Federated Learning

ECE 273 - Convex Optimization, Spring 2021

Jacob Pollard - June 10, 2021

Motivation

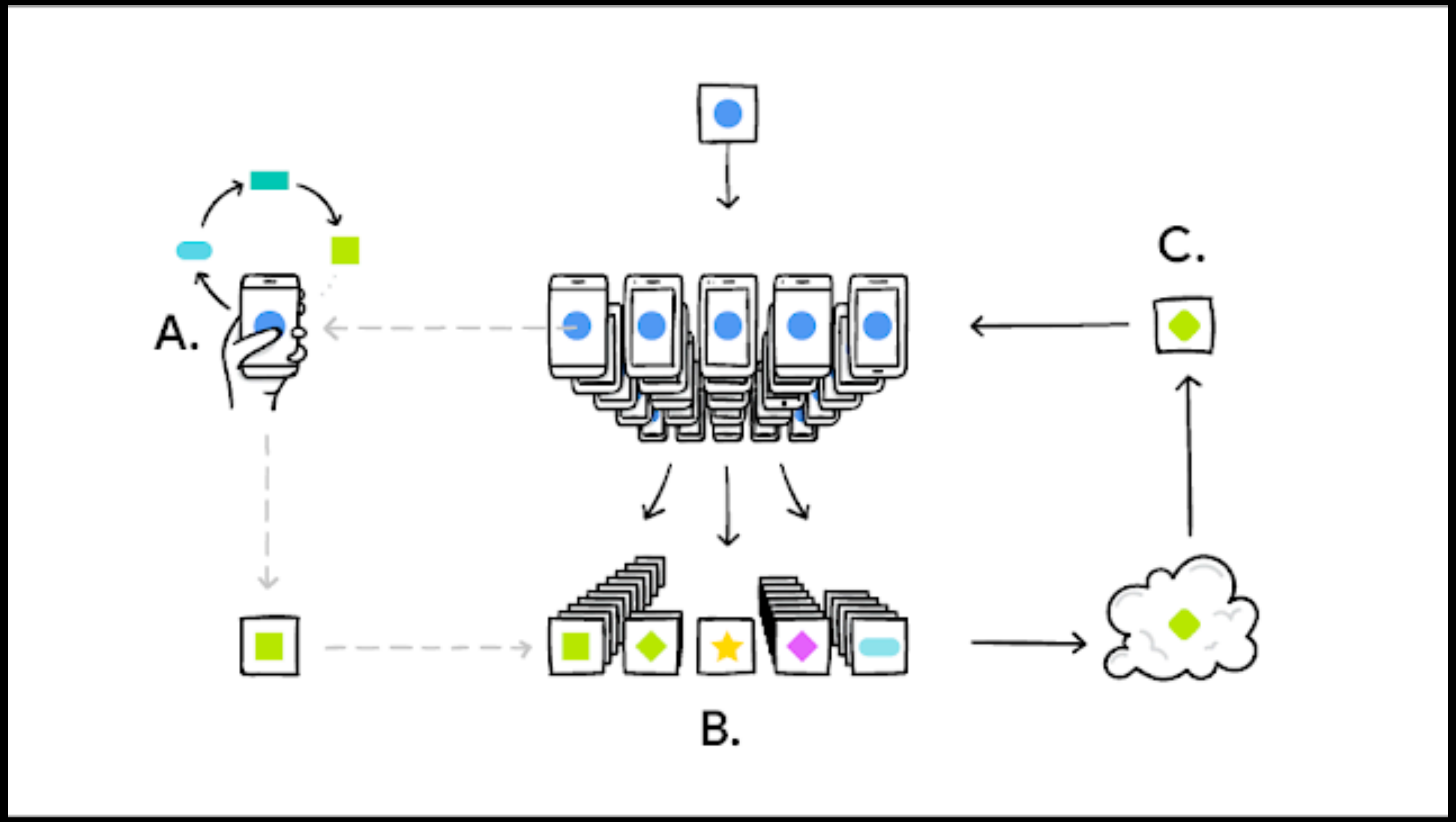
Why Federated Learning?

- Privacy is an increasingly prominent concern in training machine learning models, and sometimes is legally required (HIPPA)
- Transmission of information over certain channels is expensive
- Utilize local computational resources

Federated Learning

Overview

- Similar to a Distributed system, but more general
- Common assumptions in ML do not necessarily hold:
  - Data may not be IID
  - Compute nodes may be highly heterogeneous

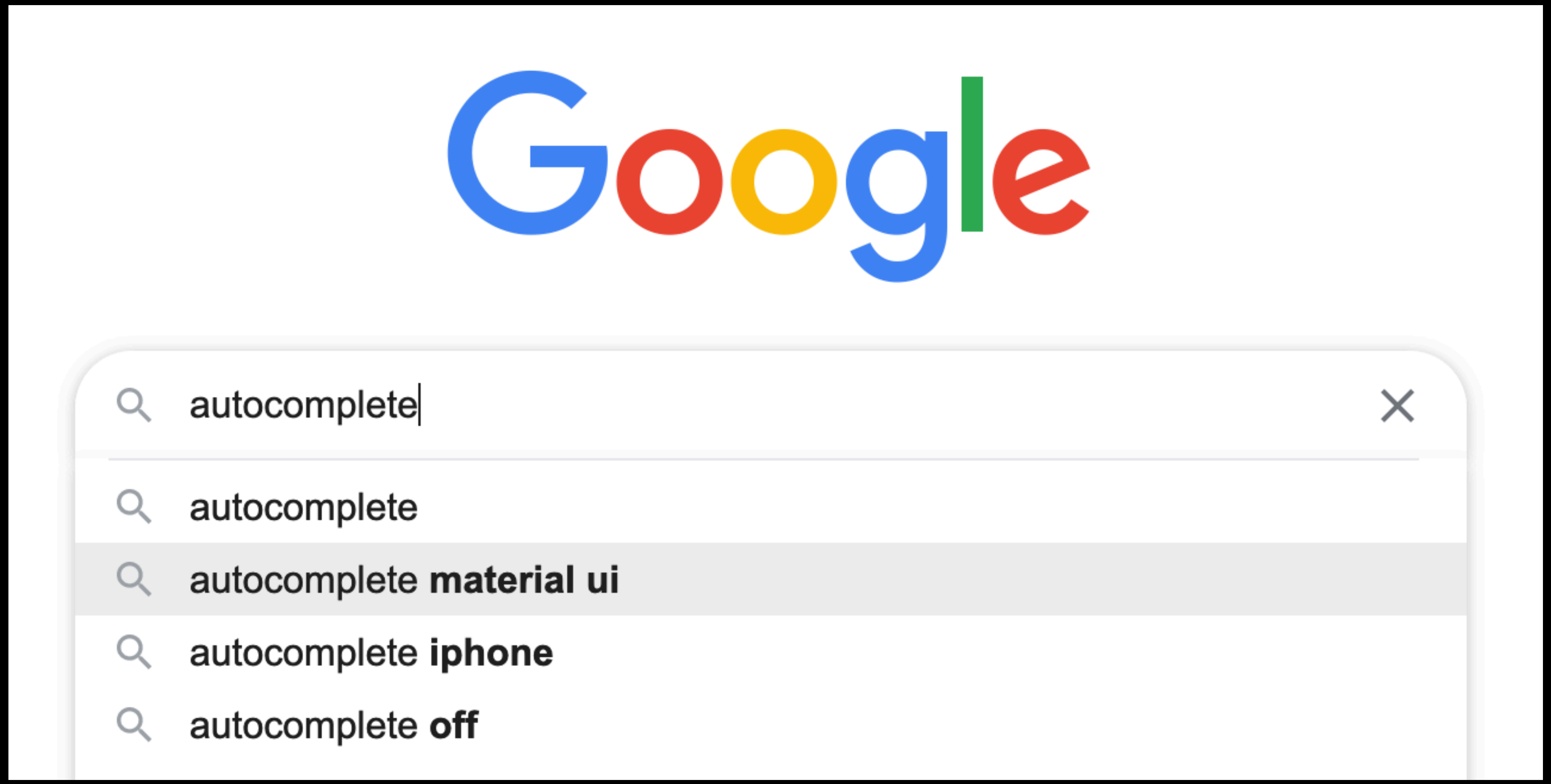


https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

Federated Learning

An example: autocomplete

- Autocomplete on mobile phones
  - Text collected in private messages, internet browsing, etc.
  - Information collected across the US
- Clearly, the standard statistical and computational assumptions do not hold.
- Many more use cases exist, Ex. Medical data



Federated Learning

The prototypical algorithm

- Dataset located on remote workers, not visible to master
- Worker weights can be used to tune less and more important workers
- The average of each worker's minibatch gradient is used for global update

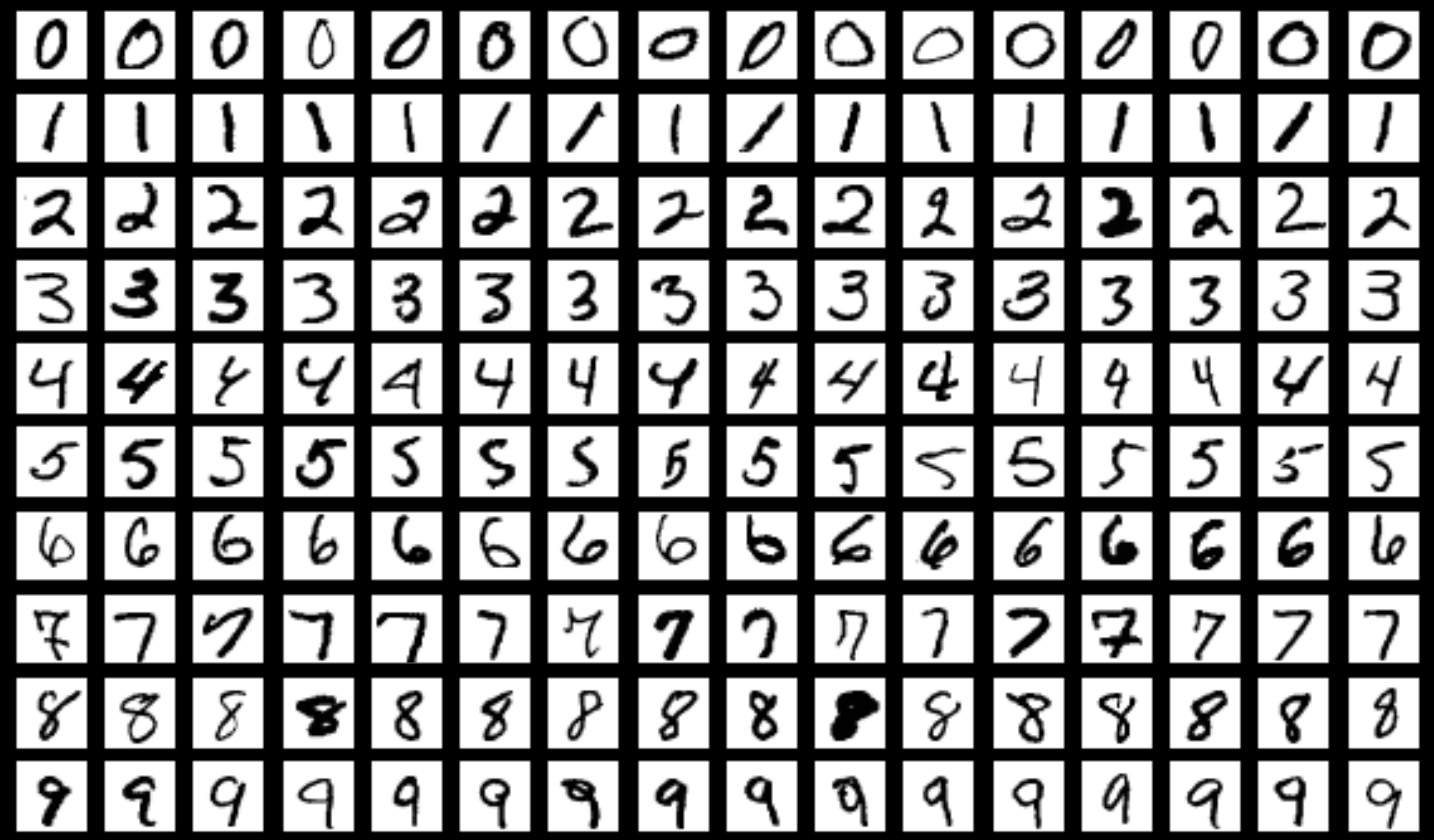
Algorithm 1: Federated SGD

Initialize weight: $w = w_0$;
Worker weights: $p_1, ..., p_k$;
Worker data: $\mathcal{D}_1, ..., \mathcal{D}_k$;
Learning rate α;
while *not converged* **do**
  Master broadcast w_t to all workers;
  foreach *worker* $i = 1, ..., k$ **do**
    $w^i_{t+1} := w_t - \alpha \nabla \mathcal{L}(w_t; \mathcal{D}_i)$;
    Worker broadcast w^i_t master;
  end
  Master calculate new weight $w_{k+1} := \sum_{i=1}^{k} p_i w^i_{t+1}$;
end

Problem Setup

A simple problem

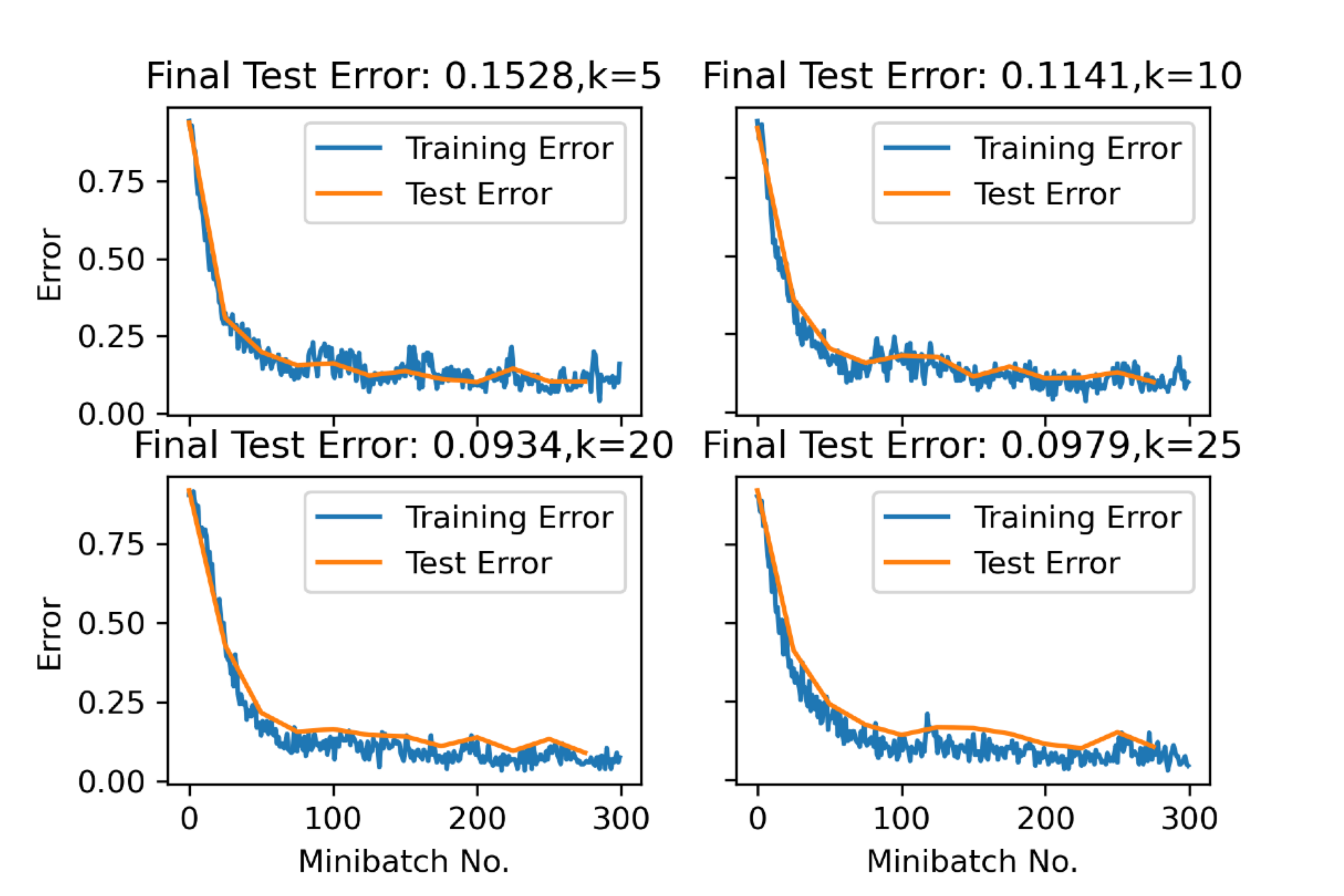
- Train a simple Convolutional Neural Network on the MNIST dataset
- The simple setup makes running multiple experiments much more feasible

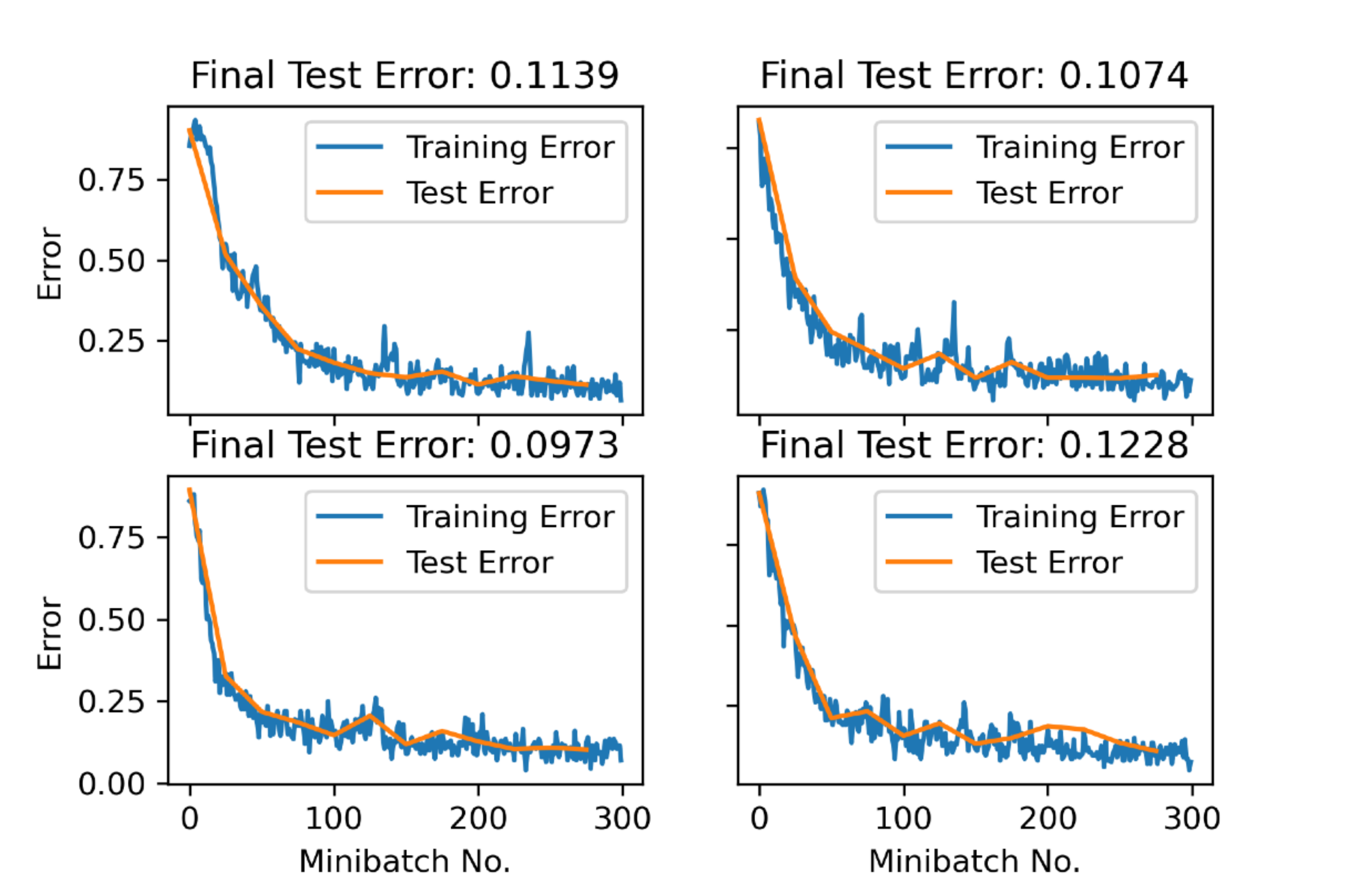


Comparing Centralized to Federated SGD

Federated at top, Centralized at bottom, k=Num Workers

- At first, they appear the same
    - However, once we realize the central node and worker nodes each have equal minibatch size, we realize this is not the case
- The Federated setup with k workers requires k times as much computation





Learning with malicious users

Corrupt Data

- Let's imagine a scenario where some workers nodes send malicious or otherwise bad updates to the master
- Examples
  - Text completion (some users often use foul language)
  - Malicious intent

Learning with malicious users

Corrupt Data

- Experimental setup:
  - Same learning problem as before (MNIST classification)
  - Now, some percentage of users say α send corrupted update
    - To mimic this, these users will train on datas with label 9-y, where y is the true label (an integer on [0,…,9]
  - To mitigate this, we will try to take taking the coordinate wise median, as well as the trimmed mean and see if performance improves.

http://proceedings.mlr.press/v80/yin18a/yin18a.pdf

Learning with malicious users

Corrupt Data

- Sometimes the raw uncorrected model achieves the best error rate, though this is by an extremely thin margin when it does happen
- Overall the median is the best performer
- While the improvements are modest, using median is simple and sensible

Num Workers: α	10	20	25
.05	X	.1272	**.1081**
.1	.1327	.1467	.1212
.2	.1845	**.1584**	.1906

Table 1: Test Error results with malicious users, using the regular mean update rule

Num Workers: α	10	20	25
.05	X	.1447	.1486
.1	**.0996**	**.1162**	.1754
.2	.1563	.1609	.1353

Table 2: Test Error results with malicious users, using coordinate-wise trimmed mean, $\alpha = \beta$

Num Workers: α	10	20	25
.05	X	**.0957**	.1091
.1	.1215	.2027	**.0998**
.2	**.1344**	.17	**.1209**

Table 3: Test Error results with malicious users, using coordinate-wise median

http://proceedings.mlr.press/v80/yin18a/yin18a.pdf

Learning with malicious users

Snooping

- Actual **training data** can be inferred from model parameter updates
- This can be achieved with knowledge only of the model state, and gradient update at a single step
- Only caveat is that batch size needs to be 1

Algorithm 1 Deep Leakage from Gradients.

Input: $F(\mathbf{x}; W)$: Differentiable machine learning model; W: parameter weights; ∇W: gradients calculated by training data

Output: private training data $\mathbf{x}, \mathbf{y}$

procedure DLG($F, W, \nabla W$)
  $\mathbf{x}'_1 \leftarrow \mathcal{N}(0,1)$, $\mathbf{y}'_1 \leftarrow \mathcal{N}(0,1)$ ▷ Initialize dummy inputs and labels.
  for $i \leftarrow 1$ to n **do**
    $\nabla W'_i \leftarrow \partial \ell(F(\mathbf{x}'_i, W_t), \mathbf{y}'_i)/\partial W_t$ ▷ Compute dummy gradients.
    $\mathbb{D}_i \leftarrow ||\nabla W'_i - \nabla W||^2$
    $\mathbf{x}'_{i+1} \leftarrow \mathbf{x}'_i - \eta \nabla_{\mathbf{x}'_i} \mathbb{D}_i$, $\mathbf{y}'_{i+1} \leftarrow \mathbf{y}'_i - \eta \nabla_{\mathbf{y}'_i} \mathbb{D}_i$ ▷ Update data to match gradients.
  end for
  return $\mathbf{x}'_{n+1}, \mathbf{y}'_{n+1}$
end procedure

- https://arxiv.org/pdf/1906.08935.pdf

Learning with malicious users

Snooping

- Able to reconstruct a training sample about 50% of the time
- It appears acquisition is more likely earlier on in the training process