Federated Learning

Lecture 1 – Foundations
Escuela de Ciencias Informáticas (ECI) 2026
Universidad de Buenos Aires

Ioannis Chatzigiannakis

Sapienza University of Rome

ECI 2026 • Buenos Aires • Lecture 1 of 5

SAPIENZA
UNIVERSITÀ DI ROMA

Federated Learning

From Theory to Practice to Advanced Topics

- The term first coined by Google researchers in 2016.
  - 2018: about 180 research papers
  - 2019: about 500 research papers
  - 2020: 1000+ research papers
  - 2021: 2500+ research papers
  - 2022: 5000+ research papers
- We have already seen some real-world deployments by companies and researchers
- Several open-source libraries are under development: PySyft, TensorFlow Federated, FATE, Flower, Substra . . .
- FL is highly multidisciplinary: it involves machine learning, numerical optimization, privacy & security, networks, systems, hardware . . .

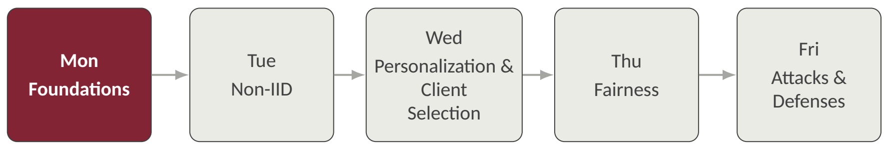
Objectives of the course

From Theory to Practice to Advanced Topics

1. Understand the FL computation model and its motivations
2. Master IID vs. non-IID notions and quantify their effect on performance and stability.
3. From Theory to Practice:
   — Use Flower to transform a centralized (PyTorch/TensorFlow) training routine into a federated one.
   — Generate and control heterogeneity with FedArtML for reproducible experimentation.
   — Implement and compare optimization/aggregation algorithms (FedAvg, FedProx, FedOpt) under diverse scenarios.
4. Advanced Topics:
   — Client selection, Personalization and Fairness
   — Recognize relevant cyber-attacks (data/model poisoning, backdoors, Byzantine behavior, inference) and apply practical defenses (Krum/median/trimmed-mean, sanitization, DP, secure aggregation).

This week schedule

Where today's lecture fits



Running thread. One image-classification pipeline, progressively partitioned → federated → personalized → fairness-audited → attacked/defended. Every afternoon lab extends the same codebase (Flower + PyTorch).

Today. Why FL exists, how a training round actually works, the main deployment variants, what can leak from "just weights," and how differential privacy bounds that leakage.

About the lecturer
Ioannis Chatzigiannakis

Full Professor, Dept. of Computer, Control and Management Engineering (DIAG), Sapienza University of Rome

Education.

- Ph.D. in Computer Science, University of Patras (2003)
- B.Eng. in Computer Systems, University of Kent (1997)

Research. EU co-funded R&D projects on:

- Pervasive systems and the Internet of Things
- Distributed computing and the Future Internet
- Applications of data science in intelligent environments

Leads two research groups at Sapienza:

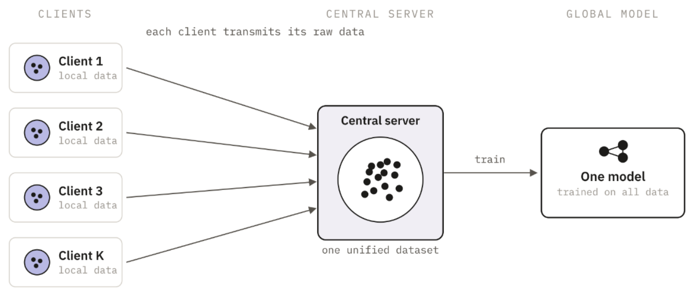
- IDSCC – Immersive & Data-Driven Sustainable Cities and Communities
- Computer Networks and Pervasive Systems

Contact
ichatz@diag.uniroma1.it

research.uniroma1.it/idscc

research.uniroma1.it/cnps

Centralized Learning
Pool all data in one place



- Standard AI/ML assumes all data can be gathered in one place.
- Centralized processing of a single, tightly integrated dataset.
- A single model is trained once, on the full aggregated dataset.

Benefits of Centralized Learning
Why it was, and still is, the default paradigm

What are the benefits of moving all data to one place?

Benefits of Centralized Learning
Why it was, and still is, the default paradigm

What are the benefits of moving all data to one place?

1. **Full statistical control.** The trainer sees the entire joint distribution at once, so batches can be shuffled to be i.i.d. - the assumption behind SGD convergence theory. Simpler, more stable optimization.

Benefits of Centralized Learning

Why it was, and still is, the default paradigm

What are the benefits of moving all data to one place?

1. **Full statistical control.** The trainer sees the entire joint distribution at once, so batches can be shuffled to be i.i.d. - the assumption behind SGD convergence theory. Simpler, more stable optimization.
2. **Straightforward debugging and reproducibility.** Every example is inspectable; ablations and hyperparameter sweeps behave predictably.

Benefits of Centralized Learning

Why it was, and still is, the default paradigm

What are the benefits of moving all data to one place?

1. **Full statistical control.** The trainer sees the entire joint distribution at once, so batches can be shuffled to be i.i.d. - the assumption behind SGD convergence theory. Simpler, more stable optimization.
2. **Straightforward debugging and reproducibility.** Every example is inspectable; ablations and hyperparameter sweeps behave predictably.
3. **Scales cleanly with compute.** Given a unified dataset, throughput is limited only by hardware (GPUs/TPUs in one cluster), not by network heterogeneity across clients.

Benefits of Centralized Learning

Why it was, and still is, the default paradigm

What are the benefits of moving all data to one place?

1. **Full statistical control.** The trainer sees the entire joint distribution at once, so batches can be shuffled to be i.i.d. - the assumption behind SGD convergence theory. Simpler, more stable optimization.
2. **Straightforward debugging and reproducibility.** Every example is inspectable; ablations and hyperparameter sweeps behave predictably.
3. **Scales cleanly with compute.** Given a unified dataset, throughput is limited only by hardware (GPUs/TPUs in one cluster), not by network heterogeneity across clients.
4. **Mature tooling.** The entire ML ecosystem (PyTorch, TensorFlow, distributed data-parallel training) was built around this assumption first.

Challenges of centralized ML

What worries you most as a practitioner?

What can go wrong when we insist on moving all data to one place?

Challenges of centralized ML

What worries you most as a practitioner?

What can go wrong when we insist on moving all data to one place?

1. Privacy/security is not guaranteed once data leaves the client

Challenges of centralized ML

What worries you most as a practitioner?

What can go wrong when we insist on moving all data to one place?

1. Privacy/security is not guaranteed once data leaves the client
2. Regulation (GDPR, CCPA, . . .) restricts moving data across institutions or borders

Challenges of centralized ML

What worries you most as a practitioner?

What can go wrong when we insist on moving all data to one place?

1. Privacy/security is not guaranteed once data leaves the client
2. Regulation (GDPR, CCPA, . . .) restricts moving data across institutions or borders
3. Bandwidth and storage cannot keep up as data volume grows

Challenges of centralized ML

What worries you most as a practitioner?

What can go wrong when we insist on moving all data to one place?

1. Privacy/security is not guaranteed once data leaves the client
2. Regulation (GDPR, CCPA, . . .) restricts moving data across institutions or borders
3. Bandwidth and storage cannot keep up as data volume grows
4. The server itself becomes a bottleneck and the single biggest attack target

Challenges of centralized ML

What worries you most as a practitioner?

What can go wrong when we insist on moving all data to one place?

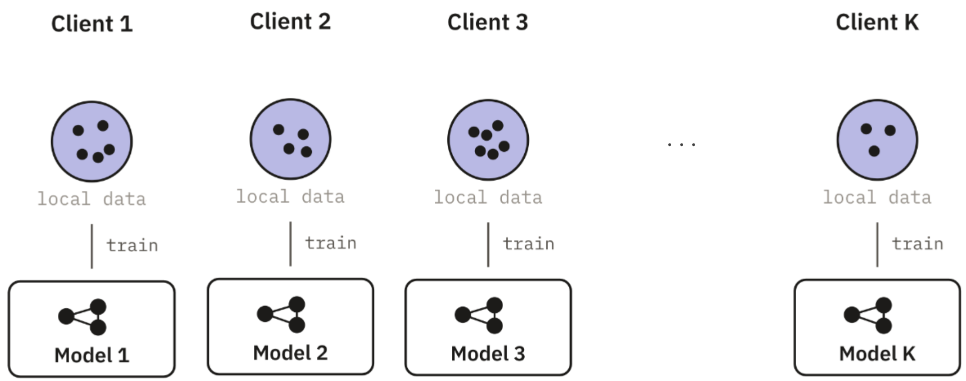
1. Privacy/security is not guaranteed once data leaves the client
2. Regulation (GDPR, CCPA, ...) restricts moving data across institutions or borders
3. Bandwidth and storage cannot keep up as data volume grows
4. The server itself becomes a bottleneck and the single biggest attack target

They are all one problem

1, 2, 3 are symptoms; 4 is the root cause – centralizing data creates a **single point of failure**: one bottleneck, one attack surface, one regulatory exposure. FL is designed to remove exactly this.

How about each party learning on its own?

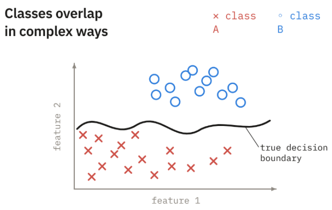
Each one uses its own local data - nothing shared between parties



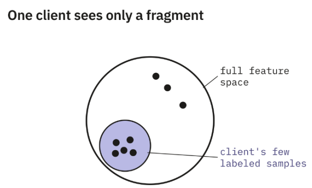
How about each party learning on its own?

What can go wrong?

1. **The local dataset may be too small**
   - Separating the classes takes a rich, intricate boundary - and that needs dense, varied data to learn.
   - Sub-par predictive performance (e.g., due to overfitting)
   - Non-statistically significant results (e.g., medical studies)

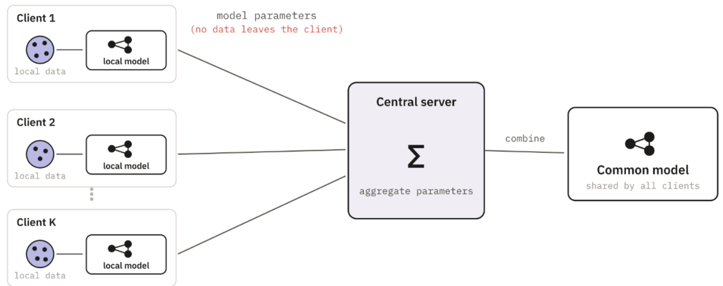


2. **The local dataset may be biased**
   - A handful of points covers only a tiny slice of the space - too few, and not statistically representative.
   - Not representative of the target distribution



From Centralized to Federateds Learning

Federated learning shares models, not data

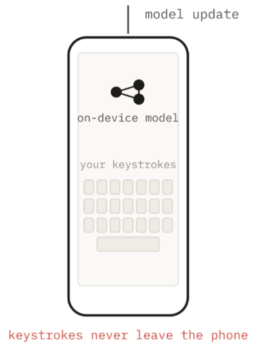


- Clients train locally and send only their model parameters – raw data never moves.
- The server combines them into one common model.
- Process is repeated until convergence.

Aggregation Functions
Federated learning shares models, not data

- Aggregation functions represent methods to combine model weights from locally trained models into one global model, ensuring the model learns effectively from all participants
- Aggregation ensures balanced learning from diverse data, influences convergence speed and model quality, and secures against malicious model updates
- Several aggregation functions have been proposed in the relevant bibliography that consider
  - data distribution (IID or non-IID),
  - scale (number of clients),
  - robustness against adversarial updates,
  - performance (convergence speed and accuracy),
  - …

Gboard trains a shared model on millions of phones
A concrete example of “moving the model instead of the data”



- Millions of Android keyboards, one shared model
  Gboard’s next-word prediction is trained collaboratively across countless phones - each contributing to a single global language model.
- Ship updates, not data
  Only model updates leave the device — orders of magnitude smaller than the raw keystroke data behind them.
- Raw data stays private
  No keystroke ever leaves the phone; no client ever exposes its raw typing.

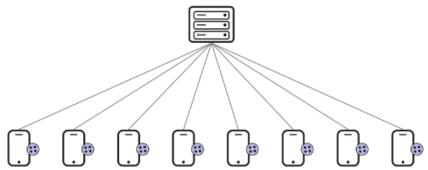
Where is federated learning used today?
From mobile keyboards to hospitals

Domain	System / study	Reference
Voice assistants	Apple Siri, on-device personalization without raw audio upload	Apple ML Research, 2019
Healthcare	Multi-institutional brain tumor segmentation across hospitals	Sheller et al., *Scientific Reports*, 2020
Finance	Cross-institution fraud/risk modeling without sharing raw records	Yang et al., *ACM TIST*, 2019 (WeBank)
Telecom / IoT security	FL-based intrusion detection for connected devices	Hernandez-Ramos et al., arXiv:2308.09522, 2023
Connected vehicles	Federated perception/prediction models across fleets	Elbir et al., *IEEE Comm. Surveys & Tutorials*, 2022

Notice the pattern: in every case, the data is sensitive, regulated, or too voluminous to move – exactly the challenges on the next slide.

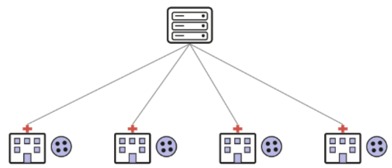
Cross-device vs cross-silo federated learning
Not all FL deployments look alike

Cross-device FL



- Massive number of parties (up to 10^{10})
- Small dataset per party (could be size 1)
- Limited availability and reliability
- Some parties may be malicious

Cross-silo FL

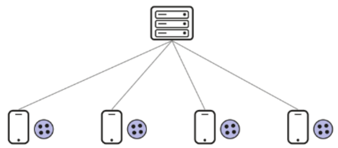


- 2-100 parties
- Medium to large dataset per party
- Reliable parties, almost always available
- Parties are typically honest

Orchestrated vs Fully Decentralized federated learning

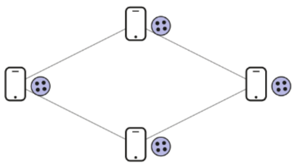
Not all FL deployments look alike

Server-orchestrated FL



- Server-client communication
- Global coordination, global aggregation
- Server is a single point of failure and may become a bottleneck

Fully decentralized FL

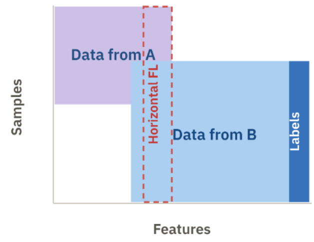


- Device-to-device communication
- No global coordination, local aggregation
- Scales to a large number of devices

Horizontal vs Vertical federated learning

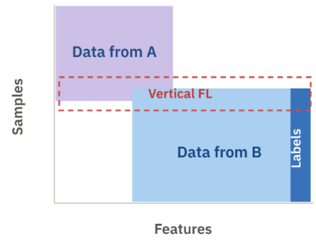
Not all FL deployments look alike

Horizontal FL



- Splits the data by samples
- Same features, different samples
- Parties hold different rows of a shared feature space.

Vertical FL



- Splits the same samples by features
- Same samples, different features
- Parties hold different columns for a shared set of entities.

Basic notation

Notation that will be used throughout the week

We consider a set of K parties (clients)

Each party k holds a dataset D_k of n_k points, i.e., $n_k = |D_k|$.

Let D_i be the set of elements held by client i, of n_k tuples:

$$D_i = \{(x_1, y_1), \dots, (x_{n_k}, y_{n_k})\}$$

where $x_i = [(x_i)_1, \dots, (x_i)_m]$ is the feature vector of m features and $y_i \in \{1, \dots, \ell\}$ the (true) label of the i-th sample.

Keep this in mind

The datasets $D_1, \dots, D_K$ are stored only on the clients.

Formalizing the centralized dataset

Notation that will be used throughout the week

A **centralized dataset** D is formed by merging the D_i of each client i, removing duplicate samples:

$$D = \bigcup_{i=1}^{K} D_i \setminus \{\text{duplicates}\}$$

Equivalently, treating each D_i as a multiset and D as a set:

$$D = \{(x, y) \; : \; (x, y) \in D_i \text{ for some } i \in \{1, \dots, K\}\}$$

The same sample (x_i, y_i) may legitimately appear at more than one client (e.g. a public benchmark item, or overlapping sensor coverage) - clients are *not* required to hold disjoint partitions of D.

Keep this in mind

The datasets $D_1, \dots, D_K$ are transmitted to the centralized server.

The federated learning computation model
Formal setup

We focus on non-convex neural network objectives.

We assume that the IID – the samples are distributed over the clients uniformly at random.

Federated learning solves

$$\min_w \; F(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w), \qquad F_k(w) = \frac{1}{n_k} \sum_{i \in D_k} \ell(w; x_i, y_i)$$

where $n_k = |D_k|$ and $n = \sum_k n_k$.

For a machine learning problem, ℓ is the loss of the prediction on sample (x_i, y_i) made with model parameters w.

The performance goals of federated learning
The high-level objectives

Today's question

We know *what* we want to minimize. How do we actually minimize $F(w)$ without ever pooling $D_1, \ldots, D_K$?

- Reveal the least amount of information regarding the local datasets to the server.
- Ensure the model learns effectively from all participants - balanced learning from diverse data.
- Minimize convergence speed while keeping model quality at par with centralized training.
- Scale to large number of clients.

The reference point: ordinary, large-batch SGD
Before federating anything at all

Ignore the FL constraint for a moment – simply pool D on one machine. The ordinary optimization problem, solved with minibatch SGD:

$$\min_w \; F(w), \qquad w_{t+1} \leftarrow w_t - \eta \, \nabla \ell(w_t; \mathcal{B}), \qquad \mathcal{B} \subset D, \; |\mathcal{B}| = B$$

where $\mathcal{B}$ is a minibatch of B examples drawn uniformly from the *pooled* data – not from a single client's D_k.

Why include it at all?

It never appears in a real deployment – pooling D is exactly what FL forbids. We keep it only as the **"SGD (centralized)"** row: an apples-to-apples yardstick to evaluate the performance of different aggregation functions and how many steps are needed to reach the *same* accuracy without ever pooling data.

A naive federated baseline – FedSGD
McMahan et al., *AISTATS* 2017

Idea. Reuse ordinary (large-batch) SGD, one step per communication round.

On each round t, clients compute the gradient of their local objective at the current model:

$$g_k = \nabla F_k(w_t)$$

The server aggregates and takes one step:

$$w_{t+1} \leftarrow w_t - \eta \sum_k \frac{n_k}{n} g_k$$

Equivalent client-centric view. $\forall k : w_t^k \leftarrow w_t - \eta g_k$, then $w_{t+1} \leftarrow \sum_k \frac{n_k}{n} w_t^k$.

We call this baseline **FederatedSGD (FedSGD)**.

The FedAvg aggregation function

McMahan et al., *AISTATS* 2017

On each round t, select a C-fraction of clients and have each compute the gradient of its local objective at the current model.

FedAvg round (McMahan et al., *AISTATS* 2017):

1. **Broadcast** the current global model w_t to a sampled subset of clients
2. **Local update**: each client runs several epochs of SGD on D_k starting from w_t
3. **Upload** the resulting local weights w_t^k (not the data)
4. **Aggregate**: $w_{t+1} = \sum_k \frac{n_k}{n} w_t^k$

Reading C

C controls the *global* batch size: $C{=}1$ is full-batch (non-stochastic) gradient descent.

Fine-tuning the performance of FedAvg

McMahan et al., *AISTATS* 2017

Once the update is written client-then-average, nothing stops each client from taking several local SGD steps before the server averages:

$$w^k \leftarrow w^k - \eta \nabla \ell(w^k; b), \qquad \text{repeated over local batches } b$$

Three hyperparameters control the amount of local computation:

- C – fraction of clients that compute each round
- E – number of local epochs (passes over D_k) per round
- B – local minibatch size ($B = \infty$ means "treat all of D_k as one batch")

A client with n_k examples then takes $u_k = E\, n_k / B$ local update steps per round.

The special case

$E{=}1,\ B{=}\infty$ collapses **exactly** to FedSGD. Everything else – $E{>}1$ and/or $B{<}\infty$ – is what the literature calls **FedAvg**, a.k.a. *Local SGD*.

The FedAvg algorithm – server side

McMahan et al., *AISTATS* 2017, Algorithm 1

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**

$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$

$$w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{\sum_{j \in S_t} n_j} w_{t+1}^k$$

A subtlety worth knowing

Normalize by $\sum_{j \in S_t} n_j$ (the data held by *sampled* clients), not by the global n – early versions of the paper had an erratum here. With $C{=}1$ the two coincide.

The FedAvg algorithm – client side

McMahan et al., *AISTATS* 2017, Algorithm 1

ClientUpdate(k, w): *// run on client k, never leaves the device*

$\mathcal{B} \leftarrow$ (split D_k into batches of size B)

for each local epoch $i = 1$ to E **do**

for batch $b \in \mathcal{B}$ **do**

$w \leftarrow w - \eta \nabla \ell(w; b)$

return w to server

Only w ever crosses the network – in both directions. D_k, gradients on raw examples, and intermediate activations stay on the client.

Aka "Local SGD": with sampling rate C and $L{=}E$ local steps, $L{=}1,\ C{=}1$ recovers classic parallel SGD; $L{>}1$ trades communication for local computation.

Convergence guarantees: the IID case

Why local steps are safe when data is IID

Under IID data, each D_k is (in expectation) a sample from the same distribution as everyone else's, so local gradients are *unbiased* estimates of the global one:

$$\mathbb{E}[\nabla F_k(w)] = \nabla F(w)$$

Taking E local SGD steps before averaging is then just *more stochastic steps toward the same minimizer* – it adds variance, not bias, which periodic averaging tames.

Formal results.

- *Stich, ICLR 2019:* local SGD matches centralized minibatch-SGD rates for (strongly) convex F, communicating far less.
- *Woodworth et al., ICML 2020:* pins down exactly when local SGD provably beats minibatch SGD – the many-local-steps / IID regime of Table 4.

Bottom line (IID)

More local computation is close to a free lunch: it cuts communication without hurting accuracy, exactly as Tables 3–4 showed.

Why does averaging *weights* even work?

A non-convex loss surface should not allow this – but it does

A legitimate worry: ℓ is non-convex (neural nets), so averaging two models' parameters could land on an arbitrarily bad point of the loss surface.

McMahan et al. test this directly by training two models w, w' on disjoint MNIST subsets and plotting the loss of $\theta w + (1-\theta)w'$ for $\theta \in [-0.2, 1.2]$:

- **Independent random initialization:** the average is often *worse* than either parent – averaging fails.
- **Shared initialization:** the average is *better* than either parent, sometimes substantially.

Why FedAvg is allowed to average weights

Every round starts every client from the *same* w_t. This shared starting point keeps the clients in the same basin of the loss landscape, which is precisely why naive parameter averaging behaves well in practice.

The experimental testbed (McMahan et al., 2017)

Tasks and datasets

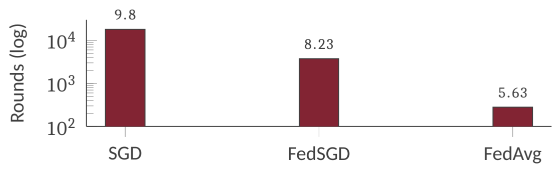
Task	Model	Clients / partition
MNIST digits	2NN (2 hidden layers, 199,210 params); CNN (∼1.6M params)	100 clients, 600 ex. each
CIFAR-10	CNN, $\sim 10^6$ params	100 clients
Shakespeare (next line)	Character LSTM	1,146 clients, one per speaking role

IID MNIST partitions: shuffle, then split into 100 clients of 600 examples each.

Headline result: communication is the bottleneck, and FedAvg fixes it

Rounds to reach a target test accuracy, CIFAR-10 (C=0.1, FedAvg: E=5, B=50)

	80% acc.	82% acc.	85% acc.
SGD (centralized)	18 000	31 000	99 000
FedSGD	3 750 (4.8×)	6 600 (4.7×)	N/A
FedAvg	280 (**64.3**×)	630 (**49.2**×)	2 000 (**49.5**×)



Tuning knob 1: client fraction C

Rounds to reach target accuracy, MNIST (2NN: 97%, $E{=}1$; CNN: 99%, $E{=}5$)

C	$B{=}\infty$	$B{=}10$
0.0	1455	316
0.1	1474 (1.0×)	87 (3.6×)
0.5	–	75 (4.2×)
1.0	–	70 (4.5×)

- With $B{=}\infty$, raising C barely helps – more clients, but each still takes one full-batch step.
- With small B (more stochasticity per client), raising C helps a lot.
- Practical takeaway (used for the rest of the paper): $C{=}0.1$ is a good efficiency/convergence trade-off.

Tuning knob 2: local computation E, B

MNIST 2NN, target 97% acc., $C{=}0.1$ fixed – $u = E\,n_k/B$ local steps/round

	E	B	u	IID
FedSGD	1	∞	1	1468
FedAvg	10	∞	10	156 (9.4×)
FedAvg	20	∞	20	92 (16.0×)
FedAvg	10	10	600	34 (**43.2×**)

- More local steps u (bigger E, smaller B) $\Rightarrow$ dramatically fewer communication rounds.

The catch: two limits to “more local computation”

Why this motivates tomorrow’s material

1. **Non-IID data blunts the gains you just saw.** Averaging models that each drifted toward a *different* local optimum is less effective than averaging models trained on similar data – quantifying and taming this drift is tomorrow’s lecture.

The catch: two limits to “more local computation”

Why this motivates tomorrow’s material

1. **Non-IID data blunts the gains you just saw.** Averaging models that each drifted toward a *different* local optimum is less effective than averaging models trained on similar data – quantifying and taming this drift is tomorrow’s lecture.
2. **Very large E can plateau or diverge.** On the Shakespeare LSTM task, pushing local epochs too high (fixed $B{=}10$, $C{=}0.1$) makes FedAvg stall rather than improve; interestingly, the CNN on MNIST barely degrades with large E, but the large-scale next-word LSTM *performs better with* $E{=}1$ *than* $E{=}5$.

Client drift, informally

As $E \to \infty$, each ClientUpdate increasingly solves *client k’s own* problem $\min_w F_k(w)$ rather than the global $F(w)$ – the initialization from w_t is the only thing tying it back to the rest of the federation.

Convergence: putting it all together

Recall the guarantees from a few slides back

- **IID data:** provably converges, with rates matching centralized SGD *(Stich, ICLR 2019; Woodworth et al., ICML 2020)* – see our earlier deep dive.
- **Non-IID data:** provably harder – under bounded gradient dissimilarity, FedAvg's rate is strictly worse than parallel SGD's *(Karimireddy et al., ICML 2020)*.
- **What FedAvg does *not* handle by design:** systems heterogeneity (it assumes every sampled client finishes the same E epochs) and any adaptivity to how skewed a client's local distribution is.

Where the course goes next

Tomorrow: quantifying non-IID-ness (Hellinger, Jensen–Shannon, EMD, PSI). Later this week: FedProx, client selection, and algorithms designed specifically to correct for this drift (e.g. SCAFFOLD).

Bonus: FedAvg without a server

For the curious – fully decentralized local SGD

The same idea generalizes beyond a central aggregator. Let $G = (\{1, \ldots, K\}, E)$ be a connected graph of clients, and $W \in [0,1]^{K\times K}$ a symmetric, doubly-stochastic mixing matrix with $W_{kl} = 0$ unless $\{k, l\} \in E$.

Each client k alternates a local SGD step with a weighted average over its graph neighbors:

$$w_k^{(t+\frac{1}{2})} \leftarrow w_k^{(t)} - \eta \nabla \ell(w_k^{(t)}; b), \qquad w_k^{(t+1)} \leftarrow \sum_l W_{kl}\, w_l^{(t+\frac{1}{2})}$$

- Doing several local steps between mixing rounds is equivalent to choosing $W = I$ on some rounds – the same E idea as FedAvg.
- Convergence speed now depends on how well-connected the graph G is.

Table of Contents

1 Hands-on session

Practical Data
Introduction
Formal setup
Baseline
FedAvg
Theory behind convergence

▶ Hands-on session

▶ Recap

This afternoon's hands-on session

From slides to code

Environment. Flower + PyTorch, single running example (image classification) that will be reused every day this week.

Steps.

1. Run the centralized PyTorch baseline (no federation yet) – record accuracy
2. Run experiments on Flower
3. Wrap it as a 1-client, then K-client Flower FedAvg simulation – confirm it matches the centralized baseline under IID partitioning

Table of Contents
2 Recap

Practical Data
Introduction
Formal setup
Baseline
FedAvg
Theory behind convergence

▶ Hands-on session

▶ Recap

Recap: Take-home messages
2 Recap

1. **FL trades data movement for model movement**: clients compute locally, the server only ever sees weights – but that shifts the deployment cost, it does not automatically guarantee privacy
2. **FL has many faces**: cross-device vs. cross-silo, hierarchical vs. vertical – the right architecture depends on client population and trust model
3. **FedAvg is FedSGD plus local computation.** Setting $E=1$, $B=\infty$ recovers classic federated (large-batch) SGD; $E>1$ and/or $B<\infty$ trade extra client-side computation for far fewer communication rounds.
4. **The speedups are real but conditional.** They are large under IID data and shrink – sometimes sharply – as local computation is pushed too far.
5. **Only w ever crosses the network**, in the broadcast and in the upload – a necessary but not sufficient condition for privacy, as we will revisit this week.

What comes next
2 Recap

Tomorrow (Lecture 2). What happens when client data are *not* alike: number of observations between clients differ (data skew) or the type of observations differ (label skew).

How to measure non-IIDness? – metrics (Hellinger, Jensen–Shannon, Earth Mover's Distance)

References
2 Recap

- McMahan, H. B. et al. *Communication-Efficient Learning of Deep Networks from Decentralized Data*. AISTATS 2017.
- Stich, S. U. *Local SGD Converges Fast and Communicates Little*. ICLR 2019.
- Woodworth, B. et al. *Is Local SGD Better than Minibatch SGD?* ICML 2020.
- Karimireddy, S. P. et al. *SCAFFOLD: Stochastic Controlled Averaging for Federated Learning*. ICML 2020.
- Li, T. et al. *Federated Optimization in Heterogeneous Networks* (FedProx). MLSys 2020.
- Kairouz, P. et al. *Advances and Open Problems in Federated Learning*. Foundations and Trends in ML, 2021.

Thank you

See you tomorrow !

ichatz@diag.uniroma1.it