CS-E4740 - Federated Learning

FL Algorithms

Assoc. Prof. Alexander Jung

Spring 2025

Playlist

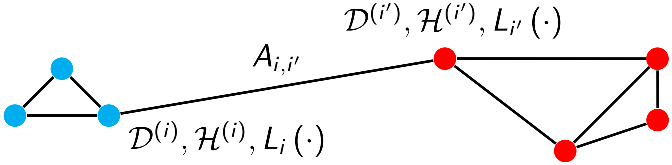
Glossary

Course Site

Outline

Table of Contents

FL Network as a Mathematical Model for FL



- An FL network consists of devices $i = 1, \ldots, n$.
- Some i, i' are connected by an edge with weight $A_{i,i'} > 0$.
- Device i **generates** data $\mathcal{D}^{(i)}$ and **trains** model $\mathcal{H}^{(i)}$.
- Data $\mathcal{D}^{(i)}$ is used to construct a loss func. $L_i(\cdot)$.[1]

[1] Can you think of other constructions of a loss function?

GTV Minimization (for Parametric Models)

We train local models in a collaborative fashion by solving

$$\min_{\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(n)}} \sum_{i=1}^{n} L_i\left(\mathbf{w}^{(i)}\right) + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2 \quad \text{(GTVMin)}.$$

- Solution consists of learnt model params. $\widehat{\mathbf{w}}^{(i)}$.
- The parameter $\alpha \geq 0$ controls the clustering of $\widehat{\mathbf{w}}^{(i)}$.
- For $\alpha = 0$, GTVMin reduces to separate ERM for each i.
- Large α results in $\widehat{\mathbf{w}}^{(i)}$ being nearly constant.[2]

[2] Nearly constant over each connected component of $\mathcal{G}$.

Learning Goals

After completing this module, you know how

- FL alg. can be obtained from **gradient descent**,
- to implement GD as **message passing**,
- to generalize GD to handle **non-parametric models**,
- to implement **asynchronous FL algorithms.**

Table of Contents

Gradient Step for GTVMin

Starting from initial local params. $\mathbf{w}^{(i,0)}$, repeat grad. steps

$$\mathbf{w}^{(i,k+1)} = \mathbf{w}^{(i,k)} - \eta_{k,i} \left[\nabla L_i \left(\mathbf{w}^{(i,k)} \right) + 2\alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \left(\mathbf{w}^{(i,k)} - \mathbf{w}^{(i',k)} \right) \right]$$

- The learning rate $\eta_{k,i}$ determines extent of update.
- $\nabla L_i \left(\mathbf{w}^{(i,k)} \right)$ **steers** the update towards **min. local loss**.
- $\left(\mathbf{w}^{(i,k)} - \mathbf{w}^{(i',k)} \right)$ **steers** to **agree with neighbours.**
- $\alpha A_{i,i'}$ balances those two steering effects.

Synchronous Operation

The gradient step

$$\mathbf{w}^{(i,k+1)} = \mathbf{w}^{(i,k)} - \eta_{k,i}\left[\nabla L_i\left(\mathbf{w}^{(i,k)}\right) + 2\alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}\left(\mathbf{w}^{(i,k)} - \mathbf{w}^{(i',k)}\right)\right]$$

has to be carried out by all nodes $i = 1, \ldots, n$.

When these (local) gradient steps are completed, each node shares its new model params. with its neighbours.

After sharing the model params., start new iteration $k := k+1$.

Message Passing Implementation

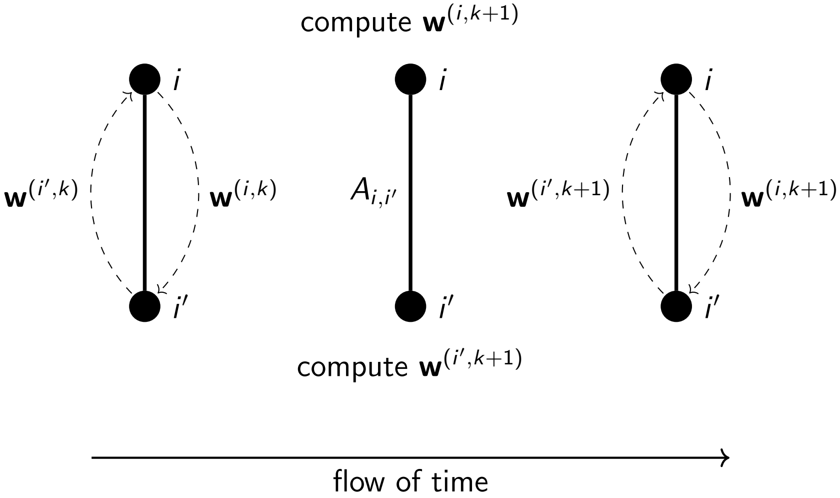


Table of Contents

Federated Gradient Descent (FedGD)

Each node $i = 1, \ldots, n$ initializes

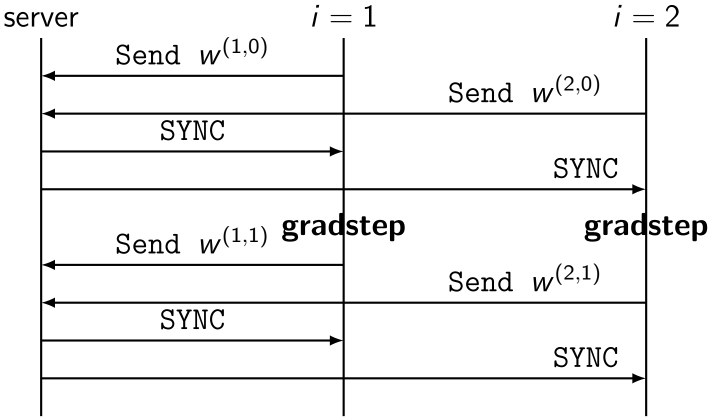
- local model params. $\mathbf{w}^{(i,0)} := \mathbf{0}$, and
- iteration counter $k := 0$.

Repeat the following steps at each node i:

- Send $\mathbf{w}^{(i,k)}$ to all neighbours $\mathcal{N}^{(i)}$.
- Do a gradient step.
- Increment iteration counter $k := k + 1$.

CAUTION: Nodes must execute steps synchronously!

Implementing FedGD with a Sync-Server



Python demo

Federated Stochastic Gradient Descent (FedSGD)

- Consider FL network with node i carrying the local dataset

$$\mathcal{D}^{(i)} = \left\{ \left(\mathbf{x}^{(i,1)}, y^{(i,1)}\right), \ldots, \left(\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}\right) \right\}.$$

- Node i uses local loss function

$$L_i\left(\mathbf{w}^{(i)}\right) := (1/m_i) \sum_{r=1}^{m_i} \left(y^{(i,r)} - \left(\mathbf{w}^{(i)}\right)^T \mathbf{x}^{(i,r)} \right)^2.$$

- FedGD requires to compute gradient,

$$\nabla L_i\left(\mathbf{w}^{(i)}\right) = (-2/m_i) \sum_{r=1}^{m_i} \mathbf{x}^{(i,r)} \left(y^{(i,r)} - \left(\mathbf{w}^{(i)}\right)^T \mathbf{x}^{(i,r)} \right).$$

Stochastic Gradient Approximation

For some applications, the computation of

$$\sum_{r=1}^{m_i} \mathbf{x}^{(i,r)} \left(y^{(i,r)} - \left(\mathbf{w}^{(i)}\right)^T \mathbf{x}^{(i,r)} \right)$$

is intractable, e.g., too many data points or too slow access.

$\Rightarrow$ Use instead a sum over random subset $\mathcal{B} \subseteq \{1, \dots, m_i\}$,

$$\sum_{r \in \mathcal{B}} \mathbf{x}^{(i,r)} \left(y^{(i,r)} - \left(\mathbf{w}^{(i)}\right)^T \mathbf{x}^{(i,r)} \right).$$

We refer to $\mathcal{B}$ as batch with batch size $|\mathcal{B}|$.

Federated Averaging (FedAvg)

- Some FL applications use common model at all nodes,

$$\mathbf{w}^{(i)} = \mathbf{w}^{(i')} \quad \forall\, i, i' \in \mathcal{V}.$$

- GTVMin becomes constrained optimization problem:

$$\min_{\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(n)}} \sum_{i \in \mathcal{V}} L_i \left(\mathbf{w}^{(i)}\right) \text{s.t.} \quad \mathbf{w}^{(i)} = \mathbf{w}^{(i')} \quad \forall\, i, i' \in \mathcal{V}.$$

- For diffable $L_i \left(\mathbf{w}^{(i)}\right)$ we can apply projected GD.

- Projection step amounts to averaging $(1/n) \sum_{i=1}^{n} \mathbf{w}^{(i)}$.

(Almost) FedAvg

Init. counter (clock) $k := 0$ and model params $\widehat{\mathbf{w}} := \mathbf{0}$.

1. **Broadcast.** Server sends $\widehat{\mathbf{w}}$ to all nodes $i \in \mathcal{V}$.
2. **Local Gradient Step.** Each node computes
$$\mathbf{w}^{(i,k)} = \widehat{\mathbf{w}} - \eta_{k,i} \nabla L_i \left(\widehat{\mathbf{w}} \right).$$
3. **Collect.** Nodes send $\mathbf{w}^{(i,k)}$ back to server.
4. **Aggregate.** Server computes $\widehat{\mathbf{w}} := (1/n) \sum_{i=1}^{n} \mathbf{w}^{(i,k)}$.
5. **Clock Tick.** Server increments $k := k + 1$. Go to step 1.

FedAvg

We obtain FedAvg via the following modifications:[3]

- Use (stochastic) approximations of gradients.
- Instead of single GD step, compute several GD steps.
- Each iteration involves only a subset of nodes.

Python demo

[3] B. McMahan et.al., Communication-Efficient Learning of Deep Networks from Decentralized Data, PMLR, 2017

FedProx

FedProx replaces GD steps in FedAvg with[4]

$$\mathbf{w}^{(i)} := \underset{\mathbf{v} \in \mathbb{R}^d}{\operatorname{argmin}} \left[L_i(\mathbf{v}) + (1/\eta) \left\| \mathbf{v} - \widehat{\mathbf{w}} \right\|_2^2 \right]$$

with current global model params $\widehat{\mathbf{w}}$.

Empirical studies found FedProx to result in more robust FL systems compared to FedAvg.

FedProx seems to require less tuning for FL networks with devices having varying computational power.

[4] T. Li, et.al, Federated Optimization in Heterogeneous Networks, Proc. of Machine Learning and Systems 2, 2020.

Federated Relaxation (FedRelax)

- Consider GTVMin objective function

$$f\left(\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(n)}\right)=\sum_{i=1}^{n} L_i\left(\mathbf{w}^{(i)}\right)+\alpha\sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\left\|\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\right\|_2^2.$$

- Complicated due to coupling terms $A_{i,i'}\left\|\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\right\|_2^2$.

- Without coupling, GTVMin would be much easier.

- Optimize $f(\cdot)$ w.r.t. $\mathbf{w}^{(i)}$, holding $\left\{\mathbf{w}^{(i')}\right\}_{i'\in\mathcal{V}\setminus\{i\}}$ fixed![5]

[5] Similar idea is used in the Jacobi method for solving linear equations.

FedRelax for Parametric Models

- **Init.** Set counter $k := 0$, local model params. $\widehat{\mathbf{w}}_0^{(i)} := \mathbf{0}$.
- **Repeat until stopping criterion:**
  - Each node i shares $\widehat{\mathbf{w}}_k^{(i)}$ with neighbours $\mathcal{N}^{(i)}$.
  - **Local Update.** Each node i computes[6]

$$\mathbf{w}^{(i,k+1)} := \underset{\mathbf{w}^{(i)} \in \mathbb{R}^d}{\operatorname{argmin}} L_i\left(\mathbf{w}^{(i)}\right) + \alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \mathbf{w}^{(i',k)} \right\|_2^2 .$$

  - **Clock Tick.** $k := k + 1$.

[6] Note the similarity of local update with ridge regression.

FedRelax for Non-Parametric Models

- **Init.** $k := 0$, construct test-set $\mathcal{D}^{\{i,i'\}}$ for each $\{i,i'\} \in \mathcal{E}$
- **Repeat until stopping criterion:**
  - Each i shares $h^{(i,k)}(\mathbf{x})$ for each $\mathbf{x} \in \mathcal{D}^{\{i,i'\}}$ and $i' \in \mathcal{N}^{(i)}$.
  - **Local Update.** Each node i computes

$$h^{(i,k+1)} \in \operatorname*{argmin}_{h^{(i)} \in \mathcal{H}^{(i)}} L_i\left(h^{(i)}\right) + \alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} D\big(h^{(i)}, h^{(i',k)}\big).$$

  - **Clock Tick.** $k := k+1$.

Here, we use the discrepancy measure

$$D\big(h^{(i)}, h^{(i')}\big) := (1/m') \sum_{\mathbf{x} \in \mathcal{D}^{\{i,i'\}}} \left[h^{(i)}(\mathbf{x}) - h^{(i')}(\mathbf{x})\right]^2.$$

Table of Contents

FL as Fixed-Point Iterations

- Consider an FL net. with parametric local models.

- FL algorithms presented so far can be written as

$$\mathbf{w}^{(i,k+1)} = \mathcal{F}^{(i)}\left(\mathbf{w}^{(1,k)}, \ldots, \mathbf{w}^{(n,k)}\right).$$

- This is a synchronous fixed-point iteration with

$$\mathcal{F}^{(i)} : \mathbb{R}^{nd} \rightarrow \mathbb{R}^{d} \text{ , for each node } i = 1, \ldots, n.$$

- **FL algorithm is determined by its fixed-point operators** $\mathcal{F}^{(1)}, \ldots, \mathcal{F}^{(n)}$, encoding local update rules.[7]

[7] Local updates can be time-varying, i.e., using $\mathcal{F}^{(i,k)}$ varying with k.

Challenges of Synchronous FL

Implementing synchronous fixed-point iteration is challenging.

- Devices might have **limited computational resources**.
- Evaluating the local loss may require **data collection**.
- **Message passing is unreliable** over wireless links.
- Devices may **spontaneously join or drop out**.

Modelling Asynchronous Federated Learning

Consider an FL algorithm with fixed-point operators $\mathcal{F}^{(i)}$.

We obtain an asynchronous variant with the update

$$\mathbf{w}^{(i,k+1)} = \begin{cases} \mathcal{F}^{(i)}\left(\mathbf{w}^{(1,s_{i,1}^{(k)})}, \ldots, \mathbf{w}^{(n,s_{i,n}^{(k)})}\right) & \text{for } k \in T^{(i)} \\ \mathbf{w}^{(i,k)} & \text{otherwise.} \end{cases}$$

- The iteration index k enumerates update events.
- Node i runs local update only during events $k \in T^{(i)}$.
- **Delay** $k - s_{i,i'}^{(k)}$ from i' to i during update event $k \in T^{(i)}$.

Example: Asynchronous FedGD

Consider FedGD with fixed-point operators

$$\mathcal{F}^{(i)}\left(\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(n)}\right) = \mathbf{w}^{(i)} - \eta\left[\nabla L_i\left(\mathbf{w}^{(i)}\right) + 2\alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}\left(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right)\right].$$

Asynchronous variant of FedGD is then

$$\mathbf{w}^{(i,k+1)} = \mathbf{w}^{(i,k)} - \eta\left[\nabla L_i\left(\mathbf{w}^{(i)}\right) + 2\alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}\left(\mathbf{w}^{(i,k)} - \mathbf{w}^{(i',s^{(k)}_{i,i'})}\right)\right],$$

for $k \in T^{(i)}$, and $\mathbf{w}^{(i,k+1)} = \mathbf{w}^{(i,k)}$ otherwise.

Python demo

Totally Asynchronous Algorithms

Consider an asynchronous FL algorithm

$$\mathbf{w}^{(i,k+1)} = \begin{cases} \mathcal{F}^{(i)}\left(\mathbf{w}^{(1,s_{i,1}^{(k)})}, \ldots, \mathbf{w}^{(n,s_{i,n}^{(k)})}\right) & \text{for } k \in T^{(i)} \\ \mathbf{w}^{(i,k)} & \text{otherwise.} \end{cases}$$

We call it **totally asynchronous** if it "works" under the following minimal assumptions:[8]

- The set $T^{(i)}$ is infinite for each $i = 1, \ldots, n$.
- The delayed update times $s_{i,i'}^{(k)}$ are unbounded,

$$\lim_{\substack{k \to \infty \\ k \in T^{(i)}}} s_{i,i'}^{(k)} = \infty.$$

[8] see Ch. 6 of D. Bertsekas, J. Tsitsiklis, "Parallel and Distributed Computation: Numerical Methods," 2015.

Partially Asynchronous Algorithms

Consider some asynchronous FL algorithm

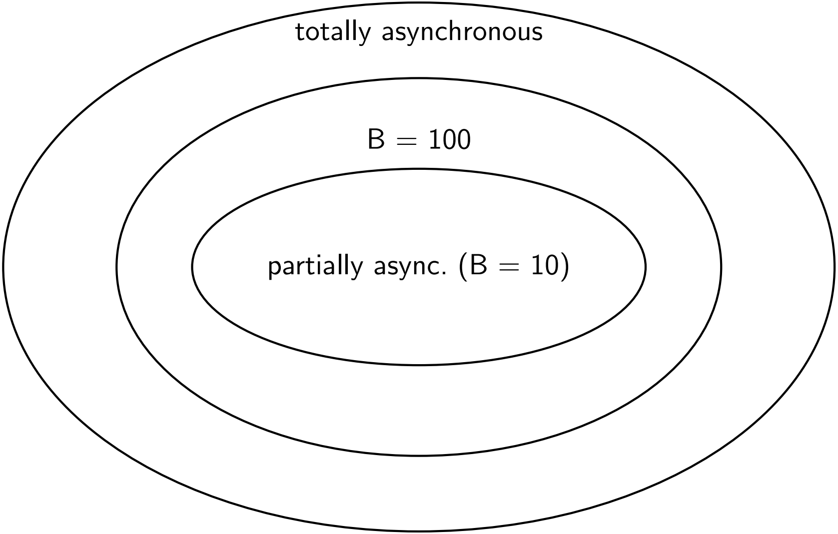
$$\mathbf{w}^{(i,k+1)} = \begin{cases} \mathcal{F}^{(i)}\left(\mathbf{w}^{(1,s_{i,i'}^{(k)})}, \ldots, \mathbf{w}^{(n,s_{i,n}^{(k)})}\right) & \text{for } k \in T^{(i)} \\ \mathbf{w}^{(i,k)} & \text{otherwise.} \end{cases}$$

We call it **partially asynchronous**, with max. delay $B \in \mathbb{N}$, if it works as long as[9]

- $\{k, \ldots, k+B-1\} \cap T^{(i)} \neq \emptyset$ for each $k \in \mathbb{N}$, and
- bounded delay $k - s_{i,i'}^{(k)} \leq B$ for all $k \in T^{(i)}$.

[9] see Ch. 7 of D. Bertsekas, J. Tsitsiklis, "Parallel and Distributed Computation: Numerical Methods," 2015.

Hierarchy of Asynchronous Computers



When Does it Work ?

An asynchronous FL algorithm is fully specified by:

- The fixed-point operators $\mathcal{F}^{(i)}$ for $i = 1, \ldots, n$.
- The update events $T^{(i)}$, for $i = 1, \ldots, n$.
- The delays $k - s_{i,i'}^{(k)}$ for $i, i' \in \mathcal{V}$, $k \in T^{(i)}$.

What are sufficient conditions on those components such that the resulting algorithm is totally (partially) asynchronous?[10]

[10] H. R. Feyzmahdavian and M. Johansson, "On the convergence rates of asynchronous iterations," Proc. IEEE CDC, 2014

Pseudo-Contractions w.r.t. Block-Maximum Norm

- Stacked local model params. $\mathbf{w} = \mathrm{stack}\{\mathbf{w}^{(i)}\}_{i=1}^{n}$.
- Define the block-maximum norm

$$\|\mathbf{w}\|_{\infty} := \max_{i=1,\ldots,n} \left\|\mathbf{w}^{(i)}\right\|_{i}.$$

FL algo. $\mathcal{F} = \left(\mathcal{F}^{(1)}, \ldots, \mathcal{F}^{(n)}\right)$ is a pseudo-contraction if[11]

$$\|\mathcal{F}\mathbf{w} - \widehat{\mathbf{w}}\|_{\infty} \leq \kappa \|\mathbf{w} - \widehat{\mathbf{w}}\|_{\infty} \tag{1}$$

with fixed point $\widehat{\mathbf{w}} = \mathcal{F}\widehat{\mathbf{w}}$ and some $\kappa \in [0, 1)$.

[11] H. R. Feyzmahdavian and M. Johansson, "Asynchronous Iterations in Optimization: New Sequence Results and Sharper Algorithmic Guarantees," JMLR, 2023

A Convergence Result

Consider FL algo. being a pseudo-contraction with $\kappa < 1$.
Then,

- it is totally asynchronous,[12] and
- for a partially asynchronous setting with max. delay B,[13]

$$\left\| \mathbf{w}^{(k)} - \widehat{\mathbf{w}} \right\|_\infty \leq \kappa^{\frac{k}{2B+1}} \left\| \mathbf{w}^{(0)} - \widehat{\mathbf{w}} \right\|_\infty .$$

$\Rightarrow$ Convergence is faster for smaller κ and smaller B.

How can we ensure this?

---

[12] Thm. 23 in H. R. Feyzmahdavian and M. Johansson, JMLR, 2023.
[13] Thm. 24 in H. R. Feyzmahdavian and M. Johansson, JMLR, 2023.

Wrap Up

- FL alg. as fixed-point iterations $\mathbf{w}^{(k+1)} = \mathcal{F}\mathbf{w}^{(k)}$.
- Fixed point of $\mathcal{F}$ is a solution of GTVMin.
- Convergence depends on contraction properties of $\mathcal{F}$.
- Tolerant against asynchronous implementation.

What's Next?

The next module studies some **main flavours of FL**.

These flavours are characterized by specific design choices arising in FL networks and GTVMin.

Further Resources

- **YouTube:** @alexjung111
- **LinkedIn:** Alexander Jung
- **GitHub:** alexjungaalto