Towards Federated Learning at Scale: System Design


Presenter: Derrick Blakely
April 24, 2019
University of Virginia

https://qdata.github.io/deep2Read/
Table of contents

1. Background and Problem

2. Federated Learning

3. Results

4. Discussion

5. Questions
Background and Problem
Motivation

Figure 1: (A) Personalize model locally, (B) aggregate many users’ model updates, and (C) update global model
Motivation

- ML without centralizing data in a data center
- On-device item ranking
- Content suggestions for keyboards
- Next word prediction
Problem Setting

- Millions of phones with copy of model for inference
- Want to use them for training
- Don’t want to drain phone battery or strain user’s data plan
- Protect privacy
- System needs to be fault tolerant
Parallelizing ML

Figure 2: Model-parallelism

Figure 3: Data-parallelism
Model Synchronization

- Parameter-server
- Decentralized: AllReduce, Gather-Apply-Scatter, and variants
- Any of these can be synchronous, stale synchronous, asynchronous training

Figure 4: Synchronous Parameter-server
Federated Learning
Overview of the System

- FL Server: global model
- FL population: set of devices that periodically compute weight updates
- Push weights to FL server; data never leaves the devices
- Aggregate weights
1. Selection: FL server selects a sub-population to participate
2. Configuration: FL server sends FL plan and checkpoint model
3. Reporting: listen for weight updates and aggregate them
Figure 5: Overview of system protocol
Algorithm 1 FederatedAveraging targeting updates from $K$ clients per round.

Server executes:

initialize $w_0$

for each round $t = 1, 2, \ldots$ do
  Select $1.3K$ eligible clients to compute updates
  Wait for updates from $K$ clients (indexed $1, \ldots, K$)
  $(\Delta^k, n^k) = \text{ClientUpdate}(w)$ from client $k \in [K]$.
  $\bar{w}_t = \sum_k \Delta^k$  // Sum of weighted updates
  $\bar{n}_t = \sum_k n^k$  // Sum of weights
  $\Delta_t = \Delta^k / \bar{n}_t$  // Average update
  $w_{t+1} \leftarrow w_t + \Delta_t$

ClientUpdate($w$):

$\mathcal{B} \leftarrow$ (local data divided into minibatches)
$n \leftarrow |\mathcal{B}|$  // Update weight
$w_{\text{init}} \leftarrow w$

for batch $b \in \mathcal{B}$ do
  $w \leftarrow w - \eta \nabla \ell(w; b)$
  $\Delta \leftarrow n \cdot (w - w_{\text{init}})$  // Weighted update
  // Note $\Delta$ is more amenable to compression than $w$
return $(\Delta, n)$ to server
1. Manage the number of devices participating in training
2. Adjust to avoid “thundering herd” problem
Figure 6: Device software architecture
Device-Side

- Example store (e.g., SQLlite DB)
- FL runtime
- Job invocation: FL runtime contacts FL server
- Task execution: FL plan received from FL server
- Reporting: sends updates to FL server
- Multi-tenancy: device can be part of many FL population
Attestation

- Want anonymous participation
- Exclude authentication
- Risk of poisoned data from extraneous actors
- Android remote attestation mechanism
- Need to circumvent content farms
Server-Side: Actor Design Pattern

- Instead of threads, use “actor” processes as concurrent primitives
- Actors don’t share data/state except by message passing
- Actors can create other actors
- Allows for concurrency to scale on the fly
- Fairly fault-tolerant
Secure Aggregation

- Uses multi-party computation protocol from [2]
- Key exchange protocol based on Shamir’s Secret Sharing [3]
- Server can aggregate encrypted weight updates, but can’t decrypt weights from individual devices
- $O(n^2)$ for $n$ protocol participants
- Requires devices synchronize, hence they use synchronized learning
Figure 7: Workflow for training in the FL framework
Results
Results in Production

- Used for several Google apps with around 10 million devices
- Up to 10K devices participate at once
- 6-10% of devices drop out due to errors, network failures, or changes in eligibility
- Federated training time is 7x slower
Figure 8: Connected devices and completion rate. Rounds typically completed at night, not during the day.
Discussion
Pros

• Built-in mechanisms to protect privacy
• Very interesting application of multiparty computation
• Robust to device dropout and other faults
• Modernizing the parameter-server model
• Training on data not available in data center
Cons

- Training time is 7 times slower than in-data center training
- Federated Averaging can’t handle more than a few hundred devices in parallel
- Example store could have very old data
- Doesn’t distinguish between FL tasks the user actually uses
- Manipulable by content farms
- Doesn’t reduce energy usage with quantization or compression
- Data distribution can vary between users and regions
- Secure aggregation is $O(n^2)$ for $n$ devices
Lessons Learned

- Bring the model to the data; not the data to the model
- Don’t need to violate privacy to train good models
- Federated computation is bigger than ML
- MapReduce and centralized ML are analogous tasks
- Interesting application of the actor model
Questions

Towards federated learning at scale: System design.


K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth.

Practical secure aggregation for privacy-preserving machine learning.


A. Shamir.

How to share a secret.
