Federated Learning (FL) Lab

饮 水 思 源 • 爱 国 荣 校

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目 录

01

FedAvg algorithm

- Background and contributions
- Federated learning review
- Federated settings
- Federated Averaging
- Experiments

FedAvg algorithm

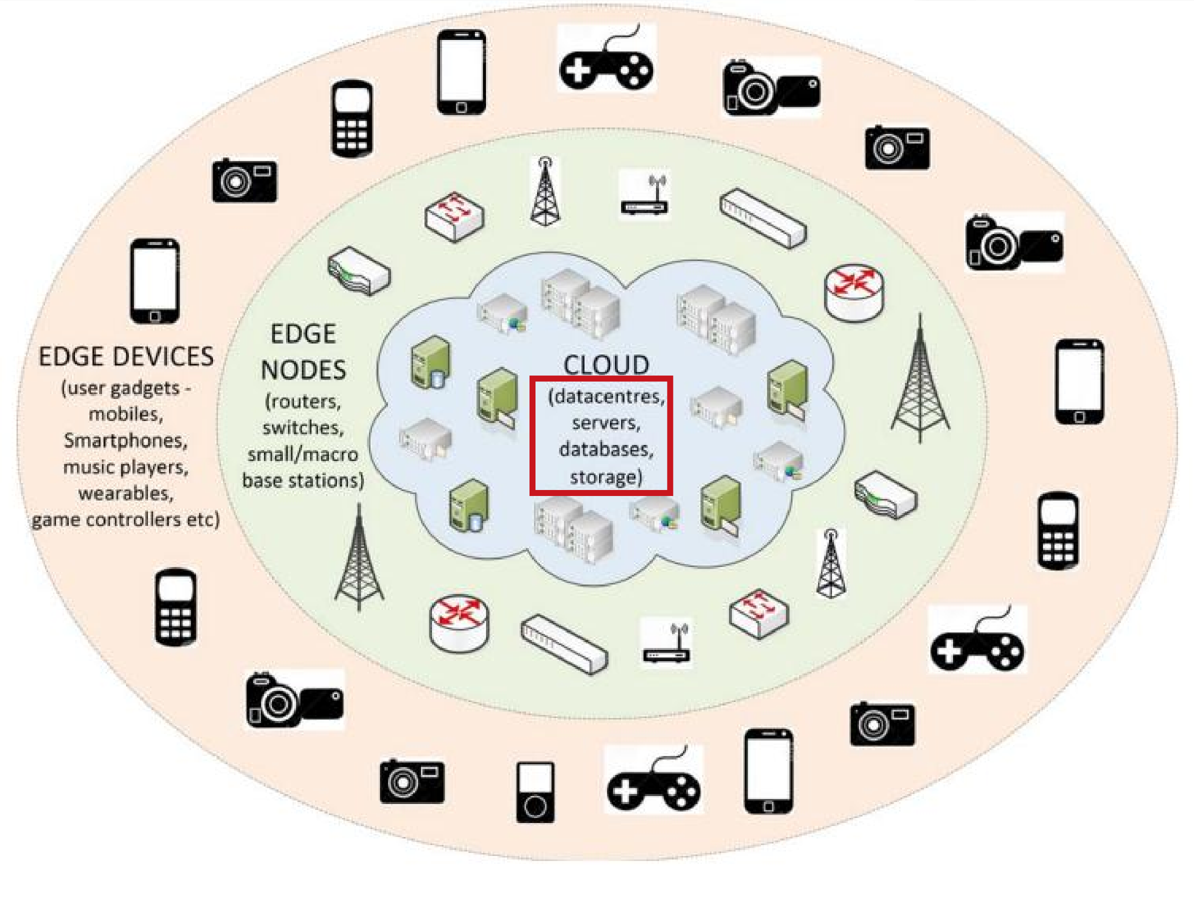
Background

- Recently, attributed to the collection of **massive data** from users or organizations, AI has been thriving for years.
- However, some **private data** is also collected during data collection, such as the shopping behaviors, facial images, house locations, etc.
- Data breach is becoming more and more severe, and many governments have issued **data privacy protecting** laws.

FedAvg algorithm

Background

- AI model training based on distributed computing and edge computing is required.



Background

- Different from traditional settings, data cannot be collected and naturally exists locally.

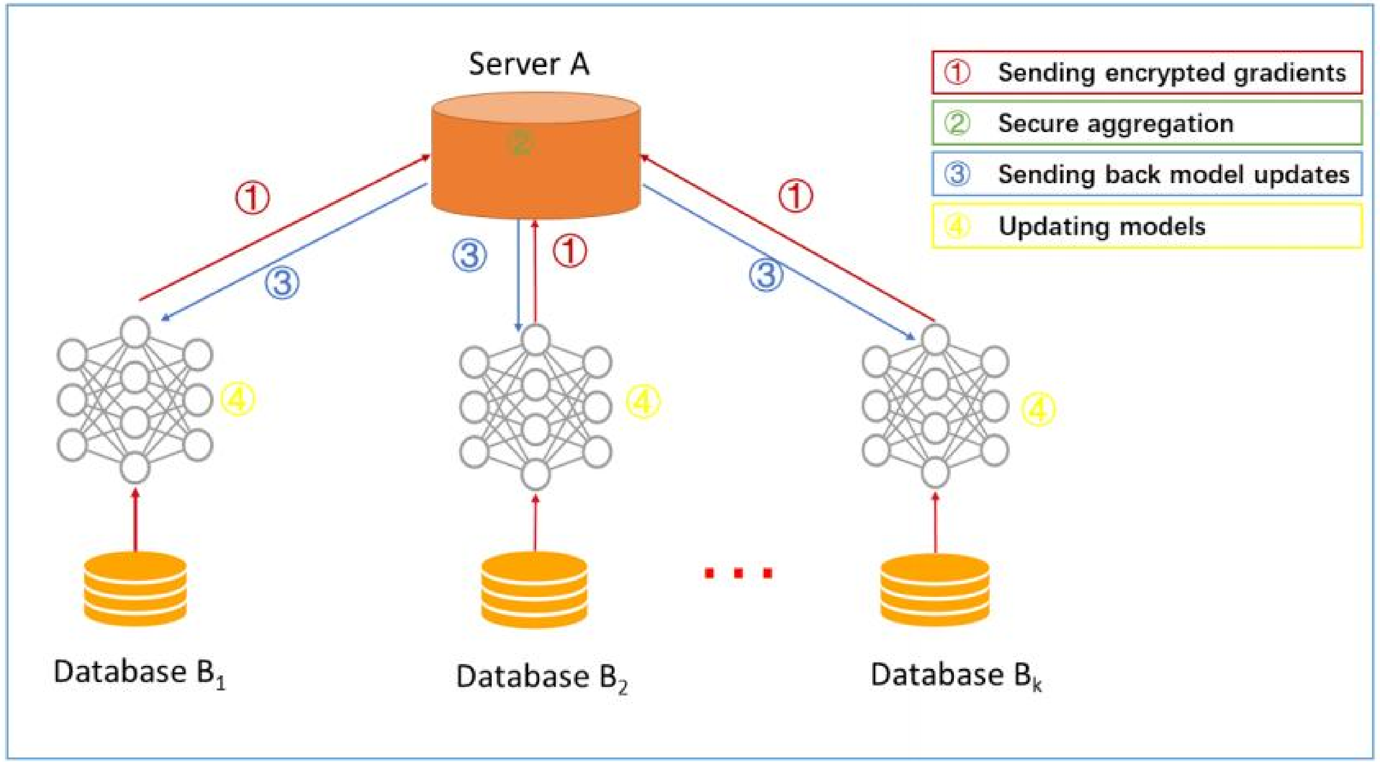


Fig. 3. Architecture for a horizontal federated learning system

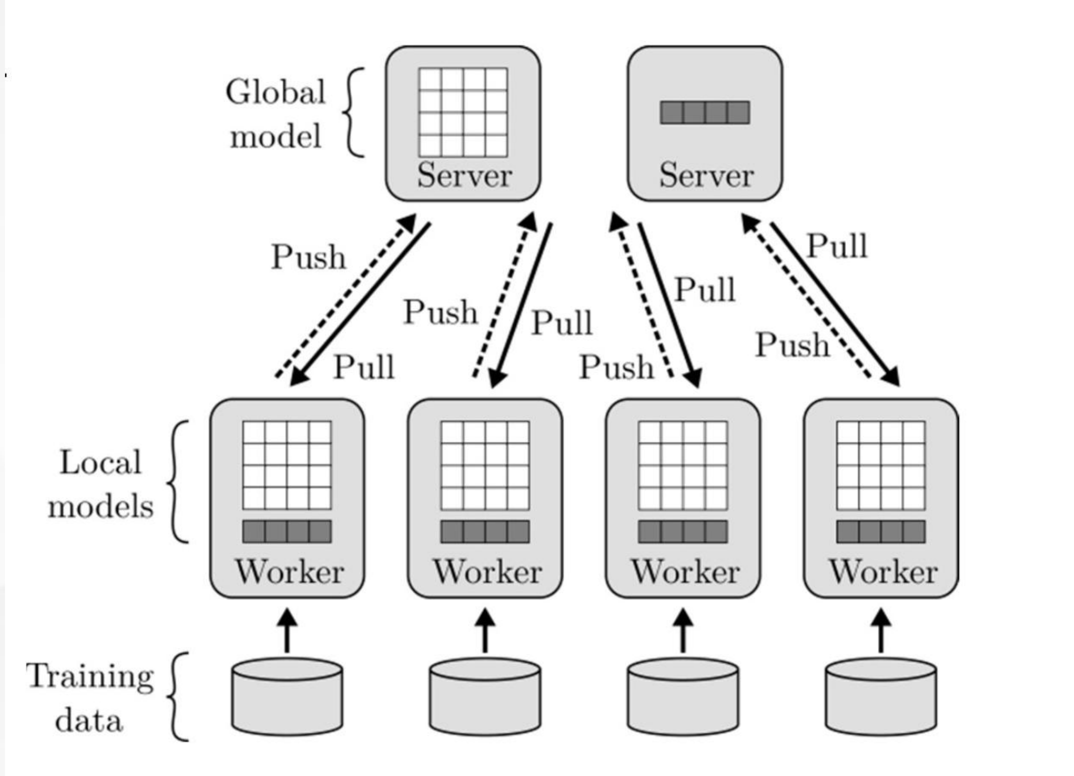
FedAvg algorithm

- Contributions
  - Consider the problem of training dispersed data from mobile devices as an important research direction.
  - Propose a simple and practical algorithm for Federated Averaging.
  - An extensive empirical evaluation of the proposed algorithms shows that they are robust to non-independently identically distributed (Non-IID) and unbalanced data.

FedAvg algorithm

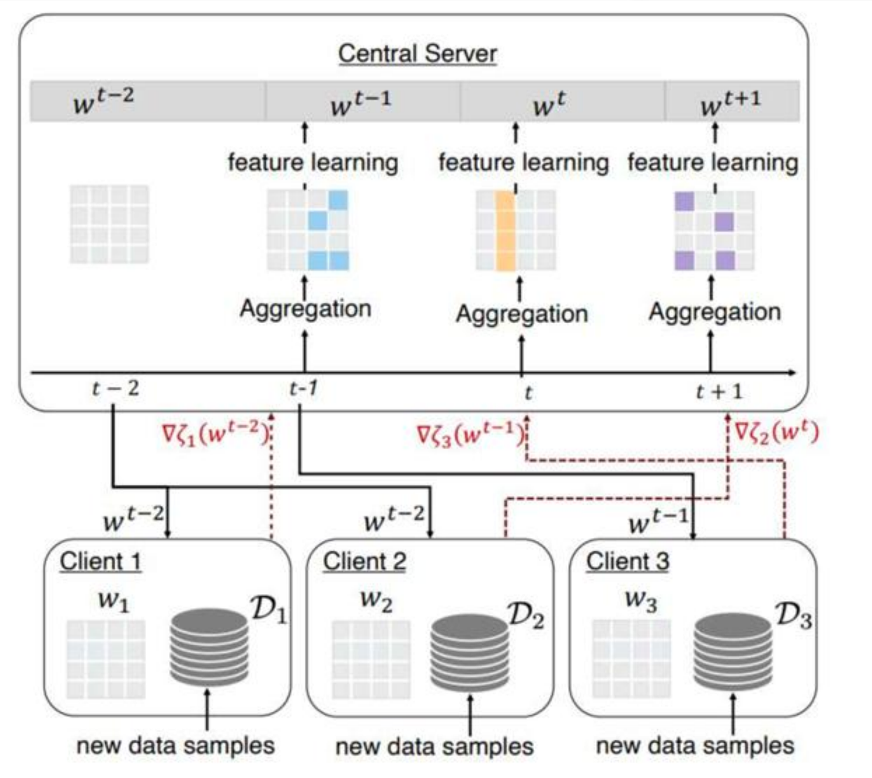
Federated learning review

- Learning tasks are handled by a loose federation of participating devices (clients) coordinated by a central server.



FedAvg algorithm

- Federated learning review
  - Each client has a local training data set that does not need to be uploaded to the server and only sends local model parameters for each update.



FedAvg algorithm

- Federated learning (FL) review: ideal FL
  - Training on **real-world data** from mobile devices has distinct advantages over the proxy data that is ubiquitous in data centers.
  - This data is **privacy-sensitive** or large (compared to the size of the model) to avoid recording it to the data center for model training.
  - For supervised tasks, labels on data can be **naturally inferred** from user interactions.

FedAvg algorithm

- Federated learning (FL) review: privacy
  - In the traditional distributed training setting, even if an "anonymous" data set is held, users' privacy will be threatened through the **connection with other data**.
  - In contrast, the information transmitted in FL is the **minimum update** (all/part of the model parameters) needed to improve a particular model, and less information means a lower risk of privacy disclosure.
  - Combining FL with **secure multi-party computing and differential privacy**.

FedAvg algorithm

- Federated settings
  - Non-IID data: Training data on a given client is usually based on mobile device usage by a particular user, so any local data set for a particular user **does not represent a group distribution**.
  - Imbalance: Some users will use the service or application more than others, resulting in **different amounts of local training data**.
  - Massive clients: We expect the number of clients participating in the FL to be **much larger than** the average number of instances per client.
  - Limited communication: Mobile devices are often **offline or in a slow or expensive connection**.

FedAvg algorithm

Objective

$$\min_{w \in \mathbb{R}^d} f(w) \qquad \text{where} \qquad f(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} f_i(w). \qquad (1)$$

$$\Downarrow$$

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w).$$

$$f_i(w) = l(x_i, y_i; w)$$

K number of clients
P_k the distribution of local data
n_k the number of local data samples

FedAvg algorithm

- Limited communication
  - In data center optimization, communication costs are relatively small while computing costs dominate, and recent emphasis has been on using GPUs to reduce these costs. In contrast, in joint optimization, **communication costs dominate** -- we're typically limited by upload bandwidth of 1MB/s or less.
  - Clients typically volunteer for optimization only when charging, plugging in, and using a non-billable Wi-Fi connection, and we expect each client to **participate in a small number of update sessions** per day.
  - Modern smartphones have **plenty of local computing power and small datasets** on a single device.

FedAvg algorithm

- Reduce communication
  - Increasing **parallelism**, that is, using more clients to work independently between rounds of communication.
  - Add computations per client, that is, **perform more complex computations** (such as cumulative training) between rounds of communication.

FedAvg algorithm

- Federated Averaging

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

K： Client amount

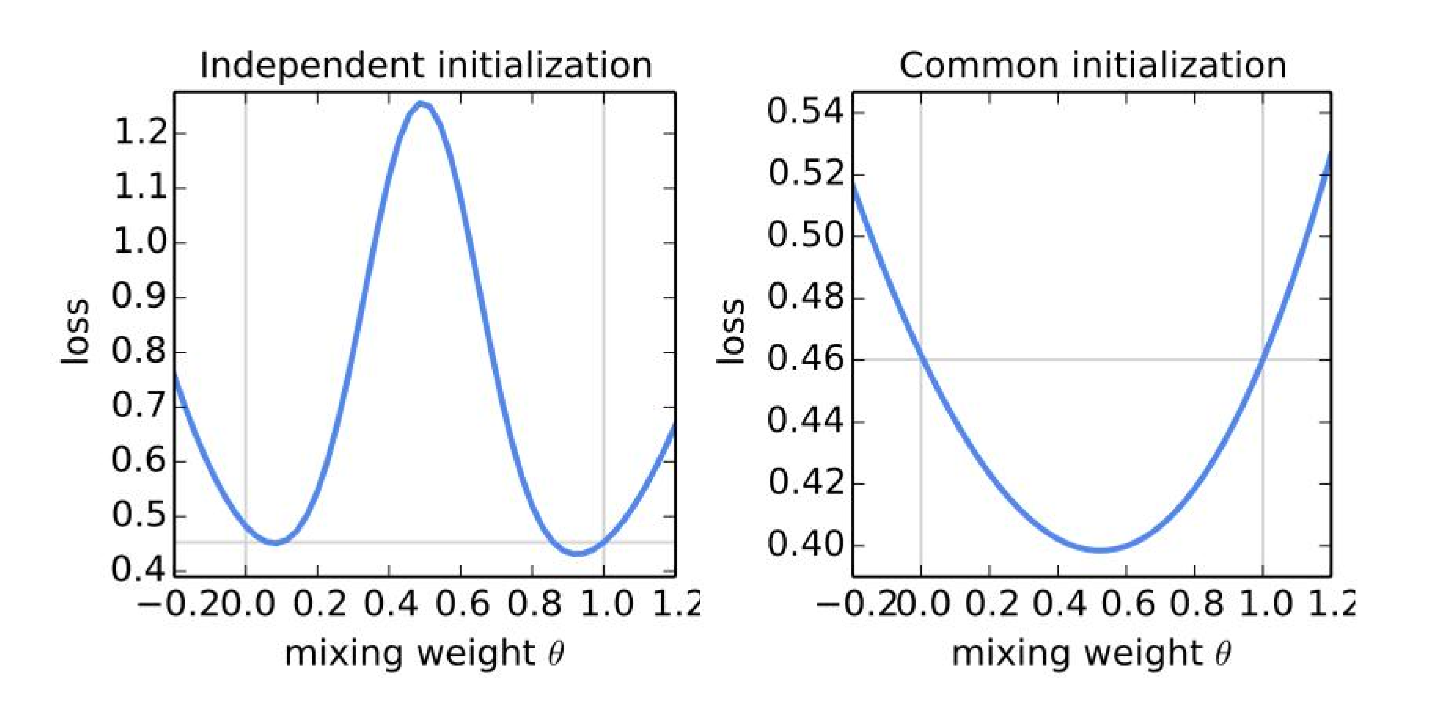
B： Batch size

E： Local training rounds

η： Local learning rate

FedAvg algorithm

- Federated Averaging
  - It is beneficial to initialize local models with the common start point
  - Mixed weight = θw + (1 - θ)w′



FedAvg algorithm

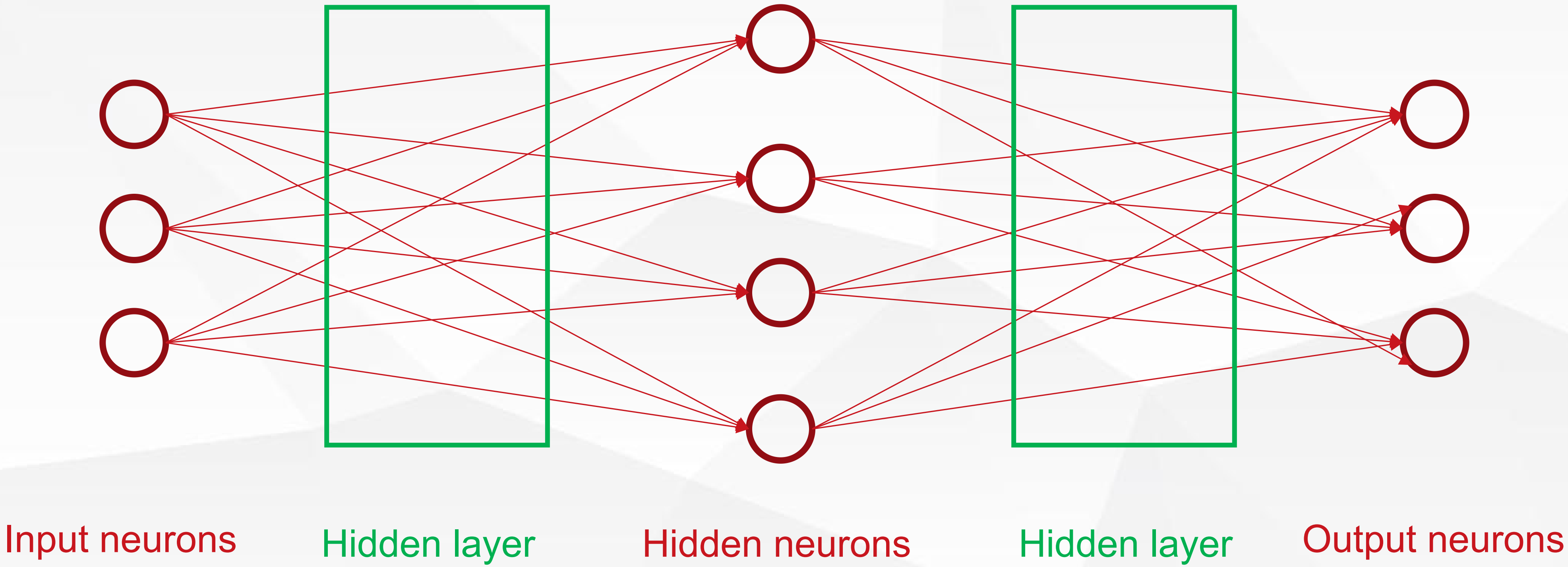
- Experiments
  - Image classification: MNIST handwritten digital recognition
  - Language modeling: Dataset based on the Complete Works of William Shakespeare

FedAvg algorithm

- Experiments (MNIST)
  - IID: shuffling the data and dividing it into 100 clients, each receiving 600 examples.
  - Non-IID: (1) sort the data by number label; (2) divide it into 200 shards of size 300; (3) assign 2 shards to each of the 100 clients, most clients will have only two number examples.

FedAvg algorithm

- Experiments (MNIST)
  - 1) A simple multilayer-perceptron with 2-hidden layers with 200 units each using ReLU activations (199,210 total parameters), which we refer to as the MNIST 2NN.



FedAvg algorithm

- Experiments (MNIST)
  - 1) A simple multilayer-perceptron with 2-hidden layers with 200 units each using ReLU activations (199,210 total parameters), which we refer to as the MNIST 2NN.

```
class FedAvgMLP(nn.Module):
    def __init__(self, in_features=784, num_classes=10, hidden_dim=200):
        super().__init__()
        self.fc1 = nn.Linear(in_features, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, num_classes)
        self.act = nn.ReLU(inplace=True)

    def forward(self, x):
        if x.ndim == 4:
            x = x.view(x.size(0), -1)
        x = self.act(self.fc1(x))
        x = self.fc2(x)
        return x
```

FedAvg algorithm

- Experiments (MNIST)
  - 2) A CNN with two 5x5 convolution layers (the first with 32 channels, the second with 64, each followed with 2x2 max pooling), a fully connected layer with 512 units and ReLU activation, and a final softmax output layer (1,663,370 total parameters).

```
class FedAvgCNN(nn.Module):
    def __init__(self, in_features=1, num_classes=10, dim=1024):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_features,
                        32,
                        kernel_size=5,
                        padding=0,
                        stride=1,
                        bias=True),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=(2, 2))
        )
```

FedAvg algorithm

- Experiments (Key hyperparameters)
  - C： Proportion of clients participating in calculation: 1 indicates that all clients participate in training
  - E： Number of training cycles per client between two communications
  - B： Mini-batch size of each client. ∞ indicates full-batch
  - $u_k = E\frac{n_k}{B}$: The total number of updates in each iteration on client k

FedAvg algorithm

- Experiments (communication rounds)

2NN	IID		NON-IID	
C	$B=\infty$	$B=10$	$B=\infty$	$B=10$
0.0	1455	316	4278	3275
0.1	1474 (1.0×)	87 (3.6×)	1796 (2.4×)	664 (4.9×)
0.2	1658 (0.9×)	77 (4.1×)	1528 (2.8×)	619 (5.3×)
0.5	— (—)	75 (4.2×)	— (—)	443 (7.4×)
1.0	— (—)	70 (4.5×)	— (—)	380 (8.6×)
CNN, $E=5$				
0.0	387	50	1181	956
0.1	339 (1.1×)	18 (2.8×)	1100 (1.1×)	206 (4.6×)
0.2	337 (1.1×)	18 (2.8×)	978 (1.2×)	200 (4.8×)
0.5	164 (2.4×)	18 (2.8×)	1067 (1.1×)	261 (3.7×)
1.0	246 (1.6×)	16 (3.1×)	— (—)	97 (9.9×)

C = 0.1 is the best

C： Proportion of clients participating in calculation
E： Number of training cycles per client between two communications
B： Mini-batch size of each client. ∞ indicates full-batch
$u_k = E\frac{n_k}{B}$**： The total number of updates in each iteration on client k**

FedAvg algorithm

- Experiments (communication rounds)

MNIST CNN, 99% ACCURACY					
CNN	E	B	u	IID	NON-IID
FEDSGD	1	∞	1	626	483
FEDAVG	5	∞	5	179 (3.5×)	1000 (0.5×)
FEDAVG	1	50	12	65 (9.6×)	600 (0.8×)
FEDAVG	20	∞	20	234 (2.7×)	672 (0.7×)
FEDAVG	1	10	60	34 (18.4×)	350 (1.4×)
FEDAVG	5	50	60	29 (21.6×)	334 (1.4×)
FEDAVG	20	50	240	32 (19.6×)	426 (1.1×)
FEDAVG	5	10	300	20 (31.3×)	229 (2.1×)
FEDAVG	20	10	1200	18 (34.8×)	173 (2.8×)
SHAKESPEARE LSTM, 54% ACCURACY					
LSTM	E	B	u	IID	NON-IID
FEDSGD	1	∞	1.0	2488	3906
FEDAVG	1	50	1.5	1635 (1.5×)	549 (7.1×)
FEDAVG	5	∞	5.0	613 (4.1×)	597 (6.5×)
FEDAVG	1	10	7.4	460 (5.4×)	164 (23.8×)
FEDAVG	5	50	7.4	401 (6.2×)	152 (25.7×)
FEDAVG	5	10	37.1	192 (13.0×)	41 (95.3×)

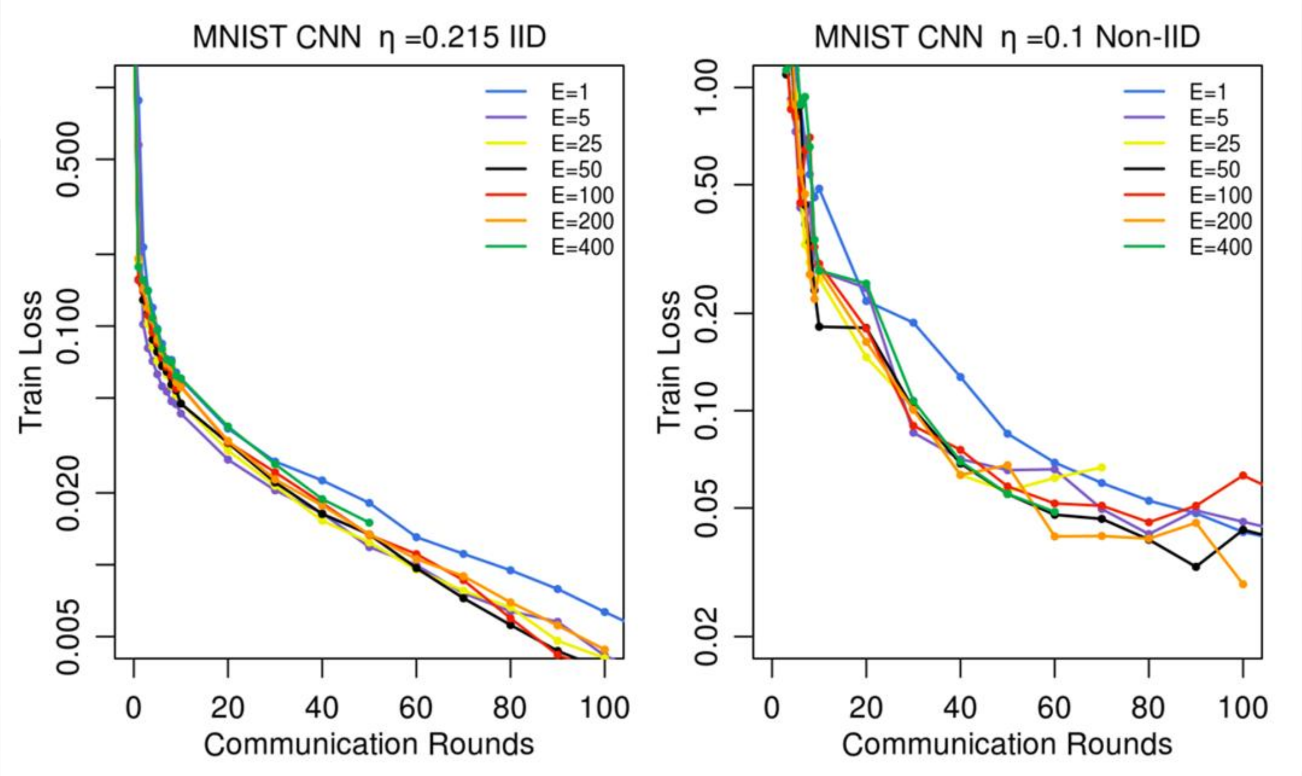
FedSGD: C = 1 and Full-Batch Optimization

FedAvg: C = 0.1

C： Proportion of clients participating in calculation
E： Number of training cycles per client between two communications
B： Mini-batch size of each client. ∞ indicates full-batch
$u_k = E\frac{n_k}{B}$**： The total number of updates in each iteration on client** k

FedAvg algorithm

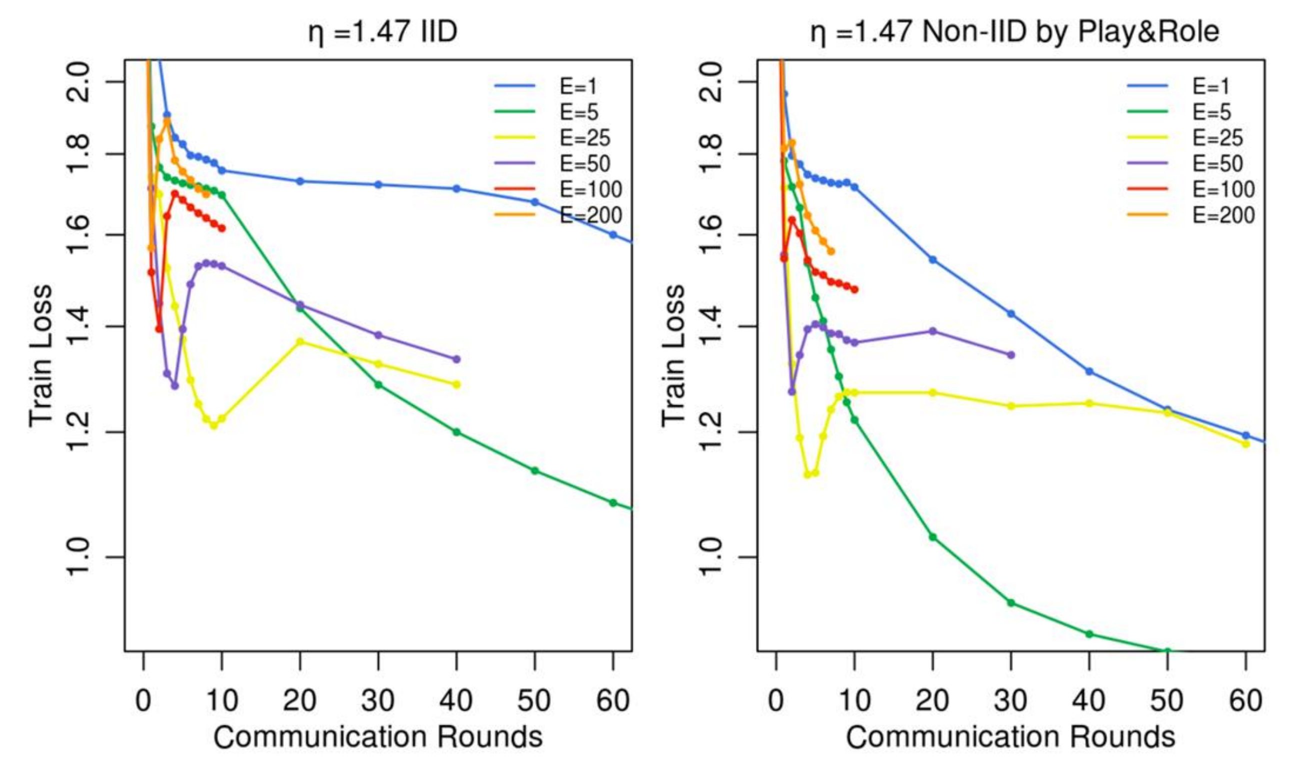
- Experiments (training loss curves)



C： Proportion of clients participating in calculation, C=0.1
E： Number of training cycles per client between two communications
B： Mini-batch size of each client. B=10.
$u_k = E\frac{n_k}{B}$**： The total number of updates in each iteration on client k**

FedAvg algorithm

- Experiments (training loss curves)



LSTM

C: Proportion of clients participating in calculation, C=0.1
E: Number of training cycles per client between two communications
B: Mini-batch size of each client. B=10.
$\boldsymbol{u}_k = E\frac{n_k}{B}$: **The total number of updates in each iteration on client $\boldsymbol{k}$**

02

MOON algorithm

- Observation
- Contrastive learning
- MOON

MOON algorithm

Observation

- The global model trained on a whole dataset can learn a better representation than the local model trained on a skewed subset.

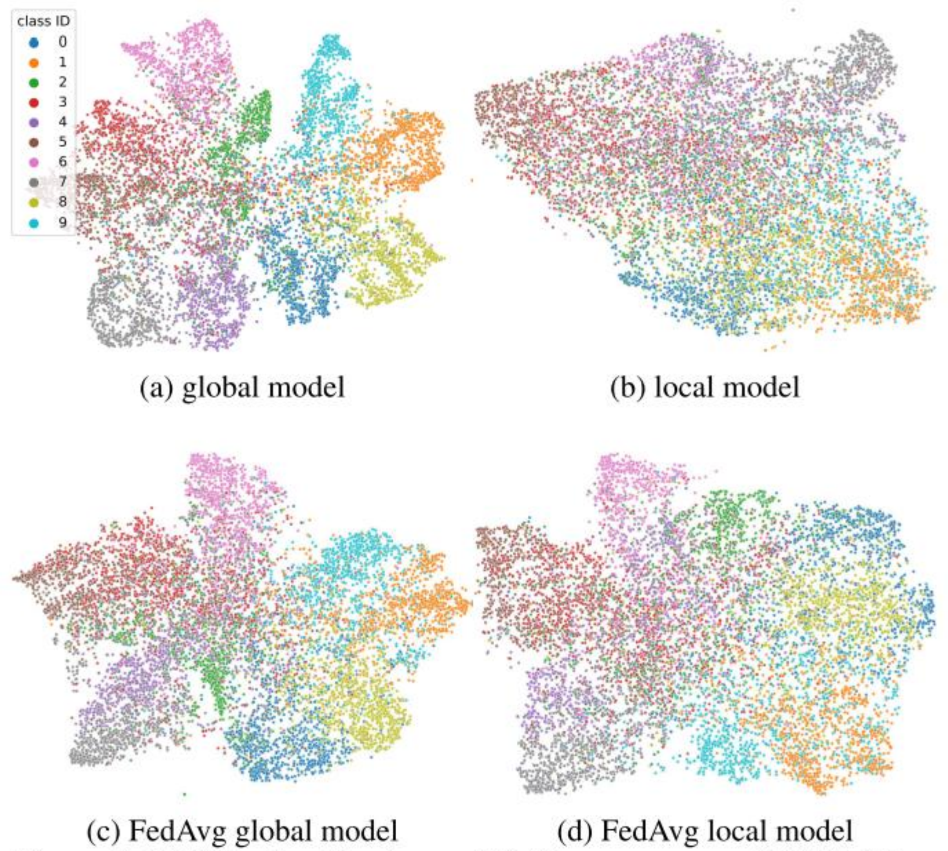


Figure 2. T-SNE visualizations of hidden vectors on CIFAR-10.

MOON algorithm

- Observation
  - The global model trained on a whole dataset can learn a better representation than the local model trained on a skewed subset.
  - Propose model-contrastive learning (MOON), which corrects the local updates by maximizing the agreement of representation learned by the current local model and the representation learned by the global model.

MOON algorithm

- Contrastive learning
  - The key idea of contrastive learning is to reduce the distance between the representations of different augmented views of the same image (i.e., positive pairs), and increase the distance between the representations of augmented views of different images (i.e., negative pairs)

$$l_{i,j} = -\log \frac{\exp(\mathrm{sim}(x_i, x_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{I}_{[k \neq i]} \exp(\mathrm{sim}(x_i, x_k)/\tau)}$$

MOON algorithm

- MOON
  - Global representation $z_{glob} = R_{w^t}(x)$
  - Local representation w_i^{t-1} (i.e., $z_{prev} = R_{w_i^{t-1}}(x)$)
  - Current representation $z = R_{w_i^t}(x)$
  -

$$\ell_{con} = -\log \frac{\exp(\text{sim}(z, z_{glob})/\tau)}{\exp(\text{sim}(z, z_{glob})/\tau) + \exp(\text{sim}(z, z_{prev})/\tau)}$$

$$\ell = \ell_{sup}(w_i^t; (x, y)) + \mu \ell_{con}(w_i^t; w_i^{t-1}; w^t; x),$$

$$\min_{w_i^t} \mathbb{E}_{(x,y)\sim D^i} [\ell_{sup}(w_i^t; (x, y)) + \mu \ell_{con}(w_i^t; w_i^{t-1}; w^t; x)].$$

MOON algorithm

MOON

Algorithm 1: The MOON framework

Input: number of communication rounds T, number of parties N, number of local epochs E, temperature τ, learning rate η, hyper-parameter μ

Output: The final model w^T

Server executes:

- initialize w^0
- **for** $t = 0, 1, ..., T-1$ **do**
  - **for** $i = 1, 2, ..., N$ **in parallel do**
    - send the global model w^t to P_i
    - $w_i^t \leftarrow$ **PartyLocalTraining**(i, w^t)
  - $w^{t+1} \leftarrow \sum_{k=1}^{N} \frac{|\mathcal{D}^i|}{|\mathcal{D}|} w_k^t$
- return w^T

PartyLocalTraining(i, w^t):

- $w_i^t \leftarrow w^t$
- **for** epoch $i = 1, 2, ..., E$ **do**
  - **for** each batch $\mathbf{b} = \{x, y\}$ of $\mathcal{D}^i$ **do**
    - $\ell_{sup} \leftarrow CrossEntropyLoss(F_{w_i^t}(x), y)$
    - $z \leftarrow R_{w_i^t}(x)$
    - $z_{glob} \leftarrow R_{w^t}(x)$
    - $z_{prev} \leftarrow R_{w_i^{t-1}}(x)$
    - $\ell_{con} \leftarrow$
      $-\log \frac{\exp(\mathrm{sim}(z, z_{glob})/\tau)}{\exp(\mathrm{sim}(z, z_{glob})/\tau) + \exp(\mathrm{sim}(z, z_{prev})/\tau)}$
    - $\ell \leftarrow \ell_{sup} + \mu \ell_{con}$
    - $w_i^t \leftarrow w_i^t - \eta \nabla \ell$
- return w_i^t to server

03

FedDyn algorithm

- Intuition
- FedDyn
- Analysis

FedDyn algorithm

- Intuition
  - Training models on local data that minimize local empirical loss appears to be meaningful, but yet, doing so is fundamentally inconsistent with minimizing the global empirical loss.
  - Dynamically modify the device objective with a penalty term so that, in the limit, when model parameters converge, they do so to stationary points of the global empirical loss.

FedDyn algorithm

- FedDyn

Algorithm 1: Federated Dynamic Regularizer - (FedDyn)

Input: $T, \boldsymbol{\theta}^0, \alpha > 0, \nabla L_k(\boldsymbol{\theta}_k^0) = \mathbf{0}$.

for $t = 1, 2, \ldots T$ **do**

  Sample devices $\mathcal{P}_t \subseteq [m]$ and transmit $\boldsymbol{\theta}^{t-1}$ to each selected device,

  for *each device* $k \in \mathcal{P}_t$, *and in parallel* **do**

    Set $\boldsymbol{\theta}_k^t = \boxed{\underset{\boldsymbol{\theta}}{\arg\min}}\, L_k(\boldsymbol{\theta}) - \langle \nabla L_k(\boldsymbol{\theta}_k^{t-1}), \boldsymbol{\theta} \rangle + \frac{\alpha}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}^{t-1}\|^2$,

    $\boxed{\text{Set } \nabla L_k(\boldsymbol{\theta}_k^t) = \nabla L_k(\boldsymbol{\theta}_k^{t-1}) - \alpha \left(\boldsymbol{\theta}_k^t - \boldsymbol{\theta}^{t-1}\right),}$

    Transmit device model $\boldsymbol{\theta}_k^t$ to server,

  end for

  for *each device* $k \notin \mathcal{P}_t$, *and in parallel* **do**

    Set $\boldsymbol{\theta}_k^t = \boldsymbol{\theta}_k^{t-1}, \nabla L_k(\boldsymbol{\theta}_k^t) = \nabla L_k(\boldsymbol{\theta}_k^{t-1})$,

  end for

  Set $\boldsymbol{h}^t = \boldsymbol{h}^{t-1} - \alpha \frac{1}{m} \left(\sum_{k \in \mathcal{P}_t} \boldsymbol{\theta}_k^t - \boldsymbol{\theta}^{t-1}\right)$,

  Set $\boldsymbol{\theta}^t = \left(\frac{1}{|\mathcal{P}_t|} \sum_{k \in \mathcal{P}_t} \boldsymbol{\theta}_k^t\right) - \frac{1}{\alpha} \boldsymbol{h}^t$

end for

FedDyn algorithm

Analysis

$$\boldsymbol{\theta}_k^t = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[\mathfrak{R}_k(\boldsymbol{\theta}; \boldsymbol{\theta}_k^{t-1}, \boldsymbol{\theta}^{t-1}) \triangleq L_k(\boldsymbol{\theta}) \boxed{- \langle \nabla L_k(\boldsymbol{\theta}_k^{t-1}), \boldsymbol{\theta} \rangle} + \frac{\alpha}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}^{t-1}\|^2 \right]$$

- The first order condition

$$\nabla L_k(\boldsymbol{\theta}_k^t) - \nabla L_k(\boldsymbol{\theta}_k^{t-1}) + \alpha(\boldsymbol{\theta}_k^t - \boldsymbol{\theta}^{t-1}) = \mathbf{0}$$

- If local device models converge, they converge to the server model, and the convergence point is a stationary point of the global loss.

if $\boldsymbol{\theta}_k^t \rightarrow \boldsymbol{\theta}_k^\infty$, it generally follows that, $\nabla L_k(\boldsymbol{\theta}_k^t) \rightarrow \nabla L_k(\boldsymbol{\theta}_k^\infty)$, and as a consequence, we have $\boldsymbol{\theta}^t \rightarrow \boldsymbol{\theta}_k^\infty$. In turn this implies that $\boldsymbol{\theta}_k^\infty \rightarrow \boldsymbol{\theta}^\infty$, i.e., is independent of k.

04

KT-pFL algorithm

- Intuition
- Knowledge Distillation (KD)
- KT-pFL

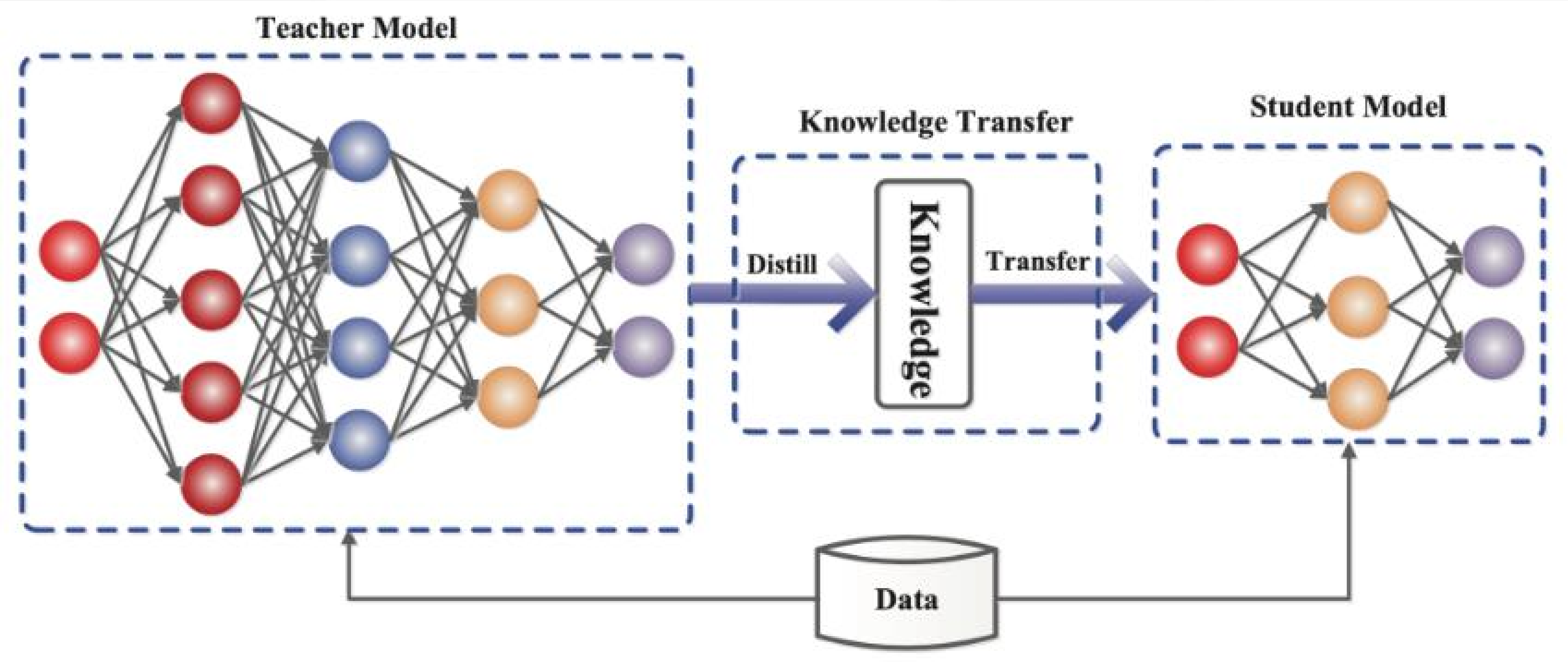
KT-pFL algorithm

- Intuition
  - Main idea is to allow each client to maintain a personalized soft prediction at the server that can be updated by a linear combination of all clients local soft predictions using a knowledge coefficient matrix.
  - Regardless of model structures

KT-pFL algorithm

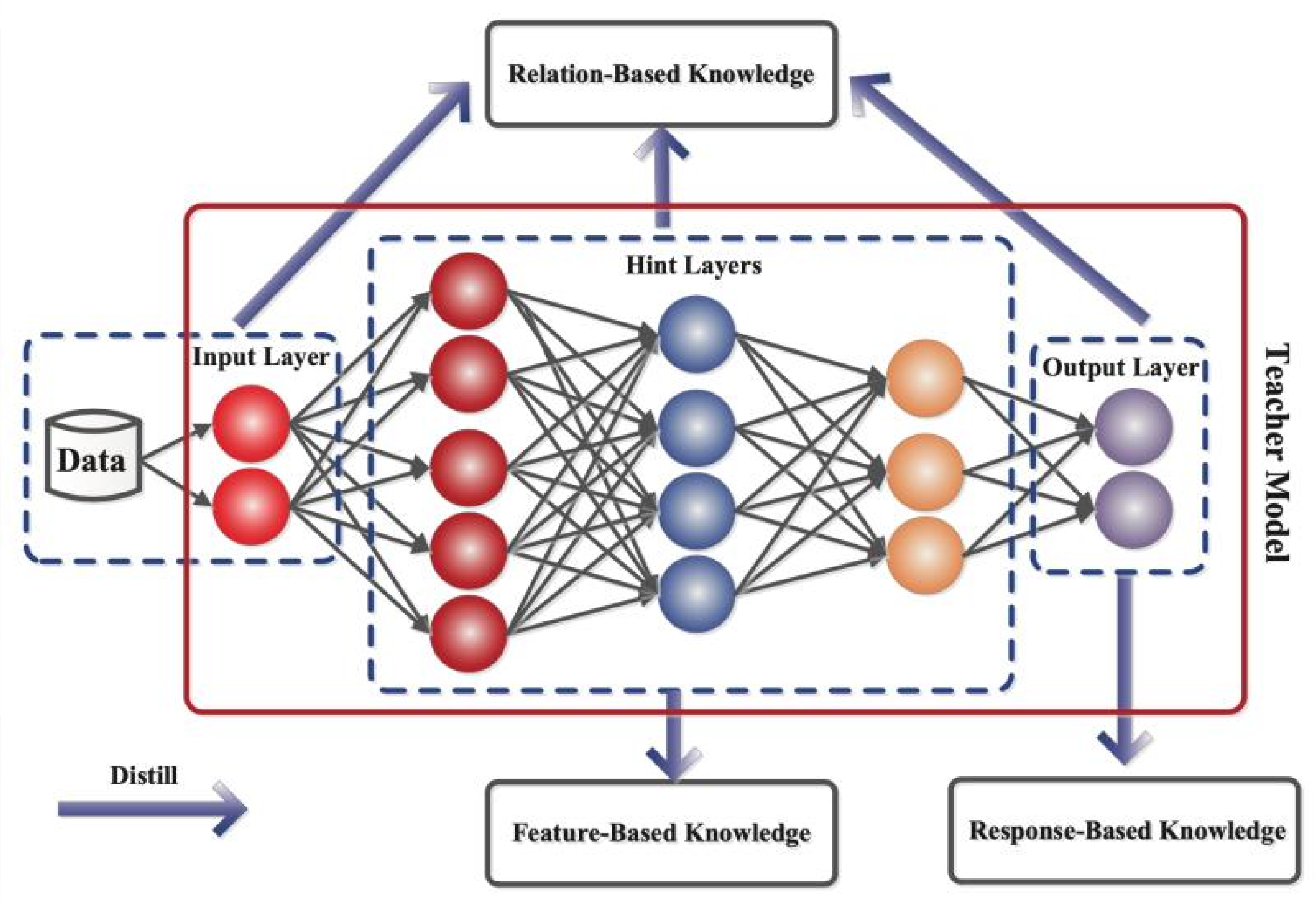
Knowledge Distillation (KD)

- Transfer knowledge from well-learned teacher model to student model



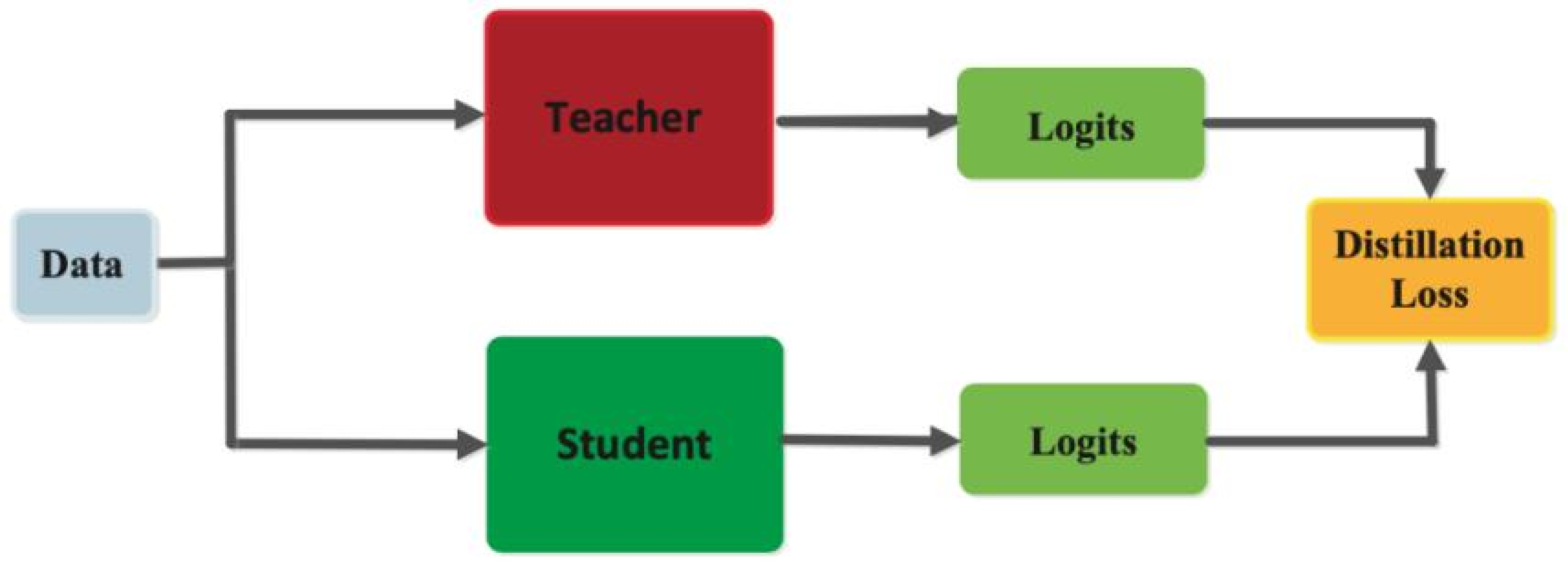
KT-pFL algorithm

- Knowledge Distillation (KD)
  - Classification



KT-pFL algorithm

- Knowledge Distillation (KD)
  - Response-based KD



KT-pFL algorithm

- KT-pFL
  - Objective

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) := \sum_{n=1}^{N} \frac{D_n}{D} \mathcal{L}_n(\mathbf{w}), \text{ where } \mathcal{L}_n(\mathbf{w}) = \frac{1}{D_n} \sum_{i=1}^{D_n} \mathcal{L}_{CE}(\mathbf{w}; x_i, y_i).$$

$$\min_{\mathbf{w}^1,\cdots,\mathbf{w}^N} \mathcal{L}(\mathbf{w}^1, \cdots, \mathbf{w}^N) := \sum_{n=1}^{N} \frac{D_n}{D} L_n(\mathbf{w}^n)$$

KT-pFL algorithm

- KT-pFL
  - Personalized loss function
    - Kullback–Leibler (KL) Divergence
    - c_{mn} is the knowledge coefficient which is used to estimate the contribution from client m to n.
    - $s(w^n, \hat{x})$ can be deemed to be a soft prediction of the client n

$$\mathcal{L}_{per,n}(\mathbf{w}^n) := \mathcal{L}_n(\mathbf{w}^n) + \lambda \sum_{\hat{x}\in\mathbb{D}_r} \mathcal{L}_{KL}\left(\sum_{m=1}^{N} c_{mn} \cdot s(\mathbf{w}^m, \hat{x}), s(\mathbf{w}^n, \hat{x})\right)$$

$$s(\mathbf{w}^n, \hat{x}) = \frac{\exp(z_c^n/T)}{\sum_{c=1}^{C} \exp(z_c^n/T)},$$

KT-pFL algorithm

- KT-pFL
  - Knowledge coefficient matrix

$$\mathbf{c} = \begin{Bmatrix} c_{11} & c_{12} & \cdots & c_{1N} \\ c_{21} & c_{22} & \cdots & c_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ c_{N1} & c_{N2} & \cdots & c_{NN} \end{Bmatrix}.$$

KT-pFL algorithm

- KT-pFL
  - Objective

$$\min_{\mathbf{w},\mathbf{c}} \mathcal{L}(\mathbf{w},\mathbf{c}) := \sum_{n=1}^{N} \frac{D_n}{D} \mathcal{L}_{per,n}(\mathbf{w}^n) + \rho \|\mathbf{c} - \frac{\mathbf{1}}{N}\|^2,$$

$$\mathbf{w} = [\mathbf{w}^1, \cdots, \mathbf{w}^N] \in \mathbb{R}^{\sum_{n=1}^{N} d_n}$$

KT-pFL algorithm

- KT-pFL
  - Training
    - Update w
      - Local Training
      - Distillation
    - Update c

$$\mathbf{w}^n \leftarrow \mathbf{w}^n - \eta_1 \nabla_{\mathbf{w}^n} \mathcal{L}_n(\mathbf{w}^n; \xi_n),$$

$$\mathbf{w}^n \leftarrow \mathbf{w}^n - \eta_2 \nabla_{\mathbf{w}^n} \mathcal{L}_{KL}\left(\sum_{m=1}^{N} \mathbf{c}_m^{*,T} \cdot s(\mathbf{w}^m, \xi_r), s(\mathbf{w}^n, \xi_r)\right)$$

$$\mathbf{c} \leftarrow \mathbf{c} - \eta_3 \lambda \sum_{n=1}^{N} \frac{D_n}{D} \nabla_{\mathbf{c}} \mathcal{L}_{KL}\left(\sum_{m=1}^{N} \mathbf{c}_m \cdot s(\mathbf{w}^{m,*}, \xi_r), s(\mathbf{w}^{n,*}, \xi_r)\right) - 2\eta_3 \rho(\mathbf{c} - \frac{\mathbf{1}}{N})$$

KT-pFL algorithm

- KT-pFL

Algorithm 1 KT-pFL Algorithm

Input: $\mathbb{D}$, $\mathbb{D}_r$, η_1, η_2, η_3 and T

Output: $\mathbf{w} = [\mathbf{w}^1, \cdots, \mathbf{w}^N]$

Initialize $\mathbf{w}_0$ and $\mathbf{c}_0$
procedure SERVER-SIDE OPTIMIZATION
    Distribute $\mathbf{w}_0$ and $\mathbf{c}_0$ to each client
    for each communication round $t \in \{1, 2, ..., T\}$ **do**
        for each client n **in parallel do**
            $\mathbf{w}^n_{t+1} \leftarrow ClientLocalUpdate(n, \mathbf{w}^n_t, \mathbf{c}_{t,n})$
        Update knowledge coefficient matrix $\mathbf{c}$ via (7)
        Distribute $\mathbf{c}_{t+1}$ to all clients
procedure CLIENTLOCALUPDATE$(n, \mathbf{w}^n_t, \mathbf{c}_{t,n})$
    Client n receives $\mathbf{w}^n_t$ and $\mathbf{c}_n$ from the server
    for each local epoch i from 1 to E **do**
        for mini-batch $\xi_t \subseteq \mathbb{D}_n$ **do**
            Local Training: update model parameters on private data via (5)
    for each distillation step j from 1 to R **do**
        for mini-batch $\xi_{r,t} \subseteq \mathbb{D}_r$ **do**
            Distillation: update model parameters on public data via (6)
    return local parameters $\mathbf{w}^n_{t+1}$

KT-pFL algorithm

- KT-pFL
  - Illustration

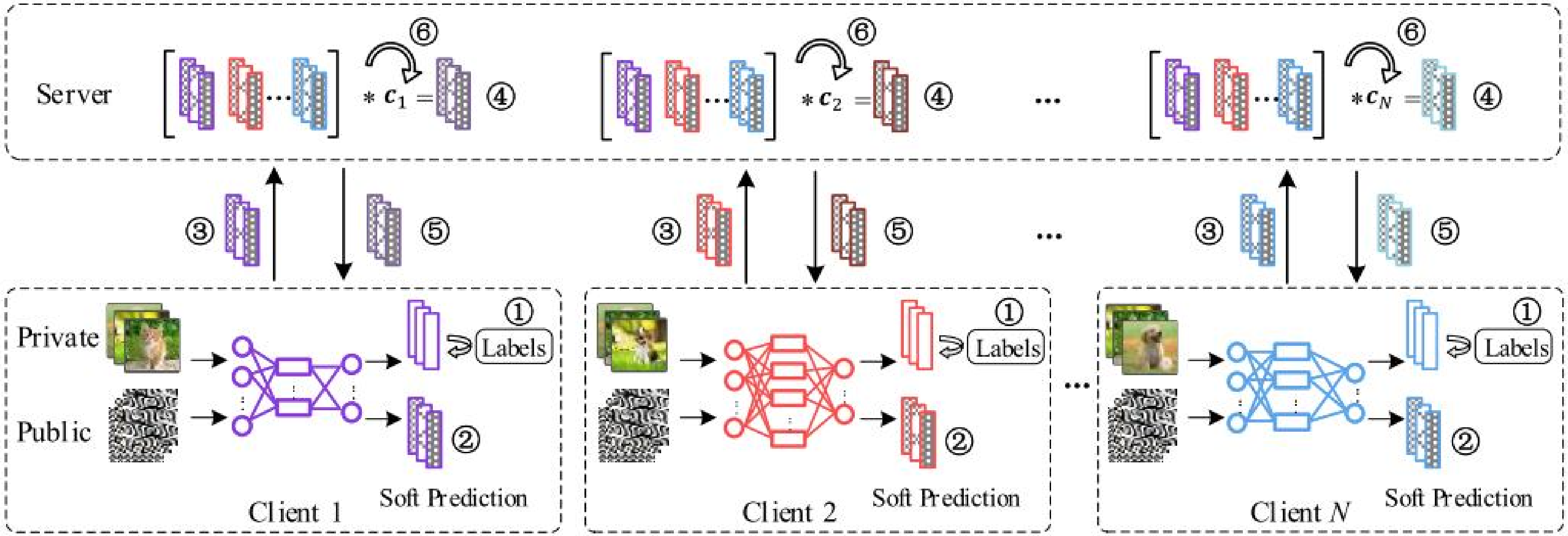


Figure 1: Illustration of the KT-pFL framework. The workflow includes 6 steps: ① local training on private data; ②, ③ each client outputs the local soft prediction on public data and sends it to the server; ④ the server calculates each client's personalized soft prediction via a linear combination of local soft predictions and knowledge coefficient matrix; ⑤ each client downloads the personalized soft prediction to perform distillation phase; ⑥ the server updates the knowledge coefficient matrix.

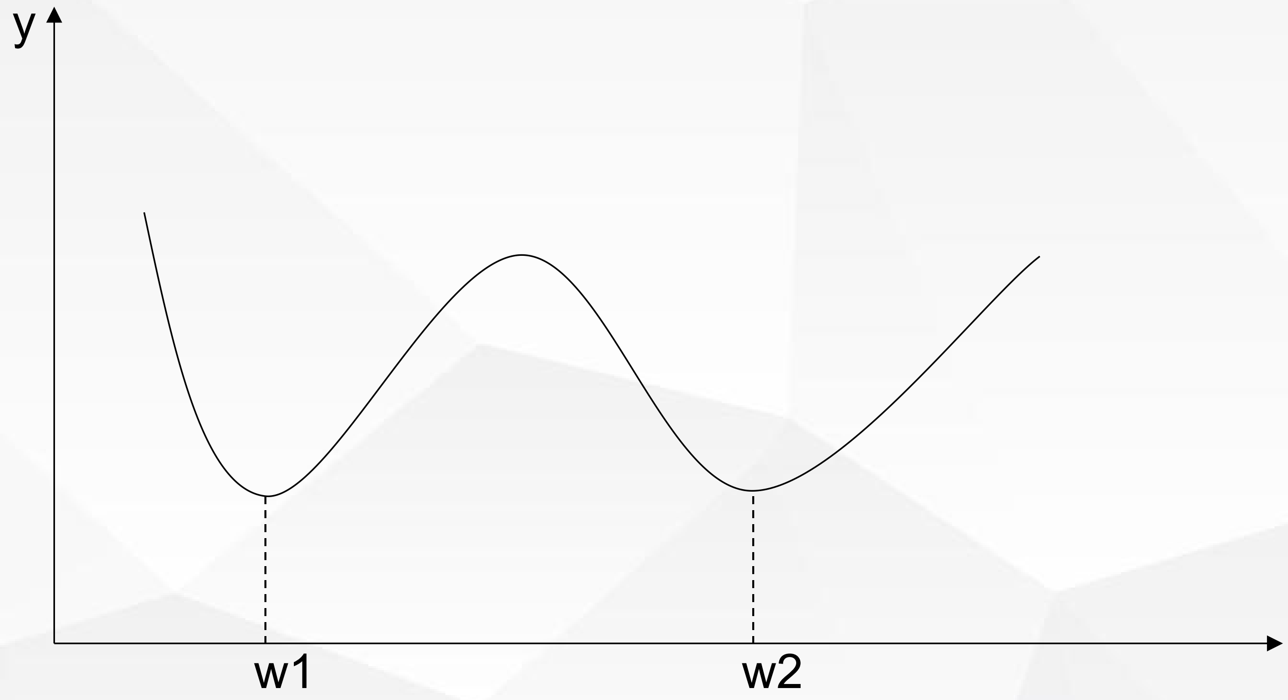
05

FedMA algorithm

- Permutation invariance
- Matched averaging formulation
- FedMA

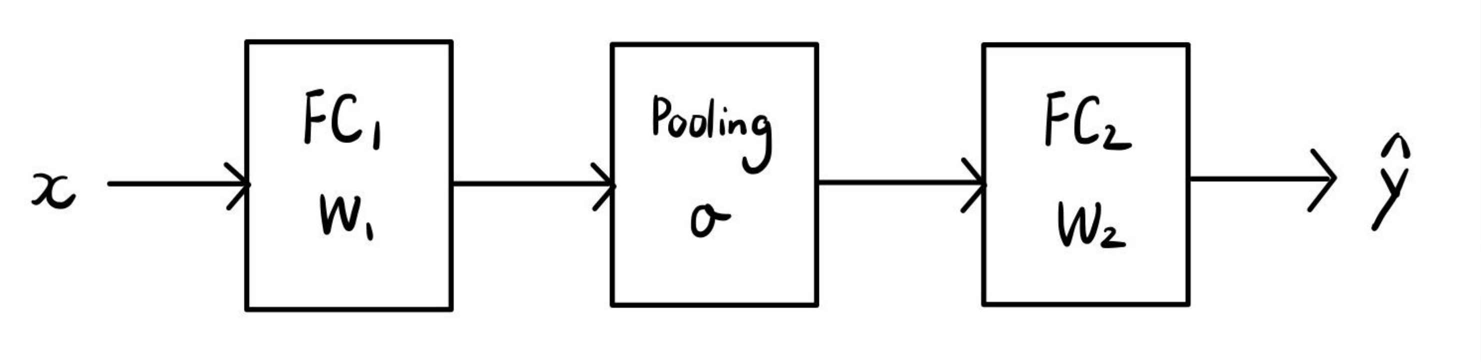
FedMA algorithm

- Permutation invariance



FedMA algorithm

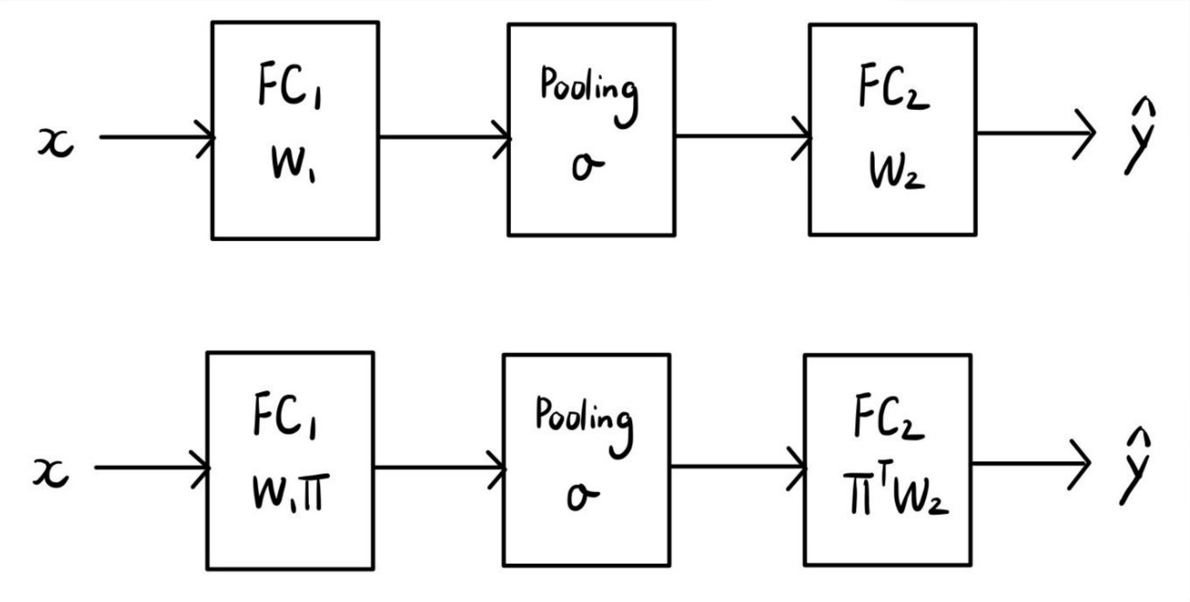
- Permutation invariance (fully-connected (FC) layer)



$$\hat{y} = \sigma(xW_1)W_2$$

FedMA algorithm

- Permutation invariance (fully-connected (FC) layer)



$$\hat{y} = \sigma(xW_1\Pi)\Pi^T W_2, \text{ where } \Pi \text{ is any } L \times L \text{ permutation matrix.}$$

FedMA algorithm

- Permutation invariance (fully-connected (FC) layer)

Client A:

$\{W_1\Pi_j, \Pi_j^T W_2\}$

Client B:

$\{W_1\Pi_{j'}, \Pi_{j'}^T W_2\}$

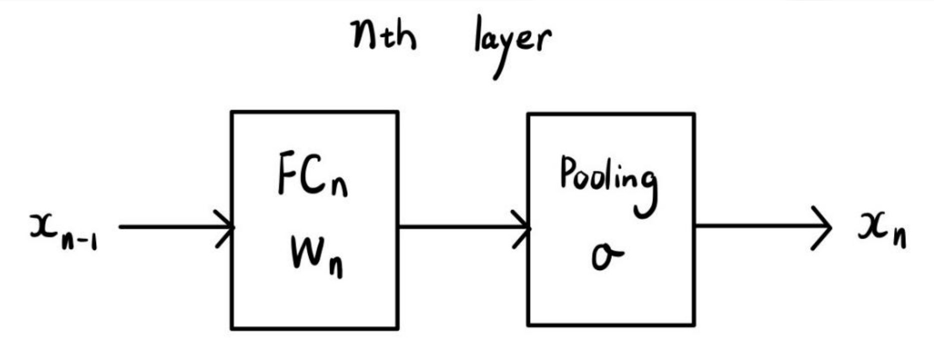
Aggregation

$$(W_1\Pi_j + W_1\Pi_{j'})/2 \neq W_1\Pi \text{ for any } \Pi$$

Solution: $(W_1\Pi_j\Pi_j^T + W_1\Pi_{j'}\Pi_{j'}^T)/2 = W_1$

FedMA algorithm

- Permutation invariance （FCs）

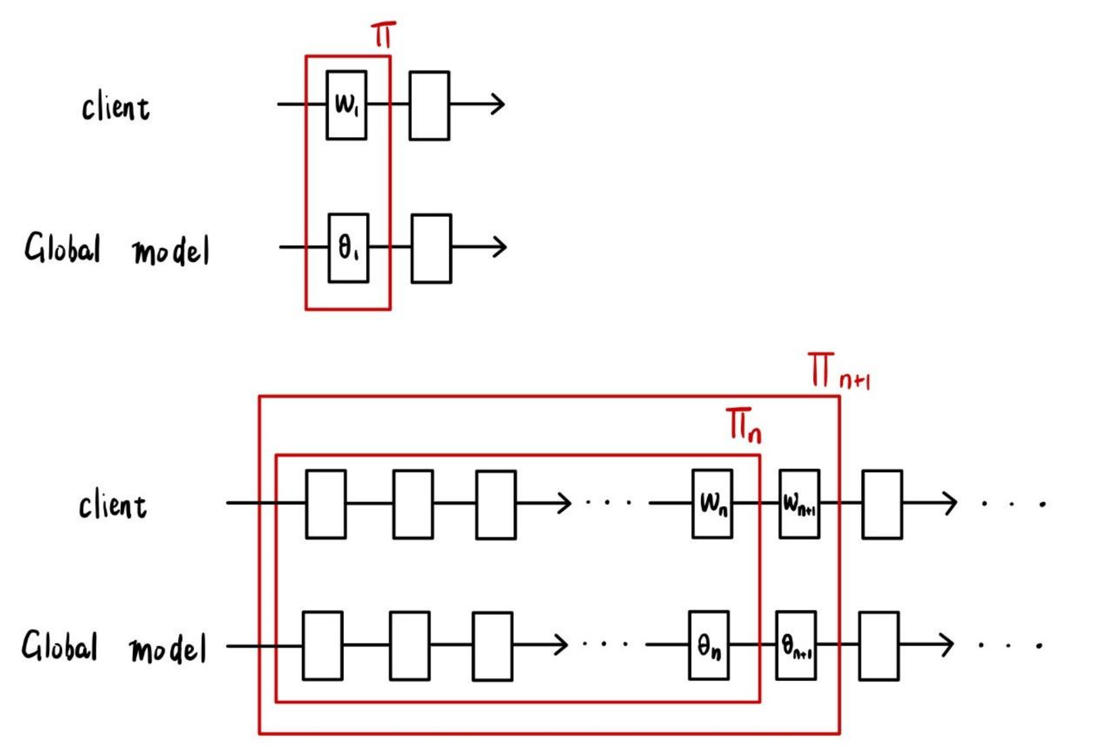


Simple FCs: $\hat{y} = \sigma(xW_1\Pi)\Pi^T W_2$

Deep FCs: $x_n = \sigma(x_{n-1}\Pi_{n-1}^T W_n \Pi_n)$

FedMA algorithm

- Permutation invariance （FCs)



$$x_n = \sigma(x_{n-1}\Pi_{n-1}^T W_n \Pi_n)$$

FedMA algorithm

- Permutation invariance （CNN)

FC : $$x_n = \sigma(x_{n-1}\Pi_{n-1}^T W_n \Pi_n)$$

CNN : $$x_n = \sigma(\text{Conv}(x_{n-1}, \Pi_{n-1}^T W_n \Pi_n))$$

FedMA algorithm

- Permutation invariance (recall)

Client A:

$$\{W_1\Pi_j, \Pi_j^T W_2\}$$

Client B:

$$\{W_1\Pi_{j'}, \Pi_{j'}^T W_2\}$$

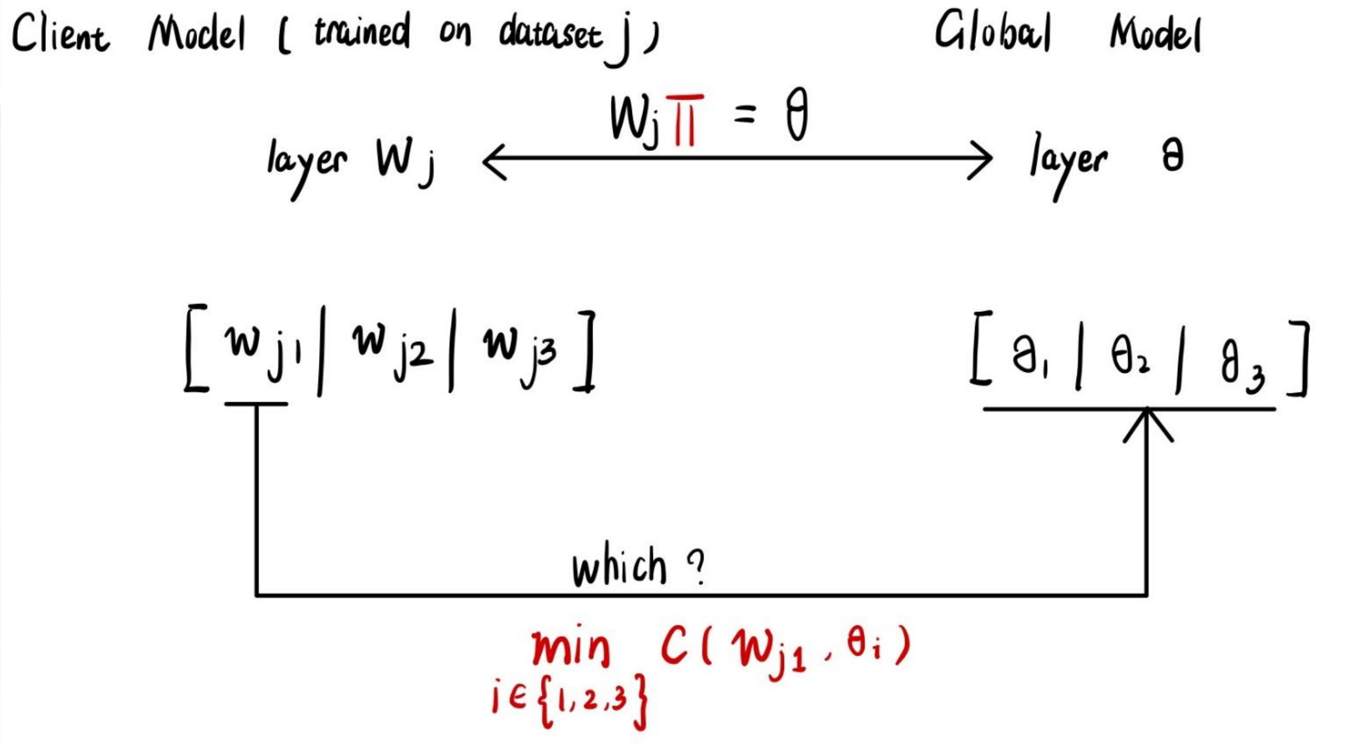
Aggregation

$$(W_1\Pi_j + W_1\Pi_{j'})/2 \neq W_1\Pi \text{ for any } \Pi$$

Solution: $(W_1\Pi_j\Pi_j^T + W_1\Pi_{j'}\Pi_{j'}^T)/2 = W_1$

FedMA algorithm

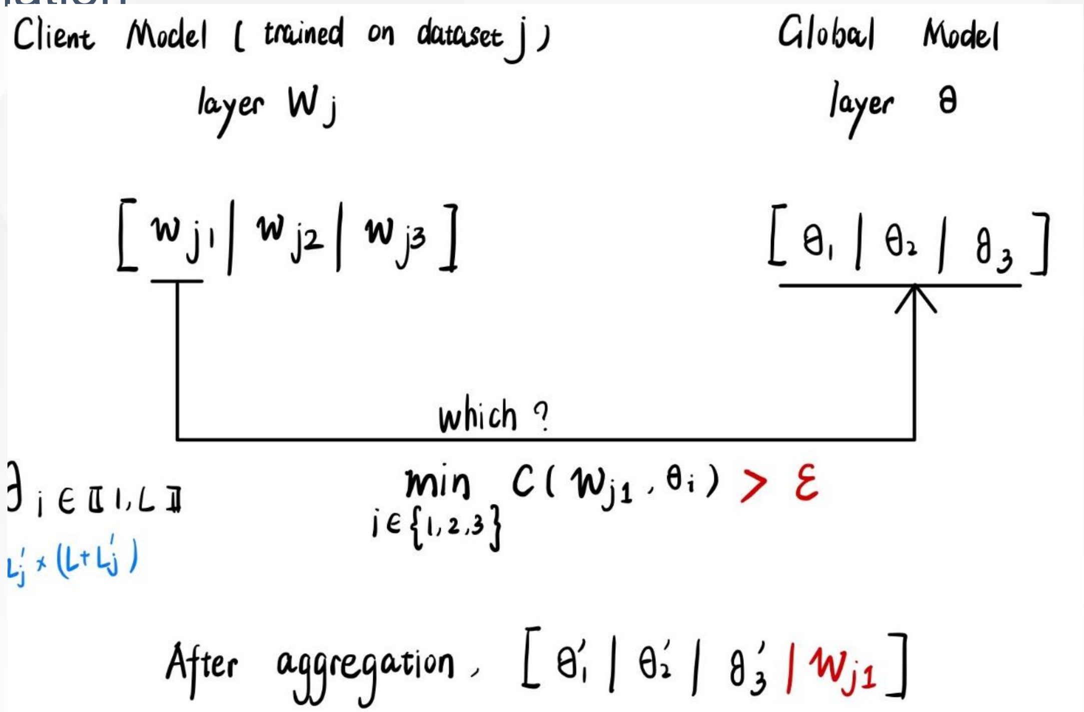
- Matched averaging formulation



$$\min_{\{\pi^j_{li}\}} \sum_{i=1}^{L} \sum_{j,l} \min_{\theta_i} \pi^j_{li} c(w_{jl}, \theta_i) \text{ s.t. } \sum_i \pi^j_{li} = 1 \,\forall\, j, l; \sum_l \pi^j_{li} = 1 \,\forall\, i, j.$$

FedMA algorithm

- Matched averaging formulation

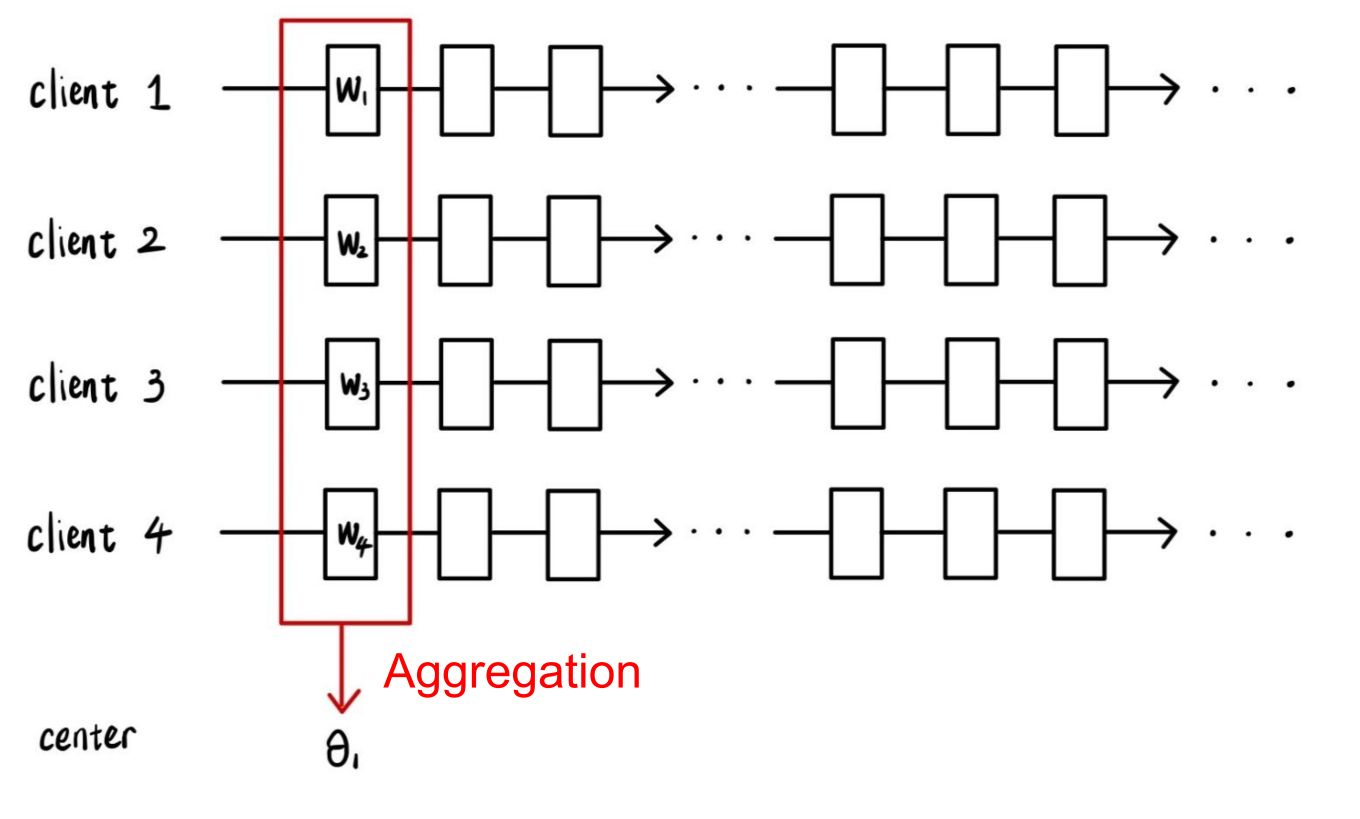


$$\min_{\{\pi^{j'}_{li}\}_{l,i}} \sum_{i=1}^{L+L_{j'}} \sum_{j=1}^{L_{j'}} \pi^{j'}_{li} C^{j'}_{li} \text{ s.t. } \sum_i \pi^{j'}_{li} = 1 \,\forall\, l;\ \sum_l \pi^{j}_{li} \in \{0,1\} \,\forall\, i, \text{ where}$$

$$C^{j'}_{li} = \begin{cases} c(w_{j'l},\theta_i), & i \le L \\ \epsilon + f(i), & L < i \le L + L_{j'}. \end{cases}$$

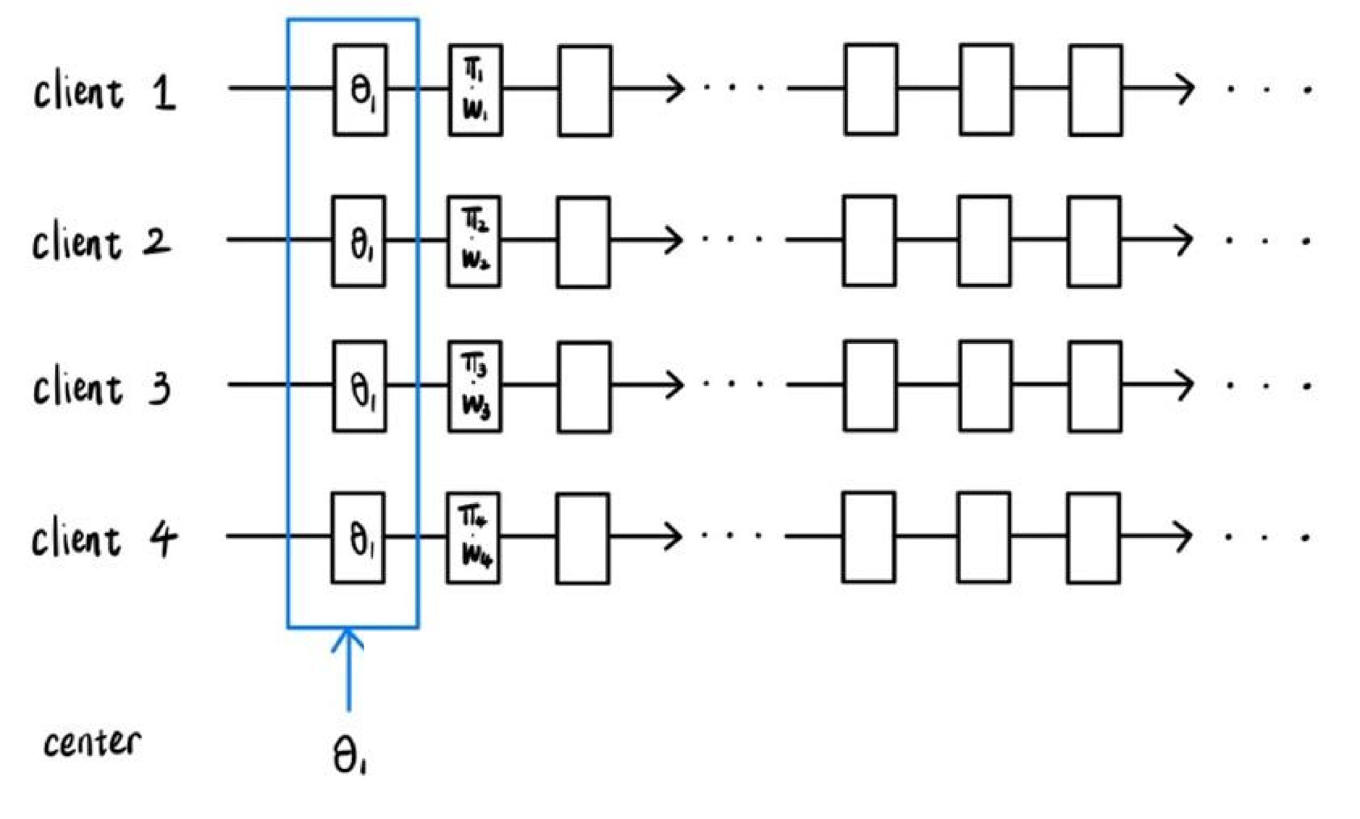
FedMA algorithm

- FedMA (https://github.com/IBM/FedMA)



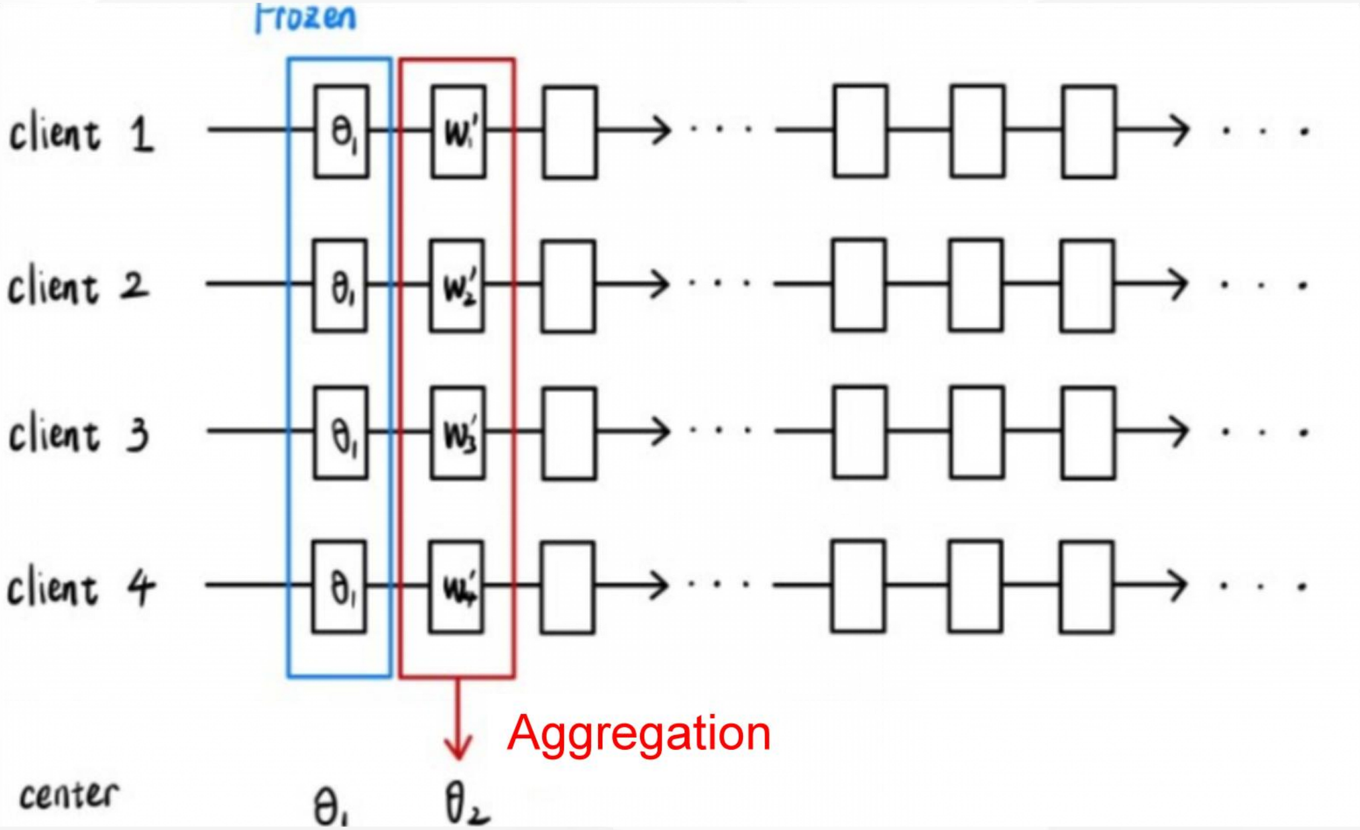
FedMA algorithm

- FedMA (https://github.com/IBM/FedMA)



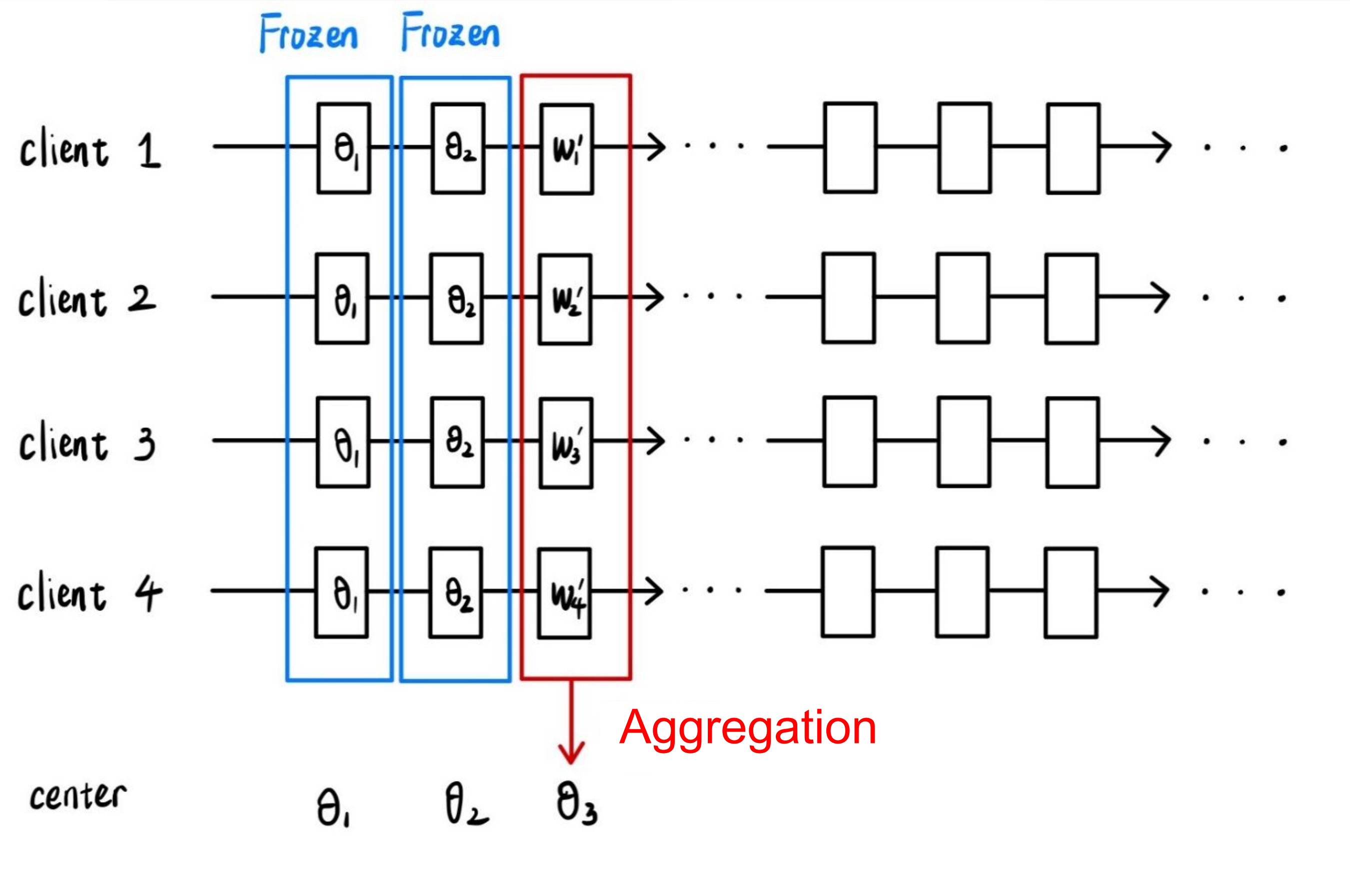
FedMA algorithm

- FedMA (https://github.com/IBM/FedMA)



FedMA algorithm

- FedMA (https://github.com/IBM/FedMA)



FedMA algorithm

- FedMA (https://github.com/IBM/FedMA)

Algorithm 1: Federated Matched Averaging (FedMA)

Input : local weights of N-layer architectures $\{W_{j,1}, \ldots, W_{j,N}\}_{j=1}^{J}$ from J clients
Output: global weights $\{W_1, \ldots, W_N\}$

$n = 1$;
while $n \leq N$ **do**
  if $n < N$ **then**
    $\{\Pi_j\}_{j=1}^{J} = \text{BBP-MAP}\left(\{W_{j,n}\}_{j=1}^{J}\right)$;   `// call BBP-MAP to solve Eq. 2`
    $W_n = \frac{1}{J} \sum_j W_{j,n} \Pi_j^T$;
  else
    $\boxed{W_n = \sum_{k=1}^{K} \sum_j p_{jk} W_{jl,n} \text{ where } p_k \text{ is fraction of data points with label } k \text{ on worker } j;}$
  end
  for $j \in \{1, \ldots, J\}$ **do**
    $W_{j,n+1} \leftarrow \Pi_j W_{j,n+1}$;   `// permutate the next-layer weights`
    Train $\{W_{j,n+1}, \ldots, W_{j,L}\}$ with W_n frozen;
  end
  $n = n + 1$;
end

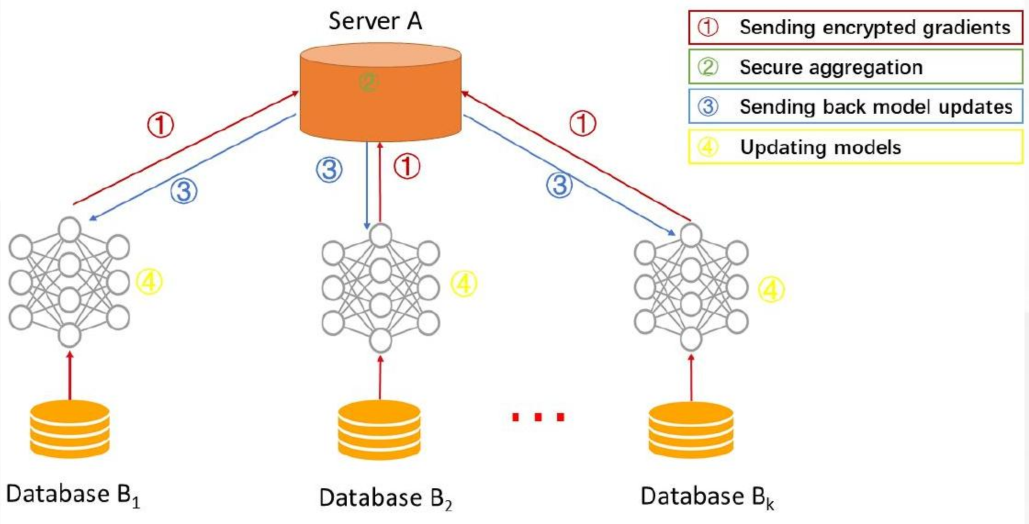
06

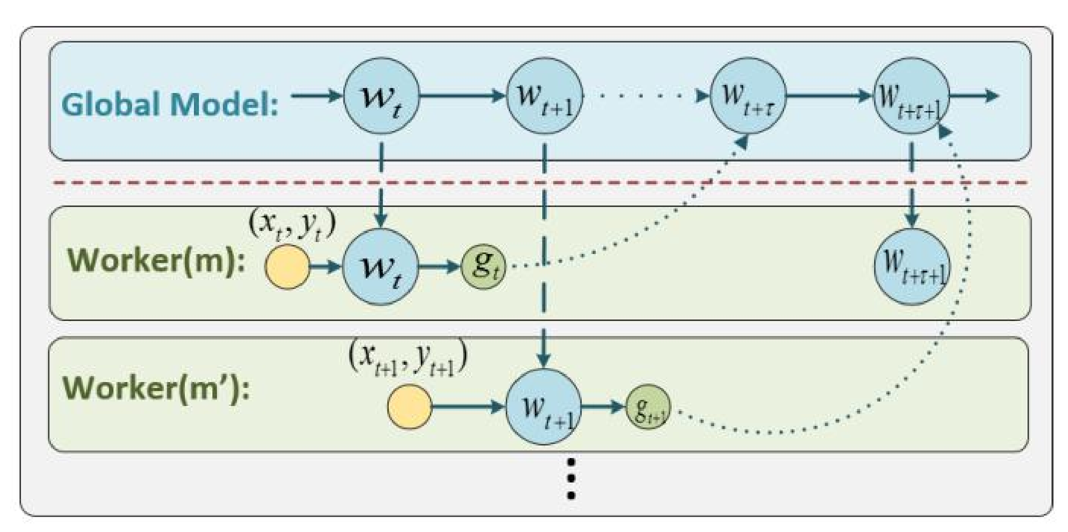
Sageflow algorithm

- Staleness-aware grouping
- Entropy-based filtering
- Loss-weighted averaging

Sageflow algorithm

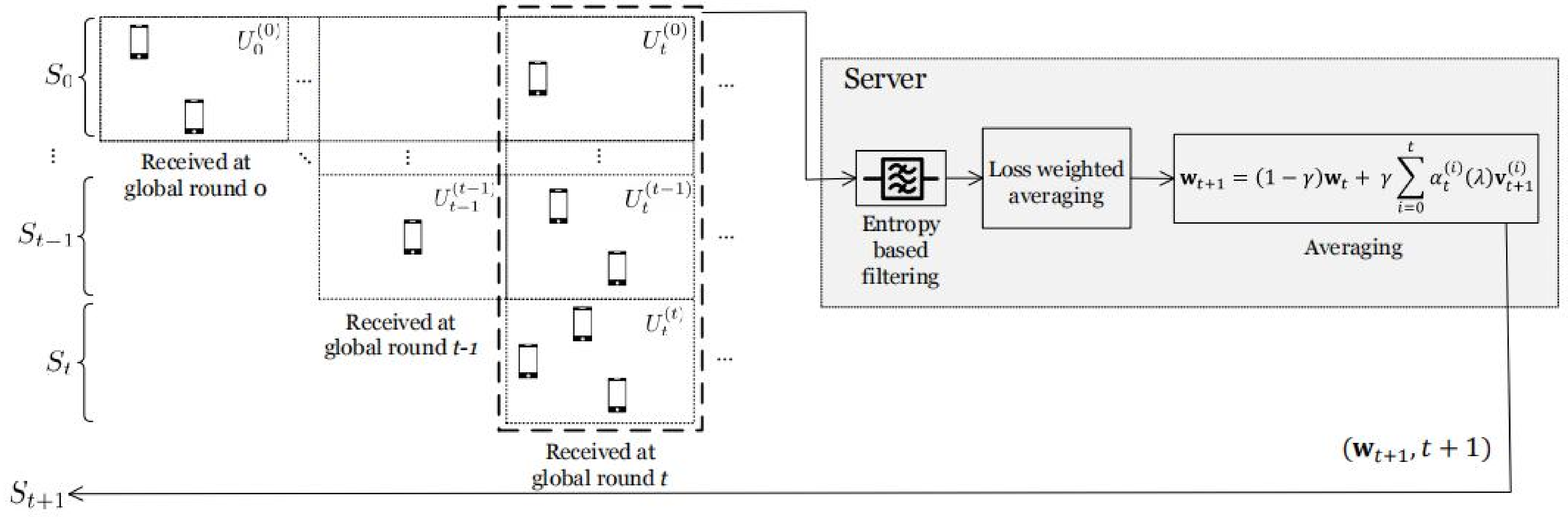
- Stragglers: slow devices
  - **Keep waiting**: slow down the overall process
  - **Drop out**: important data missing
  - **Asynchronous**(staleness): + adversaries?
- Attackers: malicious attacks launched by adversaries
  - **untargeted attacks**: model poisoning, data poisoning
  - **targeted/backdoor attacks**: misclassify the targeted subtasks
  - Robust Federated Averaging & Multi-Krum
- Large portion of adversaries
  - Straggler: increase attack ratio





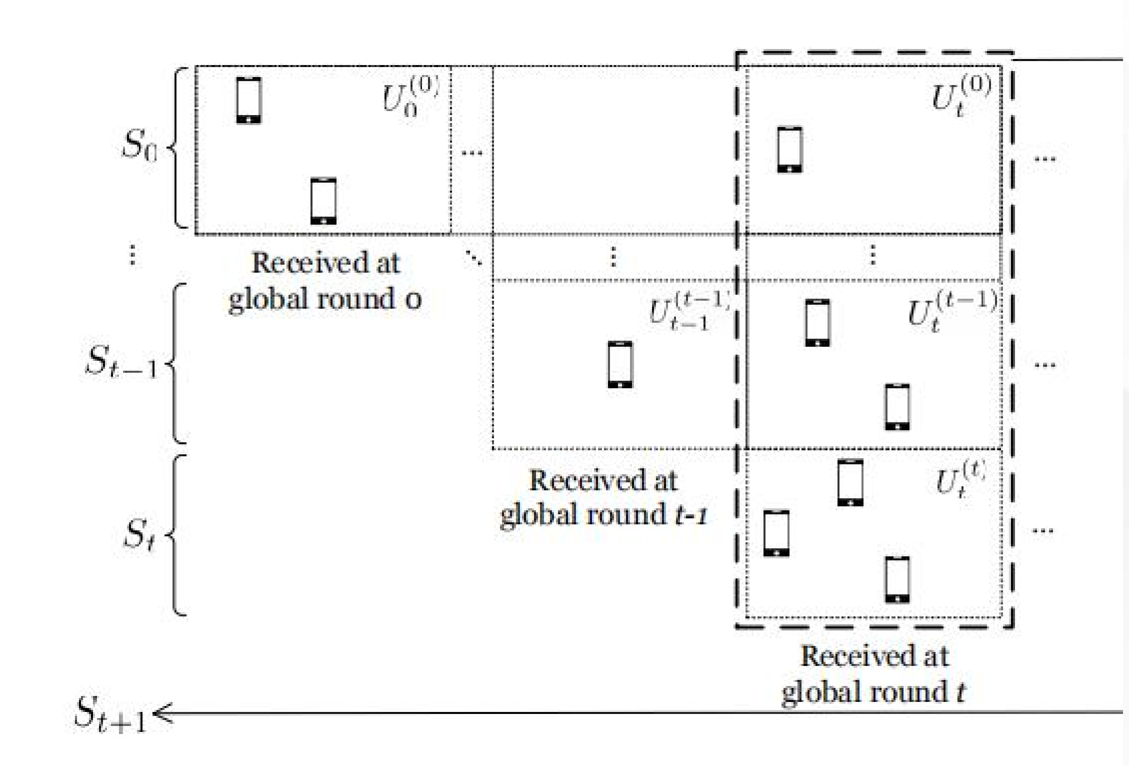
Sageflow algorithm

- Staleness-aware grouping
- Entropy-based filtering + Loss-weighted averaging



Sageflow algorithm

- Staleness-aware grouping
  - perform **periodic** global aggregation(fixed time deadline)
  - allow stragglers to be aggregated in **later rounds**
  - group with **same staleness** -> group representative model
  - aggregate according to **staleness**



Staleness group   Number of data samples

$$\mathbf{v}_{t+1}^{(i)} = \sum_{k \in U_t^{(i)}} \frac{m_k}{\sum_{k \in U_t^{(i)}} m_k} \mathbf{w}_i(k)$$

$$\mathbf{w}_{t+1} = (1-\gamma)\mathbf{w}_t + \gamma \sum_{i=0}^{t} \alpha_t^{(i)}(\lambda)\mathbf{v}_{t+1}^{(i)}$$

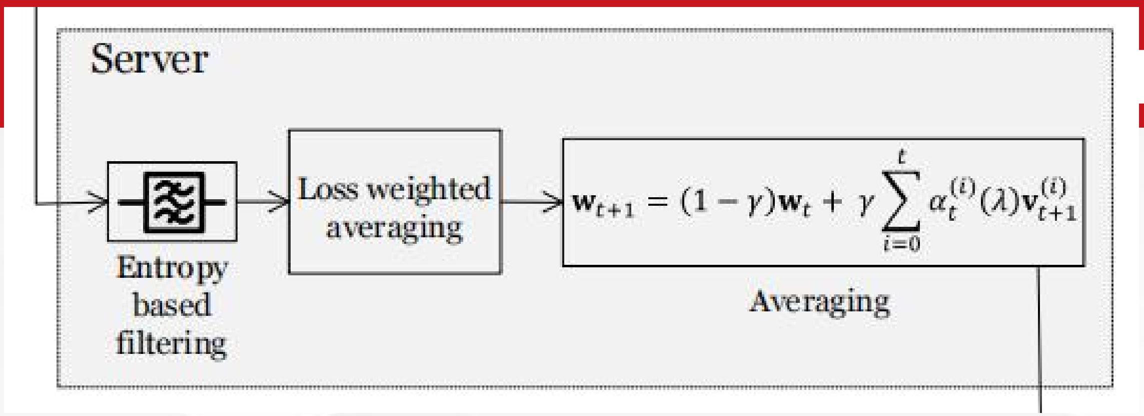
$$\sum_{i=0}^{t} \alpha_t^{(i)}(\lambda)\mathbf{v}_{t+1}^{(i)}$$

$$\alpha_t^{(i)}(\lambda) \propto \frac{\sum_{k \in U_t^{(i)}} m_k}{(t-i+1)^{\lambda}}$$

Staleness function

Sageflow algorithm

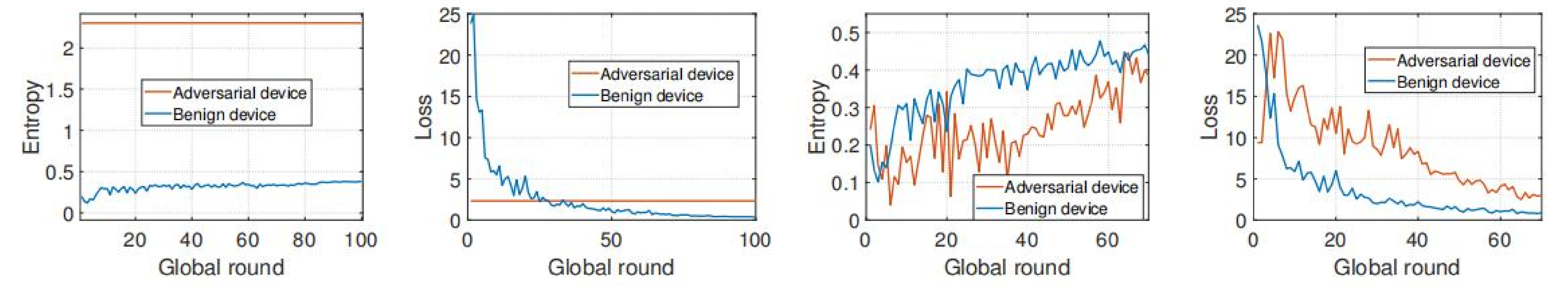
- Entropy-based filtering
  - **Public data** on server
  - **Filter out** high **entropy** models (loss)
  - For model poisoning



$$E_{x_{pub,j}}(k) = -\sum_{q=1}^{Q} P^{(q)}_{x_{pub,j}}(k) \log P^{(q)}_{x_{pub,j}}(k).$$

$$E(k) = \frac{1}{n_{pub}} \sum_{j=1}^{n_{pub}} E_{x_{pub,j}}(k)$$

Shannon entropy



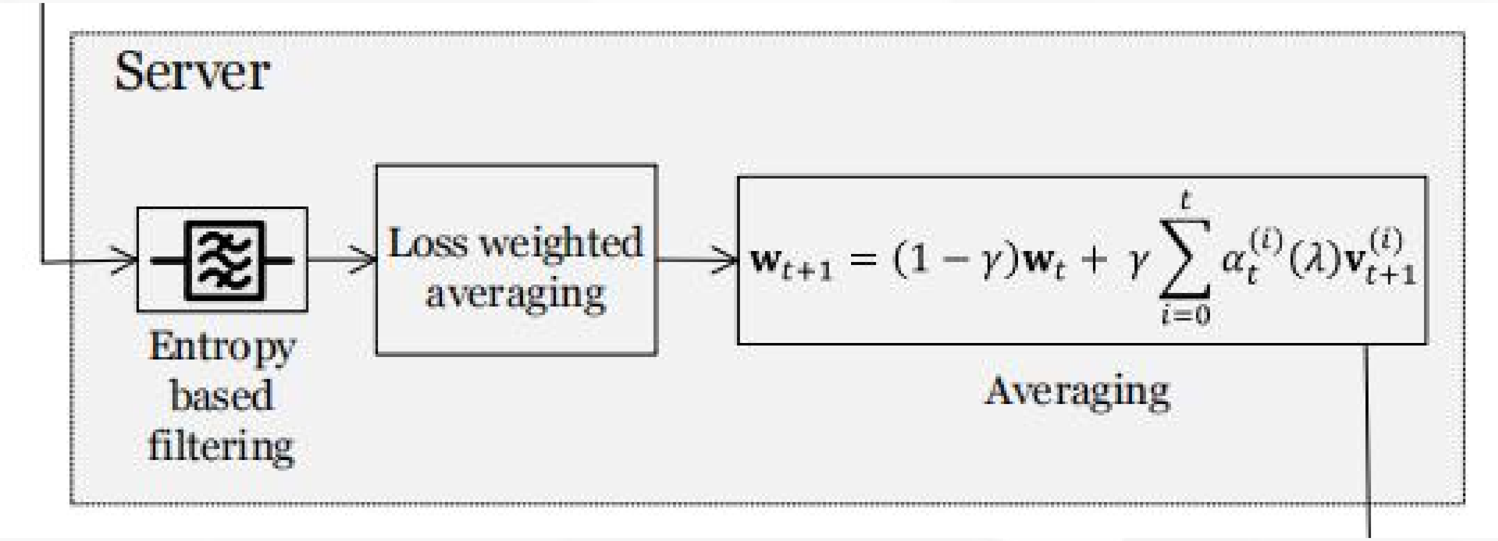
(a) Model poison, entropy (b) Model poison, loss (c) Data poison, entropy (d) Data poison, loss

Sageflow algorithm

- Loss-weighted averaging
  - Aggregation weight according to local models' **measured qualities**
  - Measure by **loss** on public data
  - data-poisoned model -> small weight + less impact
  - For data poisoning & scaled backdoor

$$\beta_t^{(k)}(\delta) \propto \frac{m_k}{\{F_{pub}(\mathbf{w}_t(k))\}^\delta} \text{ and } \sum_{k\in S_t} \beta_t^{(k)}(\delta) = 1.$$

$$\mathbf{w}_{t+1} = \sum_{k\in S_t} \beta_t^{(k)}(\delta)\mathbf{w}_t(k)$$



Sageflow algorithm

- Time complexity

Public data | Model parameters | Clients number

$$\mathcal{O}(n_{pub}|w|K)$$

Algorithm 1 Proposed Sageflow Algorithm

Input: Initialized model $\mathbf{w}_0$, **Output:** Final global model $\mathbf{w}_T$

Process at the Server

for each global round $t = 0, 1, ..., T-1$ **do**
  Choose S_t and send the current model and the global round $(\mathbf{w}_t, t)$ to the devices
  Wait for T_d and then:
  for $i = 0, 1, ..., t$ **do**
    $U_t^{(i)}(E_{th}) = \{k \in U_t^{(i)} | E(k) < E_{th}\}$ // Entropy-based filtering in each group
    $\mathbf{v}_{t+1}^{(i)} = \sum_{k \in U_t^{(i)}(E_{th})} \beta_t^{(k)}(\delta)\mathbf{w}_i(k)$ // Loss-weighted averaging in each group (with same staleness)
  end for
  $\mathbf{w}_{t+1} = (1-\gamma)\mathbf{w}_t + \gamma \sum_{i=0}^{t} \alpha_t^{(i)}(\lambda)\mathbf{v}_{t+1}^{(i)}$ // Averaging of representative models (with different staleness)
end for

Process at the Device: Device k receives $(\mathbf{w}_t, t)$ from the server and performs local updates to obtain $\mathbf{w}_t(k)$. Then each benign device k sends $(\mathbf{w}_t(k), t)$ to the server, while a malicious adversary sends a poisoned model depending on the type of attack.

Sageflow algorithm

- Theoretical Analysis
- Convergence analysis
  - Assumption 1: μ-strongly convex + L-smooth
  - Assumption 2: unbiased estimation
- Theoretical bound

$$F(x) \le F(y) + \nabla F(x)^T(x-y) - \frac{\mu}{2}\|x-y\|^2$$

$$F(x) \ge F(y) + \nabla F(x)^T(x-y) - \frac{L}{2}\|x-y\|^2$$

$$\mathbb{E}\|\nabla F_k(\mathbf{w}_t^i(k), \xi_t^i(k)) - \nabla F(\mathbf{w}_t^i(k))\|^2 \le \rho_1$$

$$\mathbb{E}[F(\mathbf{w}_T) - F(\mathbf{w}^*)] \le \nu^T[F(\mathbf{w}_0) - F(\mathbf{w}^*)] + (1-\nu^T)Z(\lambda, E_{th}, \delta)$$

Convergence speed

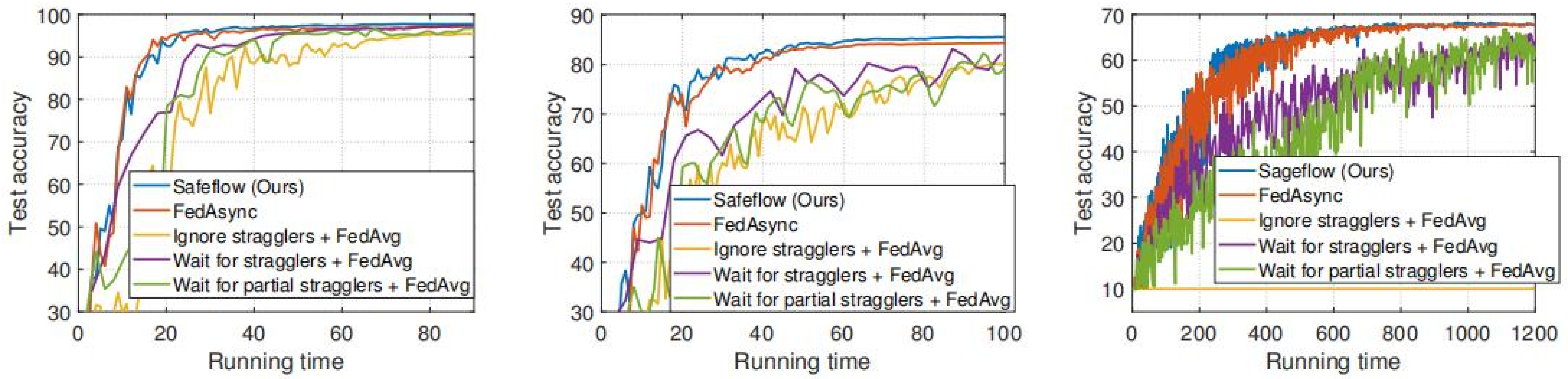
Error

Sageflow algorithm

- Datasets: MNIST, FMNIST, CIFAR10
  - 2% as public data
- Models: CNN(2conv+2fc), CNN(2conv+1fc), CNN(VGG-11)
  - ignore batchnorm
- FL setting: 100 clients, two classes for each client, 5 local epochs, batch size of 10

Sageflow algorithm

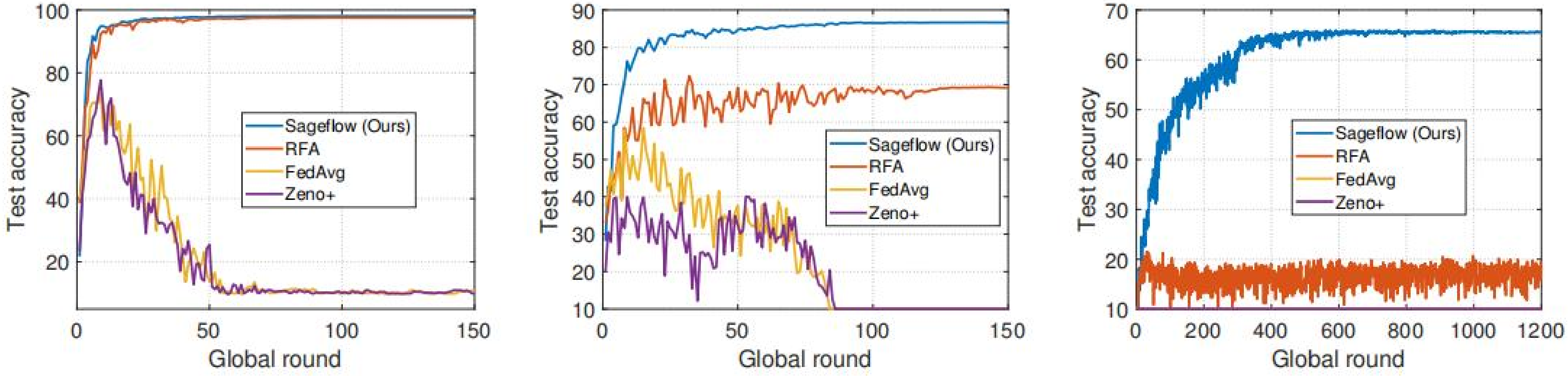
- Only stragglers: 10% participants
- Baselines: FedAvg(waiting, ignoring, waiting 50%), FedAsync
- Settings: uniform delay of [0,1,2] global rounds
- Ignoring lose significant data converges to a **suboptimal point**
- Waiting(all, 50%) requires the **largest running time** until convergence



(a) MNIST (b) FMNIST (c) CIFAR10

Sageflow algorithm

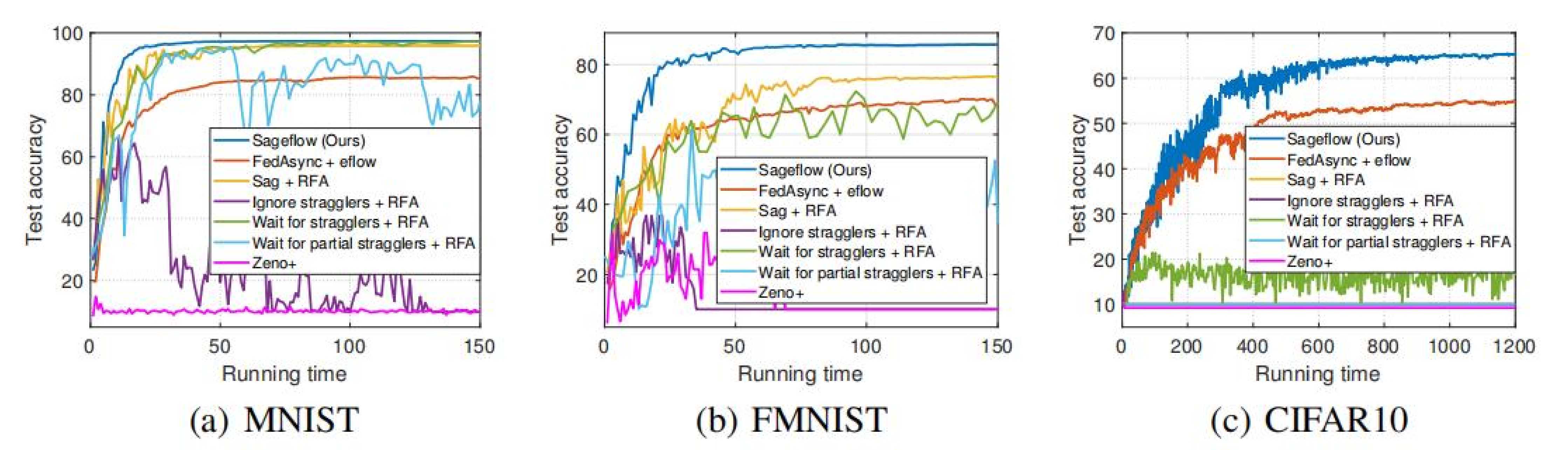
- Only adversaries: 20% participants
- Baselines: RFA, FedAvg, synchronized Zeno+, Multi-Krum
- Attacks: model(-0.1w), data(label-flipping), backdoor(model replacement, pixel-pattern attack)
- FedAvg **does not work well** on all datasets
- RFA: **complex model** led to worse performance
- Zeno+: bad on poisoning **but** good for backdoor
- Sageflow: **slow down** posioning



(a) MNIST (b) FMNIST (c) CIFAR10

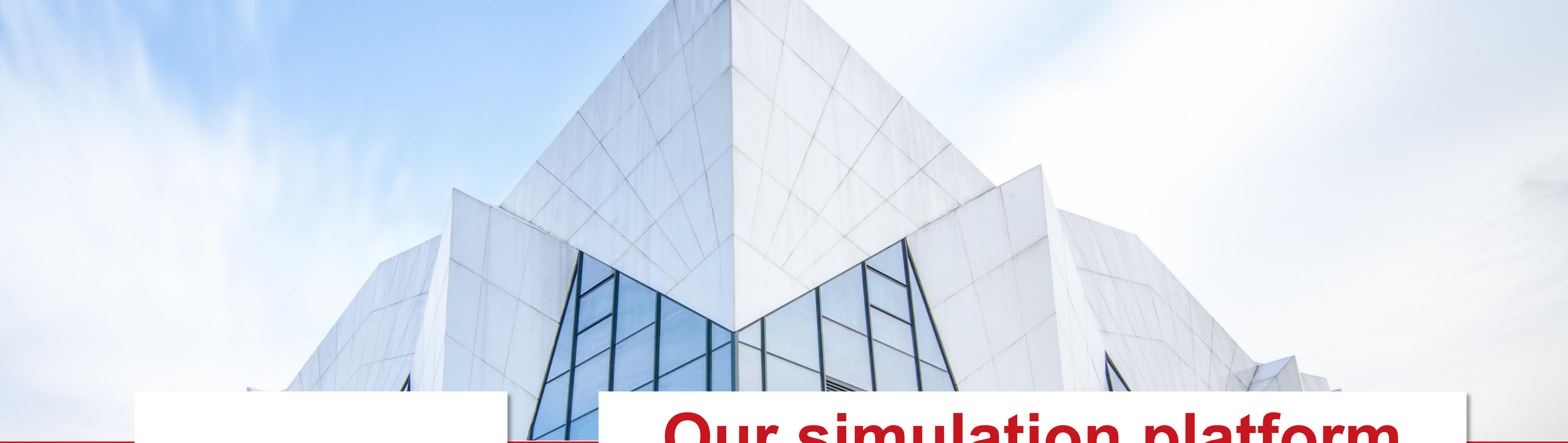
Sageflow algorithm

- Stragglers + adversaries: 20%(model/data), 10%(backdoor) participants
- Baselines: asynchronized Zeno+, Multi-Krum
- Zeno+: **does not perform well**(ignore staleness & entropy)
- Waiting + RFA: suffer from **straggler**
- Ignoring/Sag + RFA: poor(**high attack ratio**)
- eflow/RFA + FedAsync: poor(**one-by-one arrivals**)



Sageflow algorithm

- Sageflow: robust FL scheme handle both **stragglers and adversaries**
  - staleness-aware grouping: stragglers
  - entropy-based filtering: model poisoning
  - loss-weighted averaging: data poisoning + backdoor
- Theoretical convergence analysis
- Extensive experimental results
- Future issues: Sageflow + secure aggregation



07

Our simulation platform

- Overview
- Dataset generation
- System introduction
- FedAvg example

Our simulation platform

Overview

- Currently, 611 stars, 151 forks
- 29 algorithms, 8 famous datasets + 3 IoT datasets +3 Cross-domain datasets
- Record the GPU memory usage for the model
- Differential privacy

Our simulation platform

Baselines

Traditional FL

- FedAvg — Communication-Efficient Learning of Deep Networks from Decentralized Data *AISTATS 2017*

Update-correction-based FL

- SCAFFOLD - SCAFFOLD: Stochastic Controlled Averaging for Federated Learning *ICML 2020*

Regularization-based FL

- FedProx — Federated Optimization in Heterogeneous Networks *MLsys 2020*
- FedDyn — Federated Learning Based on Dynamic Regularization *ICLR 2021*

Model-splitting-based FL

- MOON — Model-Contrastive Federated Learning *CVPR 2021*

Knowledge-distillation-based FL

- FedGen — Data-Free Knowledge Distillation for Heterogeneous Federated Learning *ICML 2021*

Our simulation platform

- Baselines

Personalized FL

- **FedMTL (not MOCHA)** — Federated multi-task learning *NeurIPS 2017*
- **FedBN** — FedBN: Federated Learning on non-IID Features via Local Batch Normalization *ICLR 2021*

Meta-learning-based pFL

- **Per-FedAvg** — Personalized Federated Learning with Theoretical Guarantees: A Model-Agnostic Meta-Learning Approach *NeurIPS 2020*

Regularization-based pFL

- **pFedMe** — Personalized Federated Learning with Moreau Envelopes *NeurIPS 2020*
- **Ditto** — Ditto: Fair and robust federated learning through personalization *ICML 2021*

Personalized-aggregation-based pFL

- **APFL** — Adaptive Personalized Federated Learning *2020*
- **FedFomo** — Personalized Federated Learning with First Order Model Optimization *ICLR 2021*
- **FedAMP** — Personalized Cross-Silo Federated Learning on non-IID Data *AAAI 2021*
- **FedPHP** — FedPHP: Federated Personalization with Inherited Private Models *ECML PKDD 2021*
- **APPLE** — Adapt to Adaptation: Learning Personalization for Cross-Silo Federated Learning *IJCAI 2022*
- **FedALA** — FedALA: Adaptive Local Aggregation for Personalized Federated Learning *AAAI 2023*

Our simulation platform

Baselines

Model-splitting-based pFL

- **FedPer** — Federated Learning with Personalization Layers *2019*
- **LG-FedAvg** — Think Locally, Act Globally: Federated Learning with Local and Global Representations *2020*
- **FedRep** — Exploiting Shared Representations for Personalized Federated Learning *ICML 2021*
- **FedRoD** — On Bridging Generic and Personalized Federated Learning for Image Classification *ICLR 2022*
- **FedBABU** — Fedbabu: Towards enhanced representation for federated image classification *ICLR 2022*
- **FedGC** — Federated Learning for Face Recognition with Gradient Correction *AAAI 2022*

Knowledge-distillation-based pFL

- **FedDistill** — Federated Knowledge Distillation *2020*
- **FML** — Federated Mutual Learning *2020*
- **FedKD** — Communication-efficient federated learning via knowledge distillation *Nature Communications 2022*
- **FedProto** — FedProto: Federated Prototype Learning across Heterogeneous Clients *AAAI 2022*
- **FedPCL (w/o pre-trained models)** — Federated learning from pre-trained models: A contrastive learning approach *NeurIPS 2022* ("Our proposed framework is limited to the cases where pre-trained models are available." from https://arxiv.org/pdf/2209.10083.pdf (p. 18))
- **FedPAC** — Personalized Federated Learning with Feature Alignment and Classifier Collaboration *ICLR 2023*

Our simulation platform

- Datasets

Datasets and Separation (updating)

For the ***label skew*** setting, I introduce 8 famous datasets: **MNIST**, **Fashion-MNIST**, **Cifar10**, **Cifar100**, **AG_News**, **Sogou_News** (If ConnectionError raises, please use the given downloaded file in `./dataset`), and **Tiny-ImageNet** (fetch raw data from this site), they can be easy split into **IID** and **non-IID** version. Since some codes for generating datasets such as splitting are the same for all datasets, I move these codes into `./dataset/utils/dataset_utils.py`. In **non-IID** setting, two situations exist. The first one is the **pathological non-IID** setting, the second one is **practical non-IID** setting. In the **pathological non-IID** setting, for example, the data on each client only contains the specific number of labels (maybe only two labels), though the data on all clients contains 10 labels such as MNIST dataset. In the **practical non-IID** setting, Dirichlet distribution is utilized (please refer to this paper for details). We can input *balance* for the iid setting, where the data are uniformly distributed.

For the ***feature shift*** setting, I use one dataset that widely used in Domain Adaptation: **AmazonReview** (fetch raw data from this site), **Digit5** (fetch raw data from this site), and **DomainNet**.

For the ***real-world (or IoT)*** setting, I also introduce one naturally separated dataset: **Omniglot** (20 clients, 50 labels), **HAR (Human Activity Recognition)** (30 clients, 6 labels), **PAMAP2** (9 clients, 12 labels). For the details of datasets and FL methods in **IoT**, please refer to my FL-IoT repo.

If you need another data set, just write another code to download it and then using the utils.

Our simulation platform

Models

- for MNIST and Fashion-MNIST

    i. Mclr_Logistic(1*28*28)
    ii. LeNet()
    iii. DNN(1*28*28, 100) # non-convex

- for Cifar10, Cifar100 and Tiny-ImageNet

    i. Mclr_Logistic(3*32*32)
    ii. FedAvgCNN()
    iii. DNN(3*32*32, 100) # non-convex
    iv. ResNet18, AlexNet, MobileNet, GoogleNet, etc.

- for AG_News and Sogou_News

    i. LSTM()
    ii. fastText() in Bag of Tricks for Efficient Text Classification
    iii. TextCNN() in Convolutional Neural Networks for Sentence Classification
    iv. TransformerModel() in Attention is all you need

- for AmazonReview

    i. AmazonMLP() in Curriculum manager for source selection in multi-source domain adaptation

- for Omniglot

    i. FedAvgCNN()

Our simulation platform

Installation

Environments

Install CUDA first.

With the installed conda, we can run this platform in a conda virtual environment called *fl_torch*.

```
conda env create -f env_cuda_latest.yaml
```

Our simulation platform

Dataset

- MNIST

```
cd ./dataset
python generate_mnist.py iid - - # for iid and unbalanced scenario
# python generate_mnist.py iid balance - # for iid and balanced scenario
# python generate_mnist.py noniid - pat # for pathological noniid and unbalanced scenario
# python generate_mnist.py noniid - dir # for practical noniid and unbalanced scenario
```

The output of `generate_mnist.py iid - -`

```
Original number of samples of each label: [6903, 7877, 6990, 7141, 6824, 6313, 6876, 7293, 6825, 6958]

Client 0     Size of data: 1064  Labels:  [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
Client 0     Samples of labels:  [(0, 101), (1, 128), (2, 136), (3, 123), (4, 79), (5, 85), (6, 107), (7, 127
--------------------------------------------------
Client 1     Size of data: 1023  Labels:  [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
Client 1     Samples of labels:  [(0, 76), (1, 132), (2, 107), (3, 79), (4, 94), (5, 110), (6, 90), (7, 110),
--------------------------------------------------
Client 2     Size of data: 923   Labels:  [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
Client 2     Samples of labels:  [(0, 136), (1, 89), (2, 84), (3, 88), (4, 78), (5, 124), (6, 120), (7, 66),
--------------------------------------------------
```

Our simulation platform

- Execution

How to start simulating

- Build dataset: Datasets
- Train and evaluate the model:

```
cd ./system
python main.py -data mnist -m cnn -algo FedAvg -gr 2500 -did 0 -go cnn # for FedAvg and MNIST
```

Or you can uncomment the lines you need in `./system/examples.sh` and run:

```
cd ./system
sh examples.sh
```

Note: The hyper-parameters have not been tuned for the algorithms. The values in `./system/examples.sh` are just examples. You need to tune the hyper-parameters by yourself.

Our simulation platform

- Top-down
  - Server: Aggregate, Send, Receive, Evaluate
    - Add
  - Client: Train, Receive, Send
    - Dataset
    - Model
    - Optimizer

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 $t = 1, 2, \ldots$ **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

SJTU

Our simulation platform

- System introduction
  - main.py
    - torchvision.models
  - models.py
  - serverbase.py
  - clientbase.py

Our simulation platform

- System introduction
  - fedoptimizer.py
  - Utils
    - data_utils.py
    - mem_utils.py
    - privacy.py
    - result_utils.py

Our simulation platform

- FedAvg example
  - serveravg.py
  - clientavg.py

谢谢观看

饮 水 思 源　爱 国 荣 校