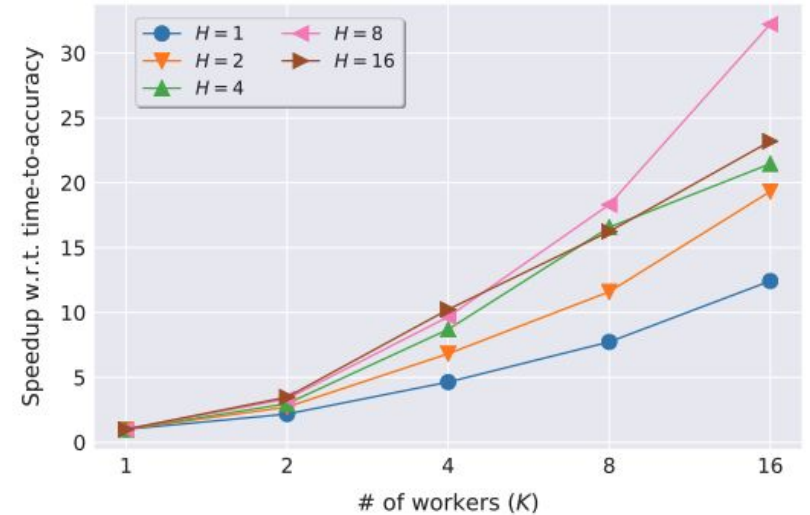


Table 1: Scaling behavior of **local SGD** in clock-time for increasing number of workers K , for different number of local steps H , for training **ResNet-20** on **CIFAR-10** with $B_{\text{loc}} = 128$. The reported **speedup** (averaged over three runs) is over single GPU training time for reaching the baseline top-1 test accuracy (91.2% as in (He et al., 2016a)). We use a 8×2 -GPU cluster with 10 Gbps network. $H = 1$ recovers mini-batch SGD.

Decentralized Federated Learning: A Segmented Gossip Approach, arXiv 2019 (124 citations)

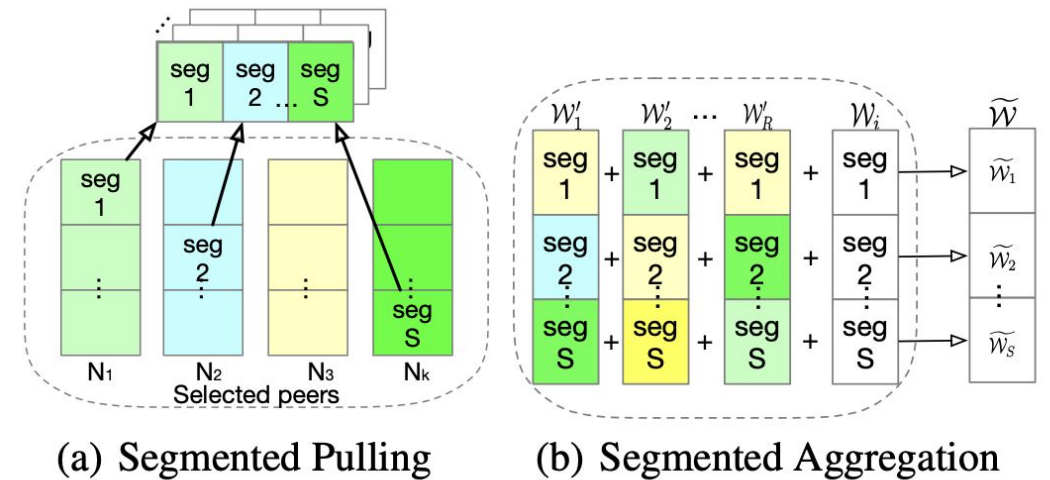
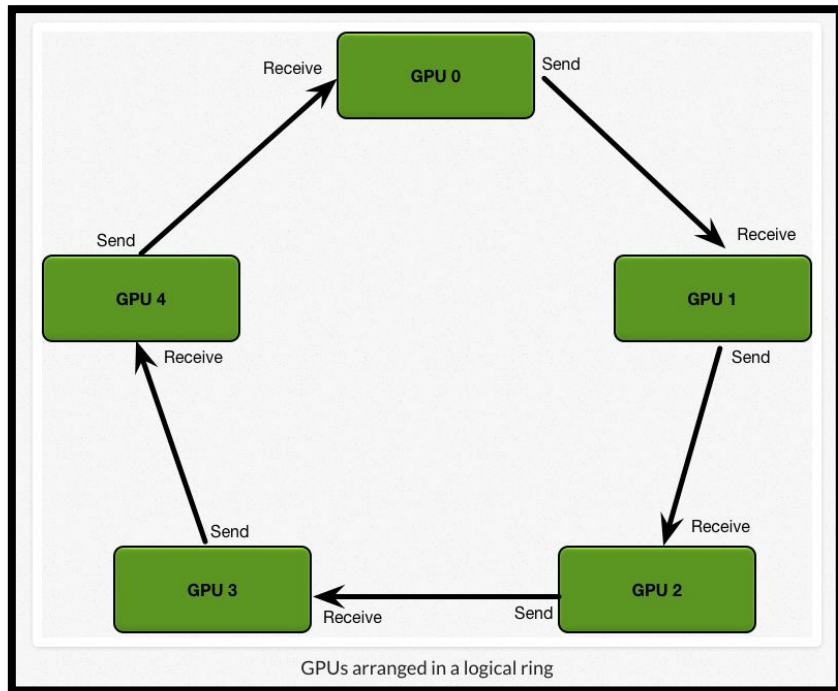


Figure 1: Segmented Gossip Aggregation

Borrow Ring Allreduce Idea From Distributed Learning to Federated Learning

And many more....

Adaption of the distributed version of optimization methods into FL

- SAG (Stochastic Average Gradient)
- SVRG (Stochastic Variance Reduced Gradient)
- ADMM (Alternating Direction Method of Multipliers)
- Proximal Point Algorithm (PPA)
- Primal Dual Method
-

Part 4: Convergence Analysis of FedAvg

- Convex Setting
- Convex Setting with non-iid
- Non-Convex Setting

Convex Setting [1]

- Assumption

- L-Smooth
- μ -Strong Convex

- Convergence

- Indications to converge faster:

- Larger batch size
- More rounds of local updates before sending to the server.

We consider finite-sum convex optimization problems $f: \mathbb{R}^d \rightarrow \mathbb{R}$ of the form

$$f(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x}), \quad \mathbf{x}^* := \arg \min_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}), \quad f^* := f(\mathbf{x}^*), \quad (2)$$

where f is L -smooth² and μ -strongly convex³. We consider K parallel mini-batch SGD sequences with mini-batch size b that are synchronized (by averaging) after at most every H iterations. For appropriate chosen stepsizes and an averaged iterate $\hat{\mathbf{x}}_T$ after T steps (for T sufficiently large, see Section 3 below for the precise statement of the convergence result with bias and variance terms) and synchronization delay $H = O(\sqrt{T/(Kb)})$ we show convergence

$$\mathbb{E} f(\hat{\mathbf{x}}_T) - f^* = O\left(\frac{G^2}{\mu b K T}\right), \quad (3)$$

with second moment bound $G^2 \geq \mathbb{E} \|\nabla f_i(\mathbf{x})\|^2$. Thus, we see that compared to parallel mini-batch SGD the communication rounds can be reduced by a factor $H = O(\sqrt{T/(Kb)})$ without hampering the asymptotic convergence. Equation (3) shows perfect linear speedup in terms of computation, but with much less communication than mini-batch SGD. The resulting speedup when taking communication cost into account is illustrated in Figure 1 (see also Section D below). Under the assumption that (3) is tight, one has thus now two strategies to improve the compute to communication ratio (denoted by ρ): (i) either to increase the mini-batch size b or (ii) to increase the communication interval H . Both strategies give the same improvement when b and H are small (linear speedup). Like mini-batch SGD that faces some limitations for $b \gg 1$ (as discussed in e.g. [8, 19, 42]), the parameter H cannot be chosen too large in local SGD. We give some practical guidelines in Section 4.

[1] Sebastian U. Stich, Local SGD Converges Fast and Communicates Little, ICLR 2019

*He is a postdoc of Yurii Nesterov

Convex and non-IID [1]

- Assumption
 - L-Smooth
 - u-Strongly Convex

Assumption 3. Let ξ_t^k be sampled from the k -th device's local data uniformly at random. The variance of stochastic gradients in each device is bounded: $\mathbb{E} \|\nabla F_k(\mathbf{w}_t^k, \xi_t^k) - \nabla F_k(\mathbf{w}_t^k)\|^2 \leq \sigma_k^2$ for $k = 1, \dots, N$.

Assumption 4. The expected squared norm of stochastic gradients is uniformly bounded, i.e., $\mathbb{E} \|\nabla F_k(\mathbf{w}_t^k, \xi_t^k)\|^2 \leq G^2$ for all $k = 1, \dots, N$ and $t = 1, \dots, T - 1$

We show in Theorem 1, 2, and 3 that FedAvg has $\mathcal{O}(\frac{1}{T})$ convergence rate. In particular, Theorem 3 shows that to attain a fixed precision ϵ , the number of communications is

$$\frac{T}{E} = \mathcal{O} \left[\frac{1}{\epsilon} \left(\left(1 + \frac{1}{K} \right) EG^2 + \frac{\sum_{k=1}^N p_k^2 \sigma_k^2 + \Gamma + G^2}{E} \right) \right]. \quad (1)$$

Here, G , Γ , p_k , and σ_k are problem-related constants defined in Section 3.1. The most interesting insight is that E is a knob controlling the convergence rate: neither setting E over-small ($E = 1$ makes FedAvg equivalent to SGD) nor setting E over-large is good for the convergence.

Non-Convex Setting [1]

- Assumptions
 - 1, L-Smooth

2. **Bounded variances and second moments:** *There exists constants $\sigma > 0$ and $G > 0$ such that*

$$\mathbb{E}_{\zeta_i \sim \mathcal{D}_i} \|\nabla F_i(\mathbf{x}; \zeta_i) - \nabla f_i(\mathbf{x})\|^2 \leq \sigma^2, \forall \mathbf{x}, \forall i$$

$$\mathbb{E}_{\zeta_i \sim \mathcal{D}_i} \|\nabla F_i(\mathbf{x}; \zeta_i)\|^2 \leq G^2, \forall \mathbf{x}, \forall i$$

- **Linear Speedup:** By part (1) of Corollary 1, Algorithm 1 with any fixed constant I has convergence rate $O(\frac{1}{\sqrt{NT}} + \frac{N}{T})$. If T is large enough, i.e., $T > N^3$, then the term $\frac{N}{T}$ is dominated by the term $\frac{1}{\sqrt{NT}}$ and hence Algorithm 1 has convergence rate $O(\frac{1}{\sqrt{NT}})$. That is, our algorithm achieves a linear speed-up with respect to the number of workers. Such linear speedup for **stochastic non-convex optimization** was previously attained by decentralized-parallel stochastic gradient descent (D-PSGD) considered in (Lian et al. 2017) by **requiring at least $T > N^5$** . See, e.g., Corollary 2 in (Lian et al. 2017).⁴

- **Communication Reduction:** Note that Algorithm 1 requires inter-node communication only at the iterations that are multiples of I . By Corollary 1, it suffices to choose any $I \leq \frac{T^{1/4}}{N^{3/4}}$ to ensure the $O(\frac{1}{\sqrt{NT}})$ convergence of our algorithm. That is, compared with parallel mini-batch SGD or the D-PSGD in (Lian et al. 2017), the number of communication rounds in our algorithm can be reduced by a factor $\frac{T^{1/4}}{N^{3/4}}$

[1] Hao Yu, et. al, Parallel Restarted SGD with Faster Convergence and Less Communication: Demystifying Why Model Averaging Works for Deep Learning, AAAI 2019

Part 5: Other Topics in Federated Learning

- Gradient Compression: SignSGD and Error Feedback
- Client Sampling
- Vertical FL
- Partial Federated Learning
- Others

Gradient Compression: SignSGD [1] and Error Feedback [2]

Algorithm 1 SIGNSGD

Input: learning rate δ , current point x_k

$\tilde{g}_k \leftarrow \text{stochasticGradient}(x_k)$

$x_{k+1} \leftarrow x_k - \delta \text{sign}(\tilde{g}_k)$

Compression ratio: Float32 (32 bits) \rightarrow 1 bit

Problem: SignSGD forgets the magnitude and it doesn't converge under certain circumstances.

Algorithm 1 EF-SIGNSGD (SIGNSGD with Error-Feedb.)

```
1: Input: learning rate  $\gamma$ , initial iterate  $\mathbf{x}_0 \in \mathbb{R}^d$ ,  $\mathbf{e}_0 = \mathbf{0}$ 
2: for  $t = 0, \dots, T - 1$  do
3:    $\mathbf{g}_t := \text{stochasticGradient}(\mathbf{x}_t)$ 
4:    $\mathbf{p}_t := \gamma \mathbf{g}_t + \mathbf{e}_t$   $\triangleright$  error correction
5:    $\Delta_t := (\|\mathbf{p}_t\|_1 / d) \text{sign}(\mathbf{p}_t)$   $\triangleright$  compression
6:    $\mathbf{x}_{t+1} := \mathbf{x}_t - \Delta_t$   $\triangleright$  update iterate
7:    $\mathbf{e}_{t+1} := \mathbf{p}_t - \Delta_t$   $\triangleright$  update residual error
8: end for
```

Error-feedback. We demonstrate that the aforementioned problems of SIGNSGD can be fixed by i) scaling the signed vector by the norm of the gradient to ensure the magnitude of the gradient is not forgotten, and ii) locally storing the difference between the actual and compressed gradient, and iii) adding it back into the next step so that the correct direction is not forgotten. We call our 'fixed' method EF-SIGNSGD (Algorithm 1).

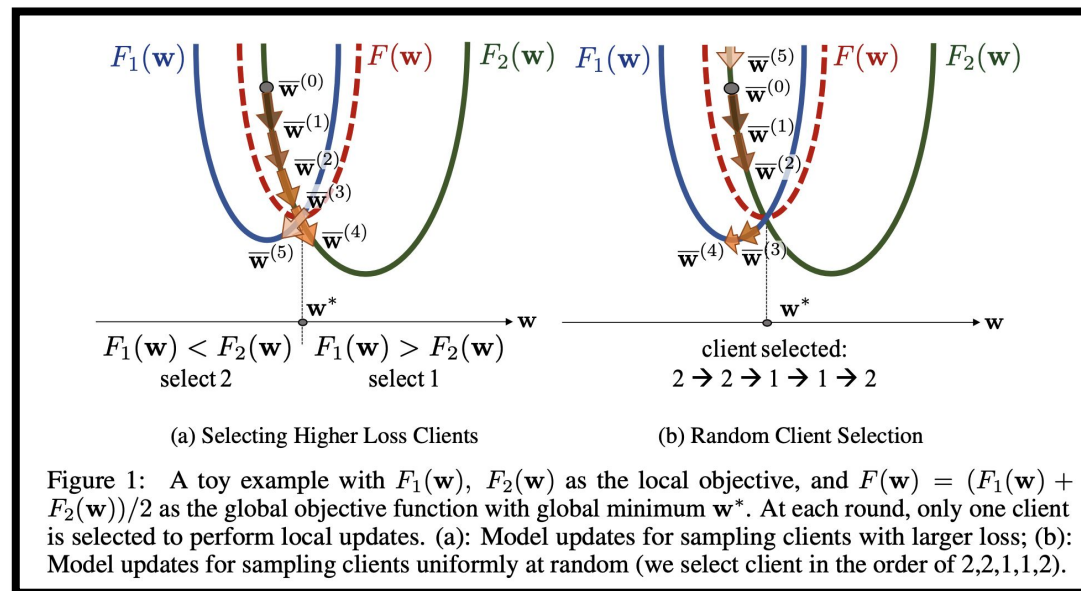
Send $\text{sign}(\mathbf{p}_t)$ and the norm of \mathbf{p}_t from clients to the server.
Notice: $\text{sign}(\mathbf{p}_t)$ is a vector; norm of \mathbf{p}_t is a scalar

[1] SignSGD: Compressed Optimization for Non-Convex Problems, ICML 2018

[2] Sebastian U. Stich, Error Feedback Fixes SignSGD and other Gradient Compression Schemes, ICML 2019

[3] Ping Li, Analysis of Error Feedback in Federated Non-convex Optimization with Biased Compression: Fast Convergence and Partial Participation, ICML 2023

(Biased) Client Sampling



Algorithm 1 Pseudo code for UCB-CS

- 1: **Input:** m, γ, p_k for $k \in [K]$
- 2: **Output:** $\mathcal{S}^{(t)}$
- 3: **Initialize:** empty sets $\mathcal{S}^{(t)}$ and list \mathcal{A} of length K
- 4: **Global server do**
- 5: Receive $\frac{1}{\tau b} \sum_{l=t-\tau+1}^t \sum_{\xi \in \xi_k^{(l)}} f(\mathbf{w}_k^{(l)}, \xi)$ from clients $k \in \mathcal{S}^{(t-1)}$ and calculate $A_t(\gamma, k)$ as (4)
- 6: Update $\mathcal{A}[k] = A_t(\gamma, k)$
- 7: Get $\mathcal{S}^{(t)} = \{m \text{ clients with largest values in } \mathcal{A} \text{ (break ties randomly)}\}$
- 8: Discount elements in \mathcal{A} by $\mathcal{A} = \gamma \mathcal{A}$
- 9: **return** $\mathcal{S}^{(t)}$

Biased client sampling convergence faster [1]



Bandit-based client sampling [2]

Problem: loss is not pre-known before a client is selected.

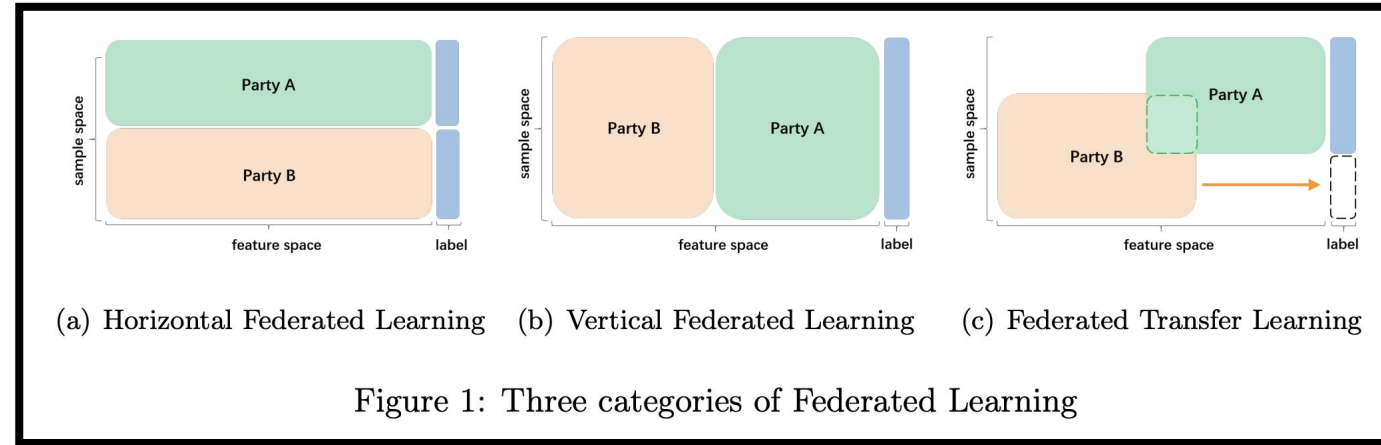
[1] Gauri Joshi, Client Selection in Federated Learning: Convergence Analysis and Power-of-Choice Selection Strategies, arXiv 2020

[2] Gauri Joshi, Bandit-based Communication-Efficient Client Selection Strategies for Federated Learning, arXiv 2020

[3] Peter Richtarik, A Guide Through the Zoo of Biased SGD, arXiv 2023

Vertical Federated Learning (VFL)

- In fact, Vertical Federated Learning [1] is the most common setting in the industry.



- FedAds, Alibaba [2]

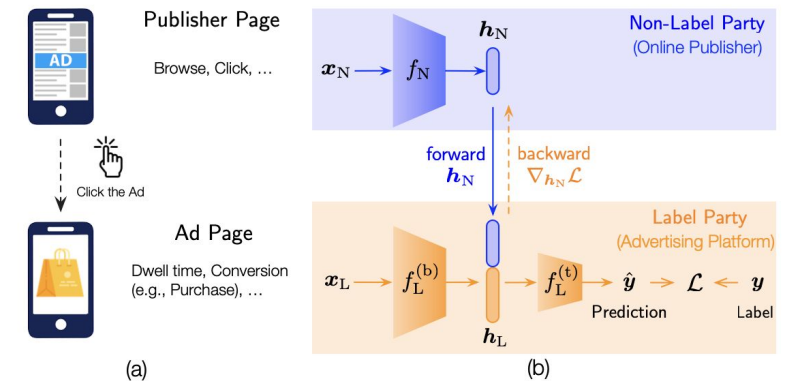


Figure 1: (a) The feedback behaviors after a user browses an ad. (b) Vertical federated learning framework for training conversion rate estimation model. Online publisher and advertising platform provide the collected user feedback data and collaboratively train the model.

[1] Qiang Yang, Vertical Federated Learning, arXiv 2020

[2] Alibaba, FedAds: A Benchmark for Privacy-Preserving CVR Estimation with Vertical Federated Learning, SIGIR 2023

[3] Xing Xie, FedCTR: Federated Native Ad CTR Prediction with Cross-platform User Behavior Data, TIST 2022

Partial FL: FL with Personalization Layers

Algorithm 1 FEDPER-CLIENT(j)

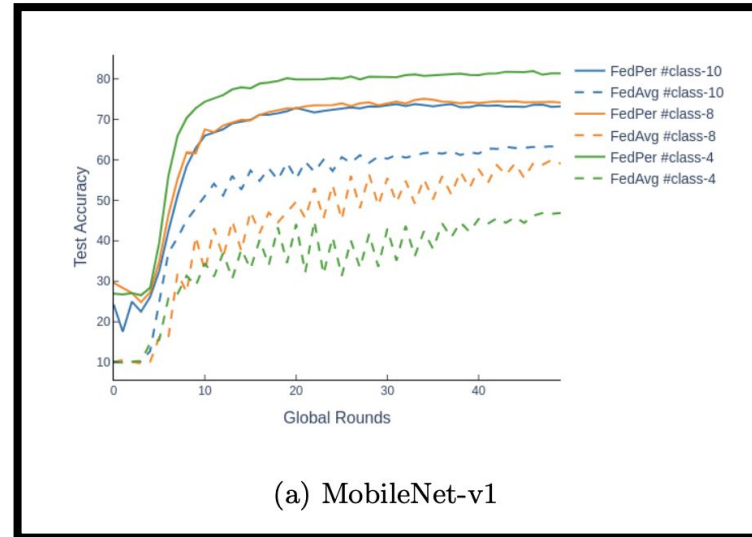
Require: $f(\cdot; \cdot, \cdot), e, b, \{(\mathbf{x}_{j,i}, y_{j,i}) \mid i \in \{1, \dots, n_j\}\}$

Require: $\eta_j^{(k)}$ for $k \in \mathbb{Z}_+$

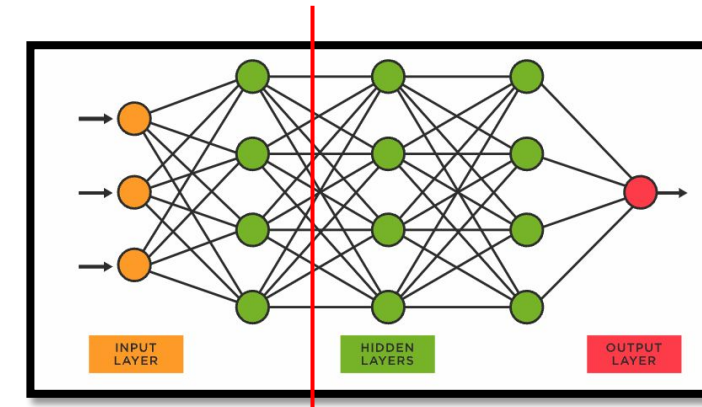
- 1: Initialize $\mathbf{W}_{P_j}^{(0)}$ at random
- 2: Send n_j to server
- 3: **for** $k = 1, 2, \dots$ **do**
- 4: Receive $\mathbf{W}_B^{(k-1)}$ from server
- 5: $(\mathbf{W}_{B,j}^{(k)}, \mathbf{W}_{P_j}^{(k)}) \leftarrow \text{SGD}_j(\mathbf{W}_B^{(k-1)}, \mathbf{W}_{P_j}^{(k-1)}, \eta_j^{(k)})$
- 6: Send $\mathbf{W}_{B,j}^{(k)}$ to server
- 7: **end for**

Algorithm 2 FEDPER-SERVER

- 1: Initialize $\mathbf{W}_B^{(0)}$ at random
- 2: Receive n_j from each client $j \in \{1, \dots, N\}$ and compute $\gamma_j = n_j / \sum_{j=1}^N n_j$
- 3: Send $\mathbf{W}_B^{(0)}$ to each client
- 4: **for** $k = 1, 2, \dots$ **do**
- 5: Receive $\mathbf{W}_{B,j}^{(k)}$ from each client $j \in \{1, \dots, N\}$
- 6: Aggregate $\mathbf{W}_B^{(k)} \leftarrow \sum_{j=1}^N \gamma_j \mathbf{W}_{B,j}^{(k)}$
- 7: Send $\mathbf{W}_B^{(k)}$ to each client
- 8: **end for**



Similar to the Vertical Model Parallelism in Distributed Learning



Vertical split

More Problem-Driven Research Directions

- Incentive of data sharing
 - Some clients are willing to share their data with certain incentives
- Some member features are public; Some member features are private
- Different communication costs for each client
- Security attacks in FL and gradient leakage of client privacy
- Fairness in FL
- Multi-task learning in FL

Questions?