CIS 490H/590J Edge Computing ML@Edge, Federated Learning

Winter 2023

Prof. Zheng Song

zhesong@umich.edu

Outline

1. **Brief introduction of machine learning**
2. Executing ML tasks at Edge
3. Training ML models at Edge

Credit:

https://github.com/wangshusen/DeepLearning/tree/master/Slides

https://inst.eecs.berkeley.edu/~cs294-163/fa19/slides/federated-learning.pdf

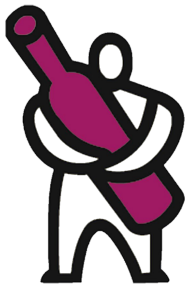
Basic Procedure of Machine Learning

Wine or Beer?

Based on two features: color (light wavelength) and alcohol content (percentage)

Basic Steps:

1) Collecting data (buy all wines and beers available on the market)

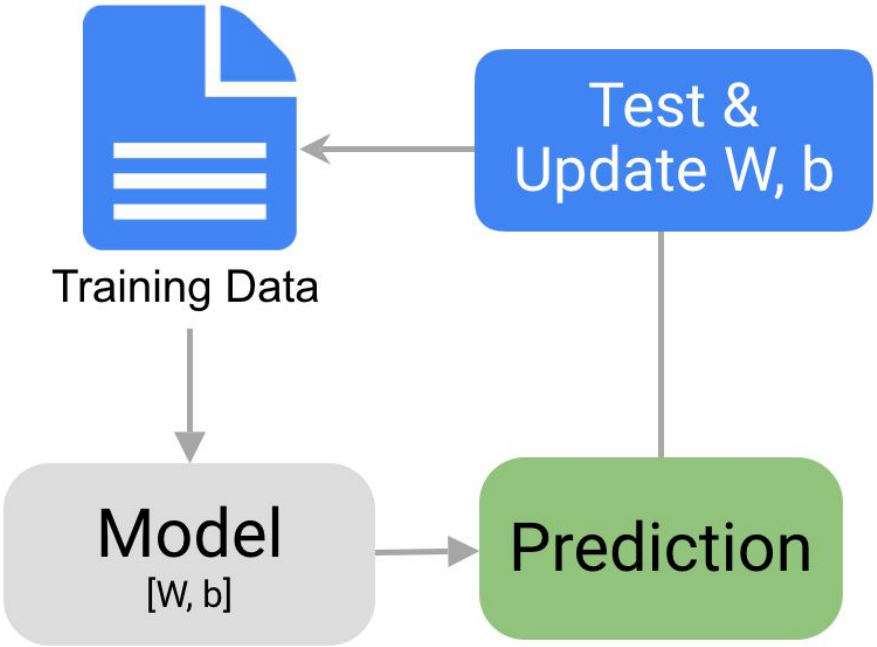


Color (nm)	Alcohol %	Beer or Wine?
610	5	Beer
599	13	Wine
693	14	Wine

Basic Procedure of Machine Learning

Basic Steps:

2) Choose a model and **training**



Basic Procedure of Machine Learning

Basic Steps:

3) **Prediction**

Color	Wavelength
violet	380–450 nm
blue	450–495 nm
green	495–570 nm
yellow	570–590 nm
orange	590–620 nm
red	620–750 nm

Ethanol sensor

Color: 660nm
Alcohol: 12%

Model
[W, b]

Prediction

Reference:
https://towardsdatascience.com/the-7-steps-of-machine-learning-2877d7e5548e

House Price Prediction by a Linear Predictor Model

- Inputs: $\mathbf{x} \in \mathbb{R}^d$ (e.g., features of a house).
- Prediction: $f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$ (e.g., housing price).

- $f(\mathbf{x}) = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$
- $w_1, w_2, \cdots, w_d$: weights
- x_1: # of bedrooms
- x_2: # of bathroom
- x_3: square feet
- x_4: age of house
- …

House Price Prediction by a Linear Predictor Model

- Inputs: $\mathbf{x} \in \mathbb{R}^d$ (e.g., features of a house).
- Prediction: $f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$ (e.g., housing price).

Question: How to find W?

Features of a House

$\mathbf{x} \in \mathbb{R}^d$

Price = \$0.5M

Prediction:

$f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$

Linear Predictor Model

- Inputs: $\mathbf{x} \in \mathbb{R}^d$ (e.g., features of a house).
- Prediction: $f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$ (e.g., housing price).

Question: How to find W?

- Training inputs: $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$.
- Training targets: $y_1, \cdots, y_n \in \mathbb{R}$.
- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2} \left(\mathbf{x}_i^T \mathbf{w} - y_i \right)^2$.

Linear Predictor Model

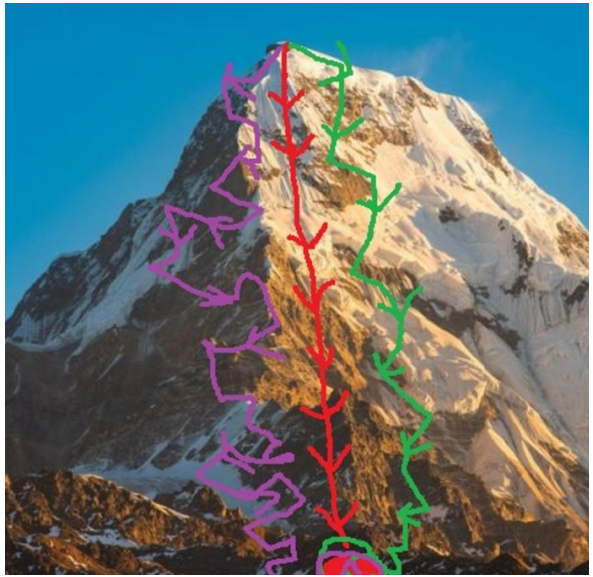
- Inputs: $\mathbf{x} \in \mathbb{R}^d$ (e.g., features of a house).
- Prediction: $f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$ (e.g., housing price).

Question: How to find W?

- Training inputs: $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$.
- Training targets: $y_1, \cdots, y_n \in \mathbb{R}$.
- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2} \left(\mathbf{x}_i^T \mathbf{w} - y_i \right)^2$.
- Least squares regression: $\mathbf{w}^\star = \underset{\mathbf{w}}{\operatorname{argmin}}\, L(\mathbf{w})$.

Solution: Gradient Descent (for Least Squares Regression)

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2$.
- Least squares regression: $\mathbf{w}^\star = \underset{\mathbf{w}}{\operatorname{argmin}}\, L(\mathbf{w})$.



Gradient: $g(\mathbf{w}) = \frac{\partial\, L(\mathbf{w})}{\partial\, \mathbf{w}} = \sum_{i=1}^{n} \frac{\partial\, \frac{1}{2}\left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2}{\partial\, \mathbf{w}} = \sum_{i=1}^{n} \left(\mathbf{x}_i^T \mathbf{w} - y_i\right) \mathbf{x}_i$

the direction of greatest change of L(w)

Solution: Gradient Descent (for Least Squares Regression)

Example: GD for least squares regression model

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2$.
- Gradient: $g(\mathbf{w}) = \sum_{i=1}^{n} g_i(\mathbf{w})$, where $g_i(\mathbf{w}) = \left(\mathbf{x}_i^T \mathbf{w} - y_i\right)\mathbf{x}_i$.
- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot g(\mathbf{w}_t)$.

\alhpa, learning rate (not too fast, not too slow)

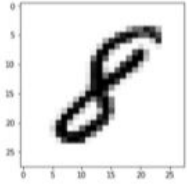
When to stop? L(w) won't change much at each step

Another Example: Handwriting Recognition

y = f (x, w)

x: {0,1} for 30*30 matrix = BoolArray[900]

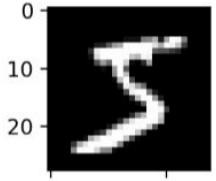
y: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Given input	Desired output
	8

Training

Given one input sample pair (x0, y0), the goal of ML model training is to find a set of parameters w, to maximize the probability of outputting y0 given x0.

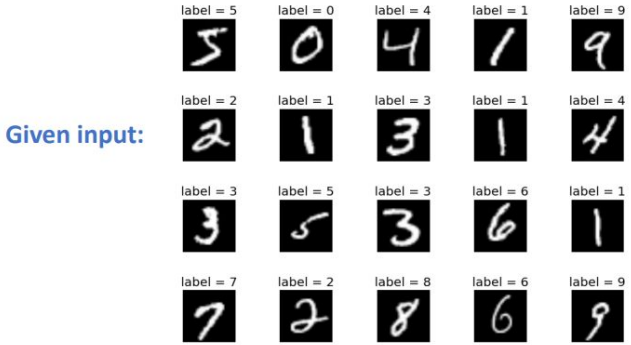
Given input: x_0

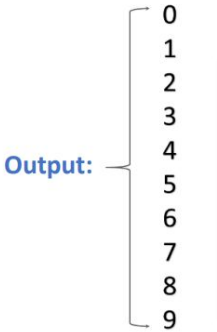


Maximize: $p(5|x_0, w)$

Training

Given a training dataset containing n input-output pairs (x_i, y_i, \forall i \in [1, n], find a set of parameter w, such that the average of p(y_i) is maximized given x_i





$$maximize \quad \frac{1}{n}\sum_{i=1}^{n} p(y_i|x_i, w)$$

$$minimize \quad \frac{1}{n}\sum_{i=1}^{n} -\log(p(y_i|x_i, w))$$

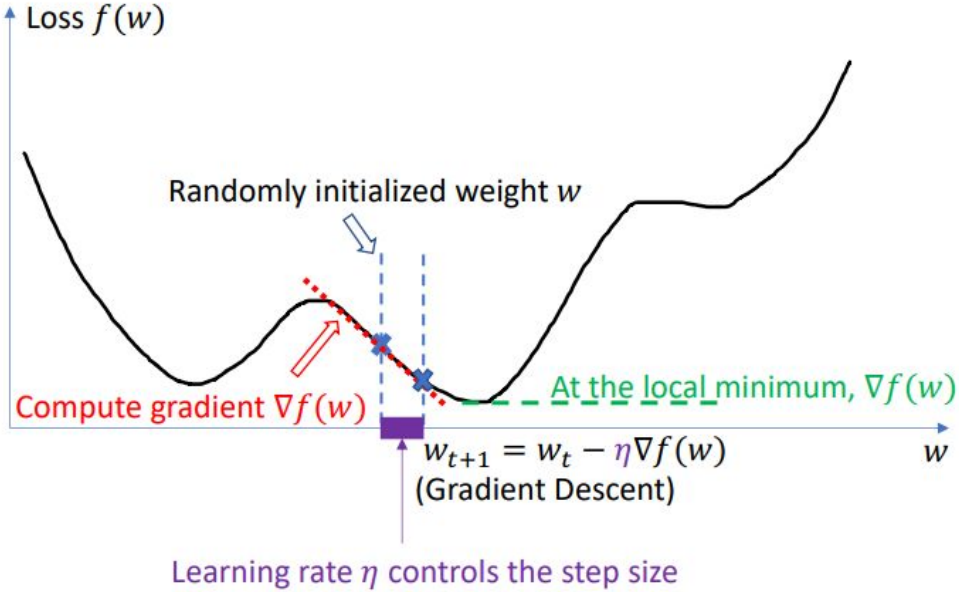
Training

For a training dataset containing n samples $(x_i, y_i), 1 \leq i \leq n$, the training objective is:

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

$f_i(w) = l(x_i, y_i, w)$ is the loss of the prediction on example (x_i, y_i)

Training Solution: Gradient Descent



How to stop? – when the update is small enough – converge.

$$\| w_{t+1} - w_t \| \leq \epsilon$$

or $$\| \nabla f(w_t) \| \leq \epsilon$$

Problem: *Usually the number of training samples n is large – slow convergence*

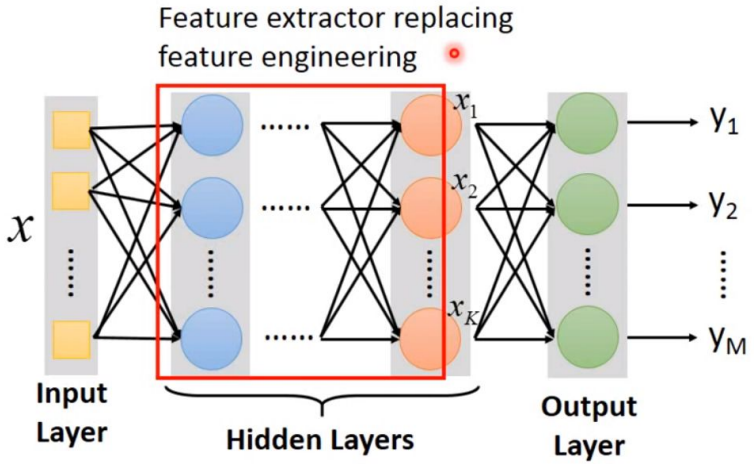
Deep Learning

More layers than traditional ML

Models are **BIG:** ResNet-50 has 25M parameters

Big Models are trained on big data, e.g., ImageNet has 14M images

Big Model + Big Data → Big Training Cost



Training Solution: Stochastic Gradient Descent (SGD)

At each step of gradient descent, instead of compute for all training samples, randomly pick a small subset (mini-batch) of training samples (x_k, y_k)

$$w_{t+1} \leftarrow w_t - \eta \nabla f(w_t; x_k, y_k)$$

Compared to gradient descent, SGD takes more steps to converge, but each step is much faster.

Outline

1. Brief introduction of machine learning
2. **Executing ML tasks at Edge**
3. Training ML models at Edge

ML@Edge Applications

Usages

Computer Vision

Natural Language Processing

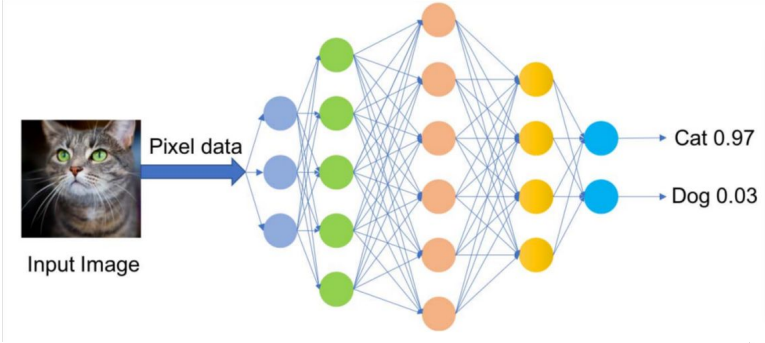
…

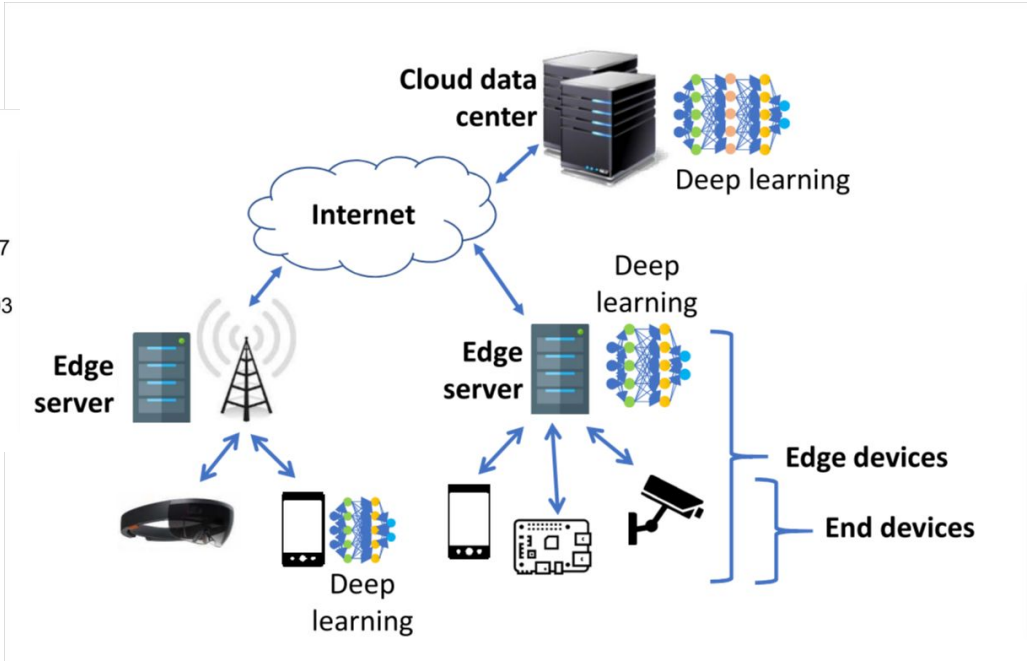
Wearable Camera

IEEE Computer, 2017

Executing ML Tasks @Edge

For latency, privacy reasons…

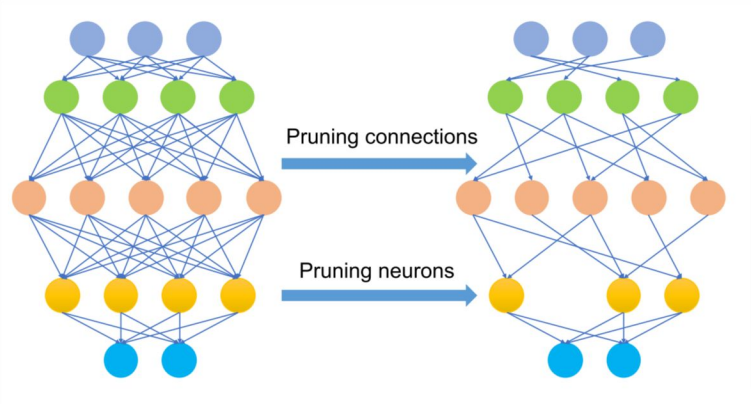




How to adapt ML for resource-constrained Edges?

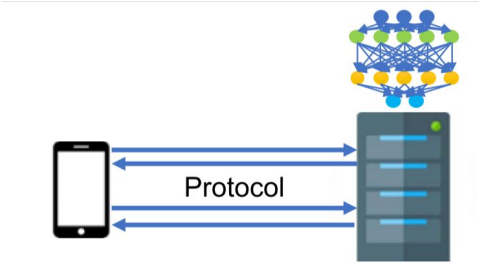
Run ML on Mobile Edge Devices:

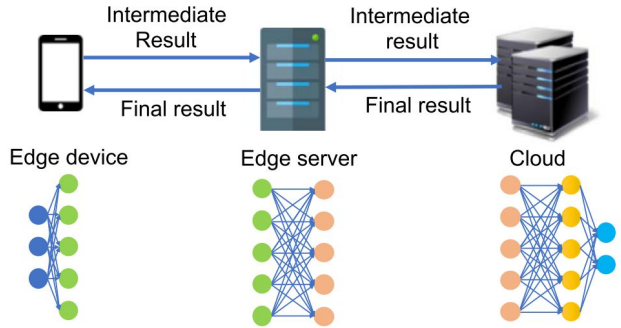
- Simplify ML models: MobileNets, Yolo, SqueezeNet (e.g., decompose the convolution filter into two simpler operations)
- Model Compression:
  - Parameter quantization: floating point parameters → low-bit width numbers
  - Pruning: removing the least important parameters
- Hardware: Tensor processing Unit



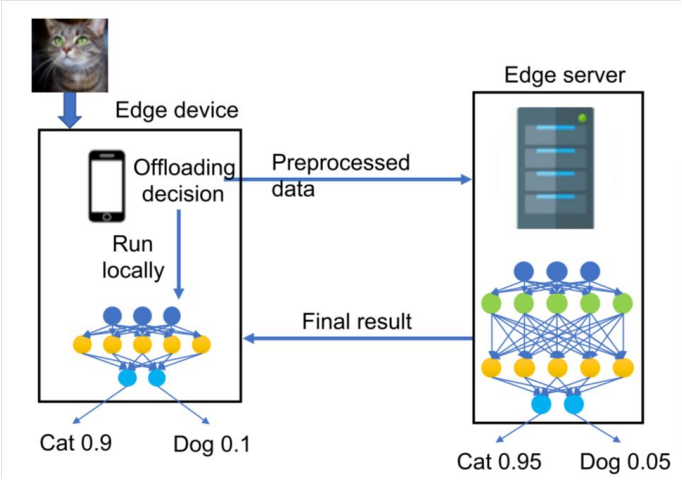
How to adapt ML for resource-constrained Edges?

Run ML on Dedicated Edge Servers





Model Partitioning



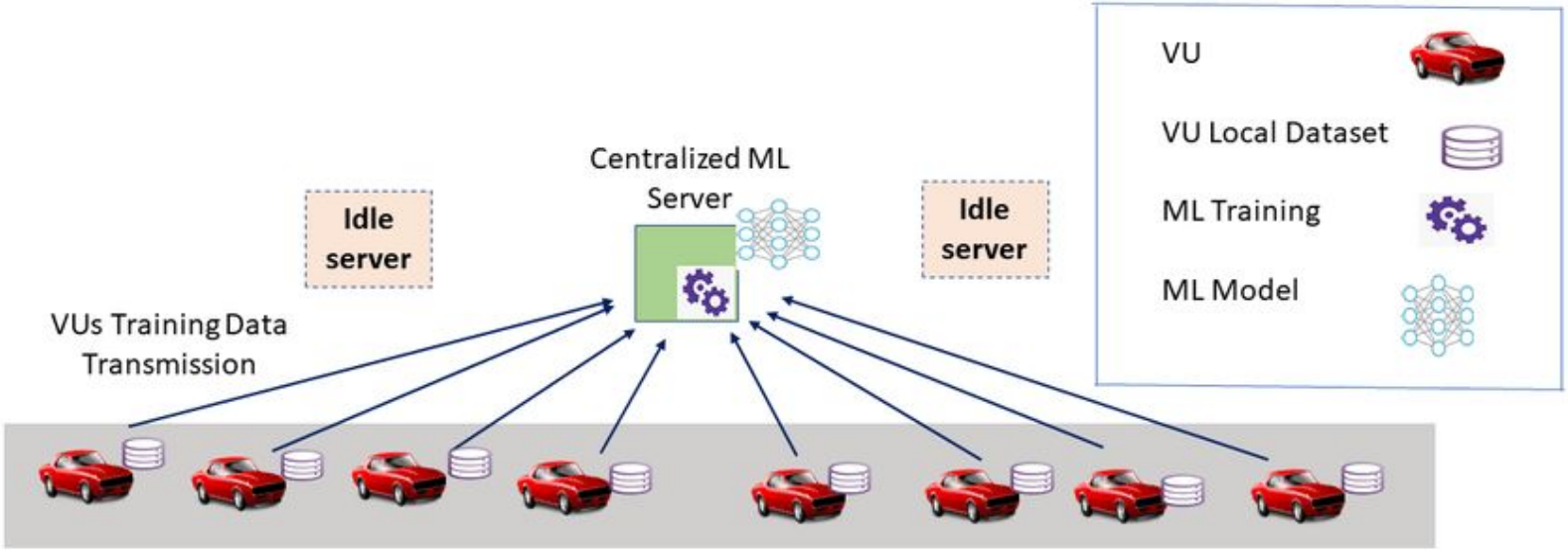
Outline

1. Brief introduction of machine learning
2. Executing ML tasks at Edge
3. **Training ML models at Edge**

Cloud-based ML Training

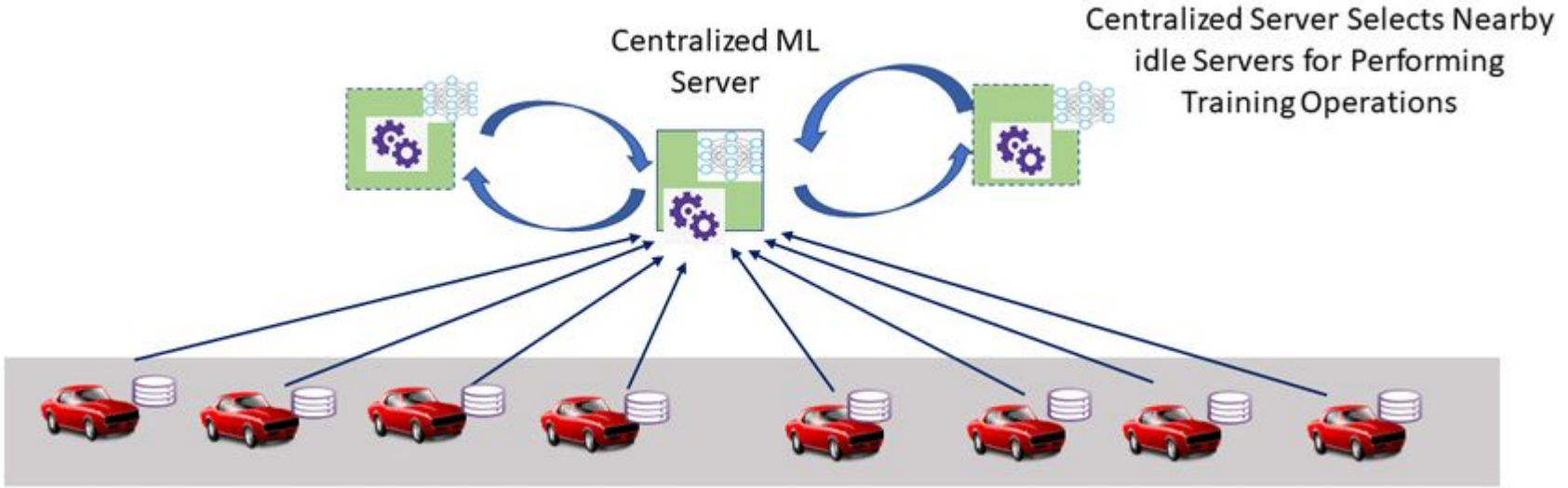
Problems:

1. Take pretty long to transmit sensing data to cloud server
2. Take pretty long to train the ML model on centralized server



Cloud-based Distributed ML

To solve Problem #2, centralized ML server distributes training tasks on nearby servers



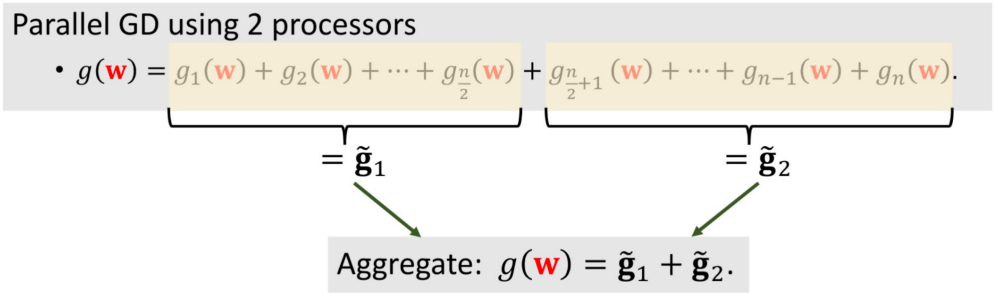
Recalling the previous example

Example: GD for least squares regression model

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2$.
- Gradient: $g(\mathbf{w}) = \sum_{i=1}^{n} g_i(\mathbf{w})$, where $g_i(\mathbf{w}) = \left(\mathbf{x}_i^T \mathbf{w} - y_i\right)\mathbf{x}_i$.

Parallel GD using 2 processors

- $g(\mathbf{w}) = g_1(\mathbf{w}) + g_2(\mathbf{w}) + \cdots + g_{\frac{n}{2}}(\mathbf{w}) + g_{\frac{n}{2}+1}(\mathbf{w}) + \cdots + g_{n-1}(\mathbf{w}) + g_n(\mathbf{w})$.



Parallel Gradient Descent

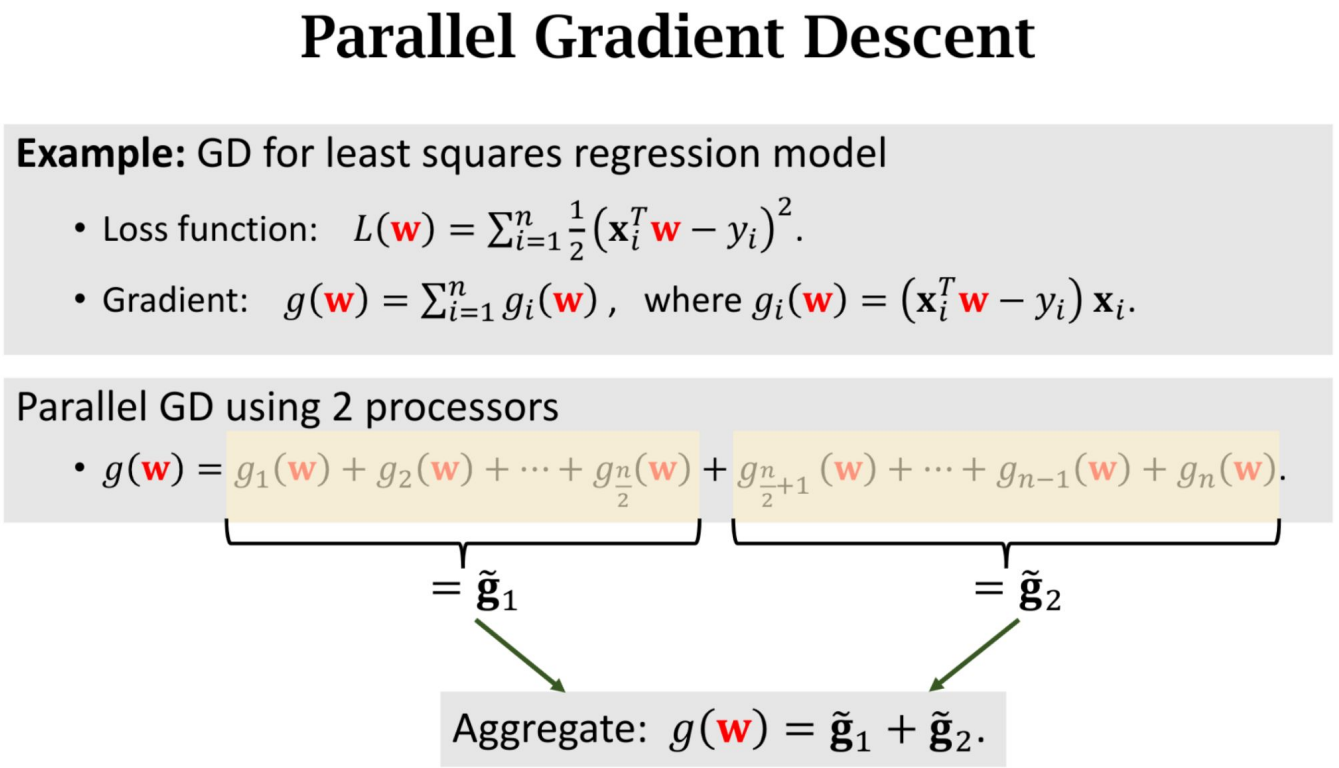
Example: GD for least squares regression model

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2$.
- Gradient: $g(\mathbf{w}) = \sum_{i=1}^{n} g_i(\mathbf{w})$, where $g_i(\mathbf{w}) = \left(\mathbf{x}_i^T \mathbf{w} - y_i\right)\mathbf{x}_i$.

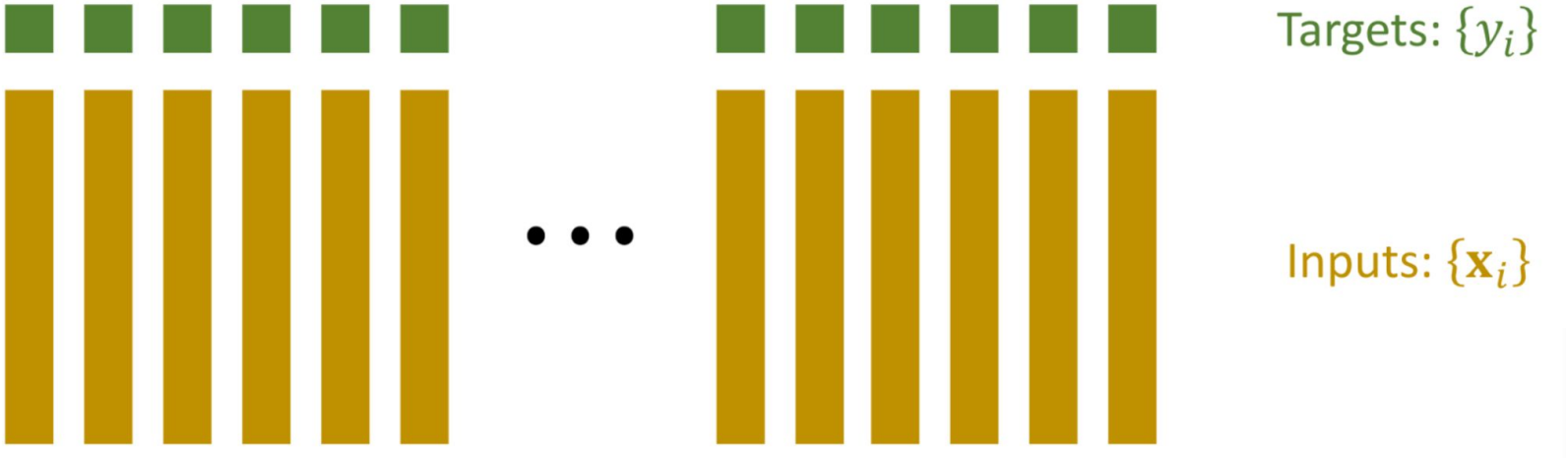
Parallel GD using 2 processors

- $g(\mathbf{w}) = g_1(\mathbf{w}) + g_2(\mathbf{w}) + \cdots + g_{\frac{n}{2}}(\mathbf{w}) + g_{\frac{n}{2}+1}(\mathbf{w}) + \cdots + g_{n-1}(\mathbf{w}) + g_n(\mathbf{w})$.

Parallel Gradient Descent

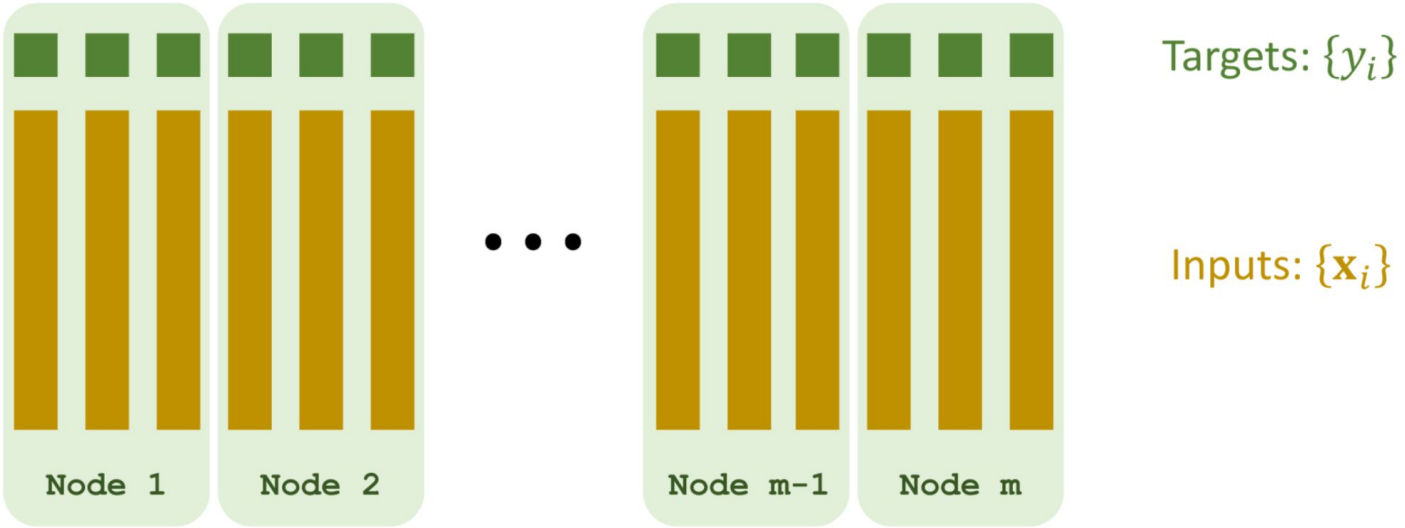


Distributed Machine Learning



Distributed Machine Learning

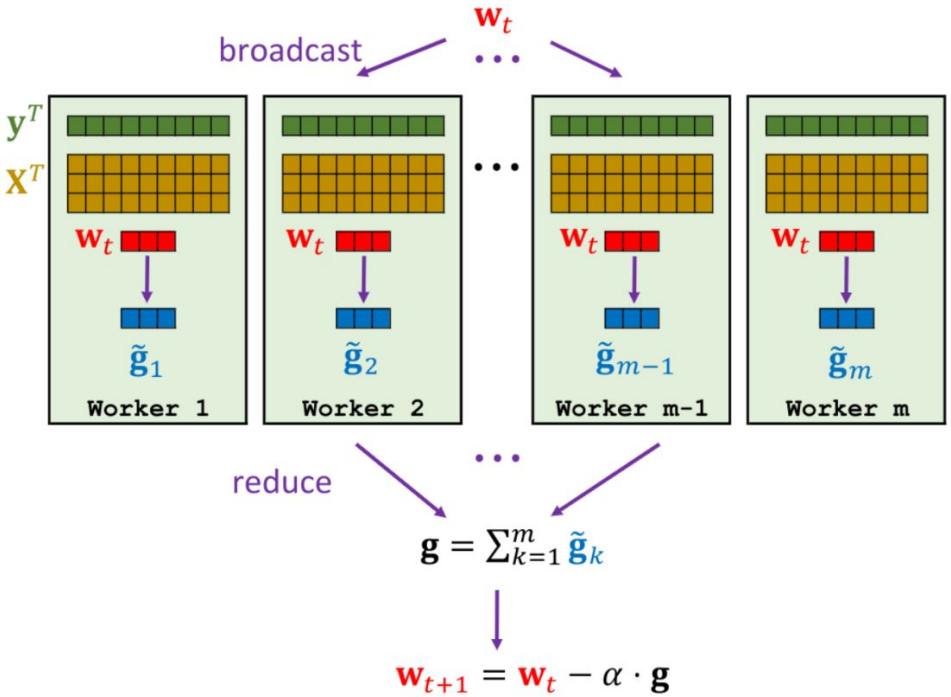
Partition the data among worker nodes (each node has a subset of data)



Distributed Machine Learning by Map-reduce

- Broadcast: Server broadcast the up-to-date parameters $\mathbf{w}_t$ to workers.
- Map: Workers do computation locally.
  - Map $(\mathbf{x}_i, y_i, \mathbf{w}_t)$ to $\mathbf{g}_i = \left(\mathbf{x}_i^T \mathbf{w}_t - y_i\right) \mathbf{x}_i$.
  - Obtain n vectors: $\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \cdots, \mathbf{g}_n$.
- Reduce: Compute the sum: $\mathbf{g} = \sum_{i=1}^{n} \mathbf{g}_i$.
  - Every worker sums all the $\{\mathbf{g}_i\}$ stored in its local memory to get a vector.
  - Then, the server sums the resulting m vectors. (There are m workers.)
- Server updates the parameters: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \mathbf{g}$.

Distributed Machine Learning by Map-reduce



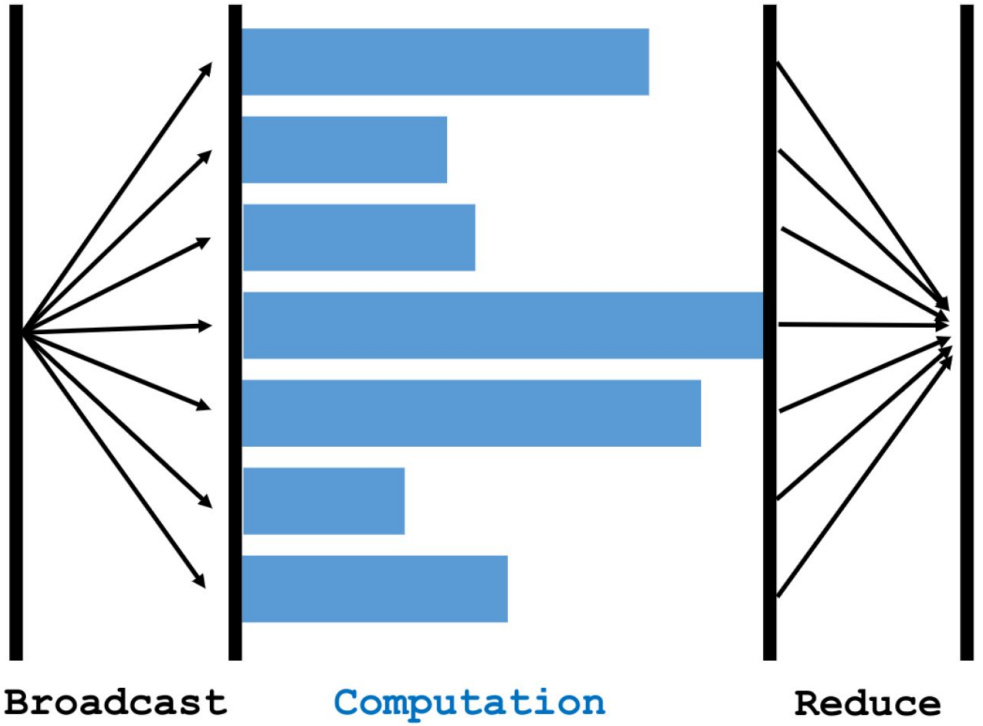
- Every worker stores $\frac{1}{m}$ of the data.
- Every worker does $\frac{1}{m}$ of the computation.
- Is the runtime reduced to $\frac{1}{m}$?

- No. Because communication and synchronization must be considered.

Bulk Synchronous

Slow Nodes: Straggler

Problem: low efficiency



Distributed Machine Learning by Parameter Server

Parameter Server was proposed by [1] for scalable machine learning

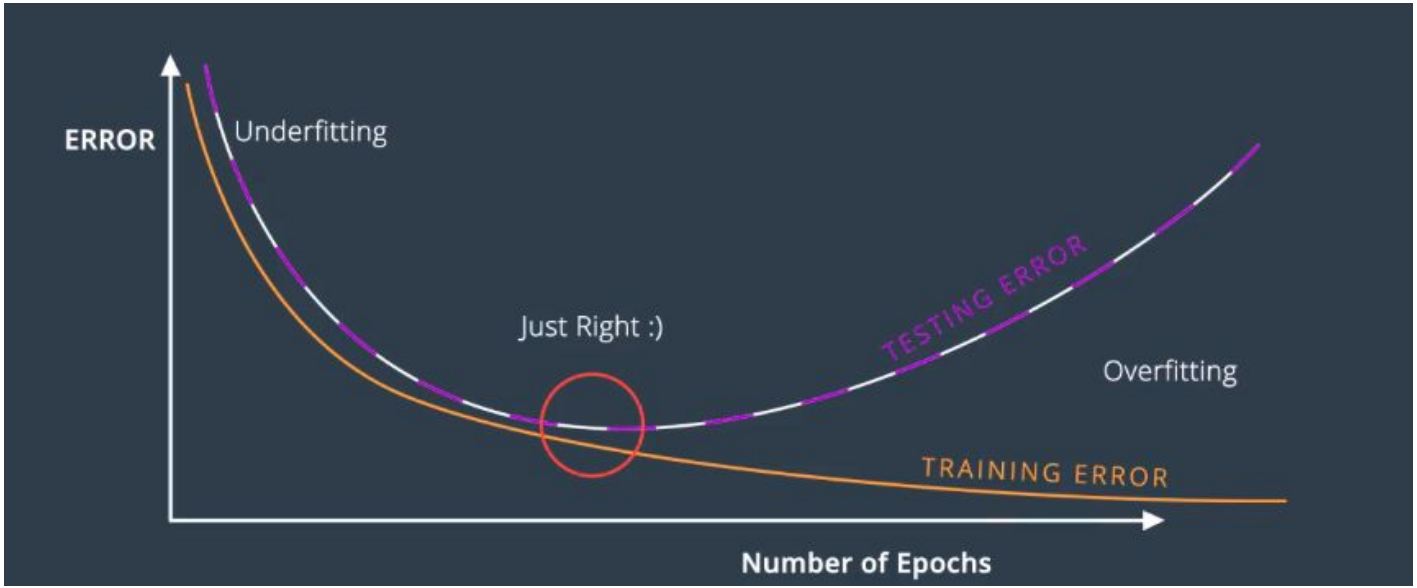
Characters: client-server architecture, message passing communication and asynchronous.

[1] Li MU et al. : Scaling Distributed Machine Learning with the Parameter Server, In OSDI, 2014

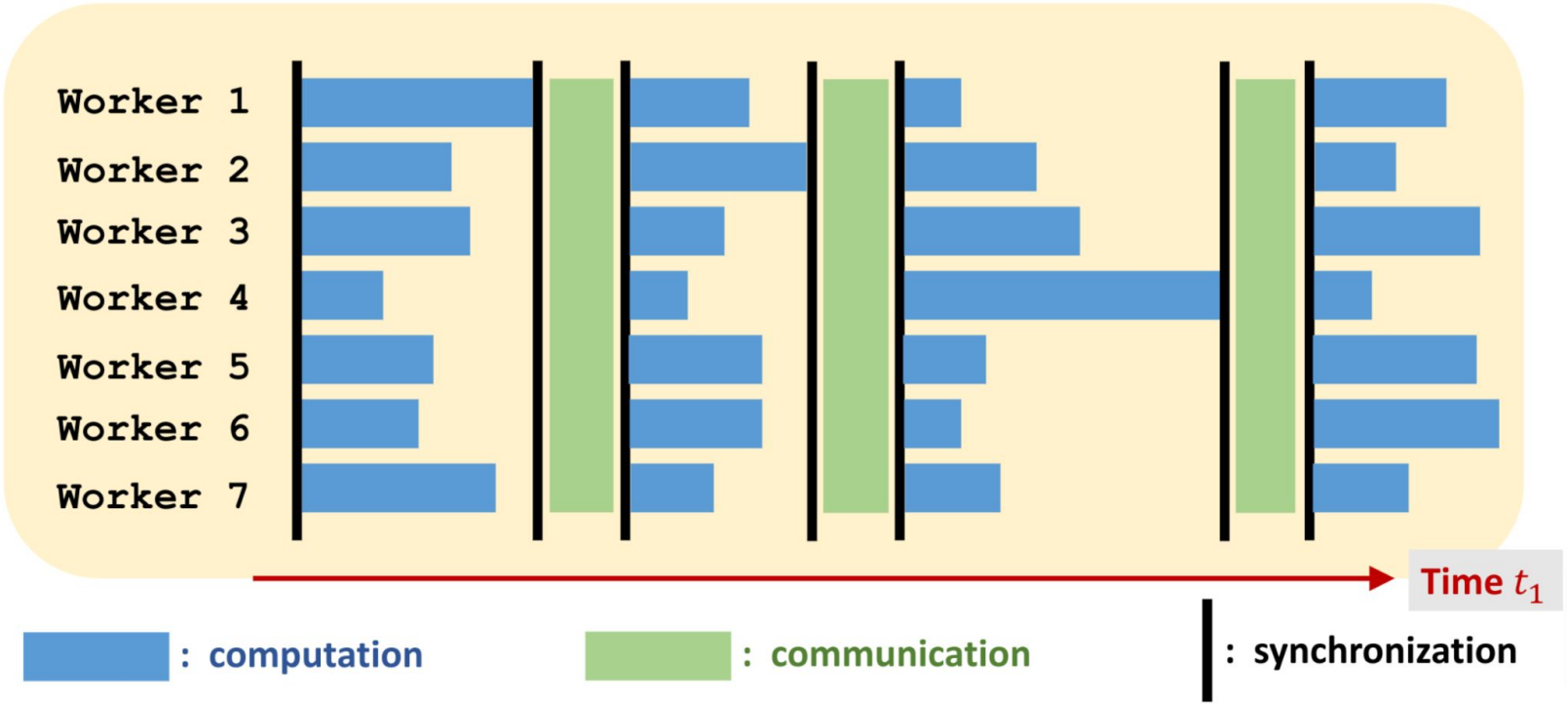
Epoch:

We often need to train the model multiple times over the dataset

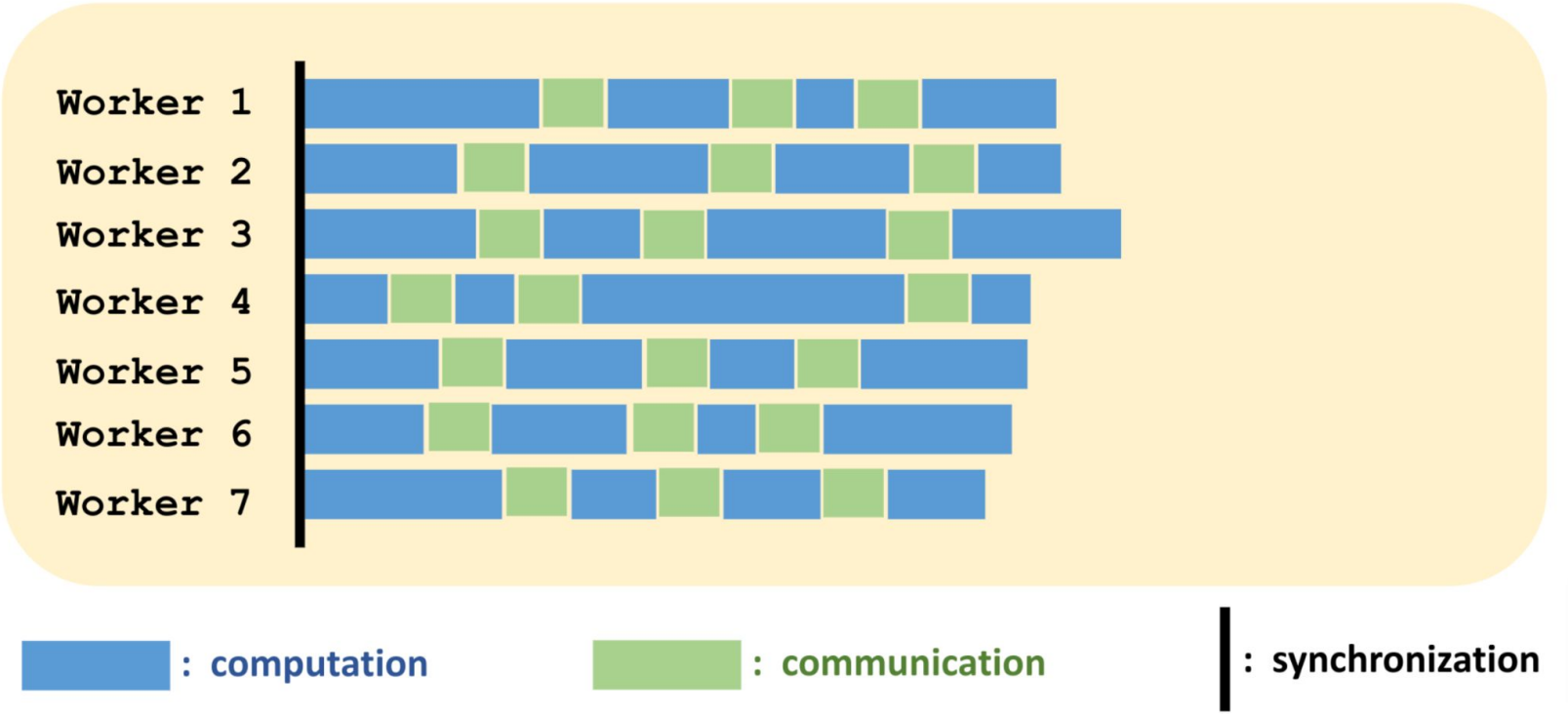
Epoch: the number of complete passes through the training dataset.



Synchronous Algorithm



Asynchronous Algorithm



Asynchronous Gradient Descent

The i-th worker repeats:

1. Pull the up-to-date model parameters $\mathbf{w}$ from the server.
2. Compute gradient $\tilde{\mathbf{g}}_i$ using its local data and $\mathbf{w}$.
3. Push $\tilde{\mathbf{g}}_i$ to the server.

The server performs:

1. Receive gradient $\tilde{\mathbf{g}}_i$ from a worker.
2. Update the parameters by:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \tilde{\mathbf{g}}_i.$$

Asynchronous Gradient Descent

The i-th worker repeats:

1. Pull the up-to-date model parameters $\mathbf{w}$ from the server.
2. Compute gradient $\tilde{\mathbf{g}}_i$ using its local data and $\mathbf{w}$.
3. Push $\tilde{\mathbf{g}}_i$ to the server.

The server performs:

1. Receive gradient $\tilde{\mathbf{g}}_i$ from a worker.
2. Update the parameters by:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \tilde{\mathbf{g}}_i.$$

What’s wrong with cloud-based training?

The biggest obstacle to using advanced data analysis isn’t skill base or technology; it’s plain old access to the data --- Edd Wilder-James, Harvard Business Review

Two Main Problems

- Privacy: photo/password/url/messages
- Network Overhead

Solution: train a model locally and upload the model

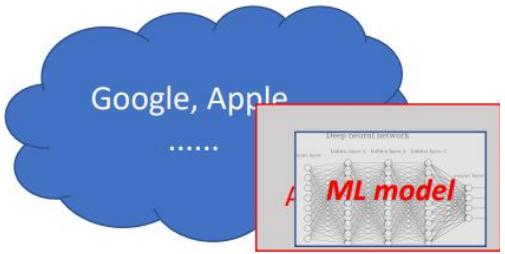
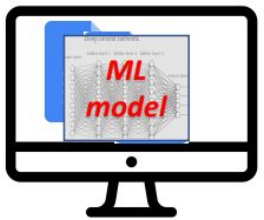


image classification:
e.g. to predict which photos are most likely to be viewed multiple times in the future;

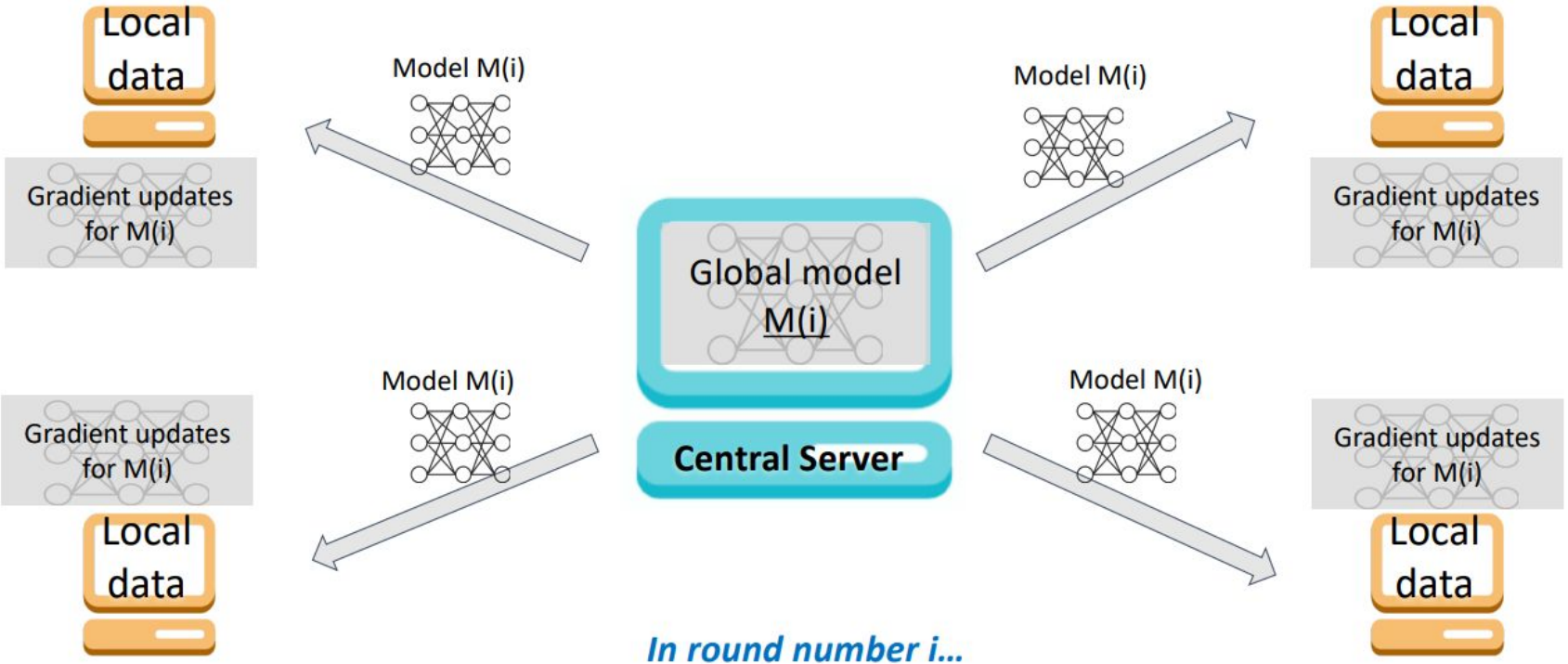
language models:
e.g. voice recognition, next-word-prediction, and auto-reply in Gmail



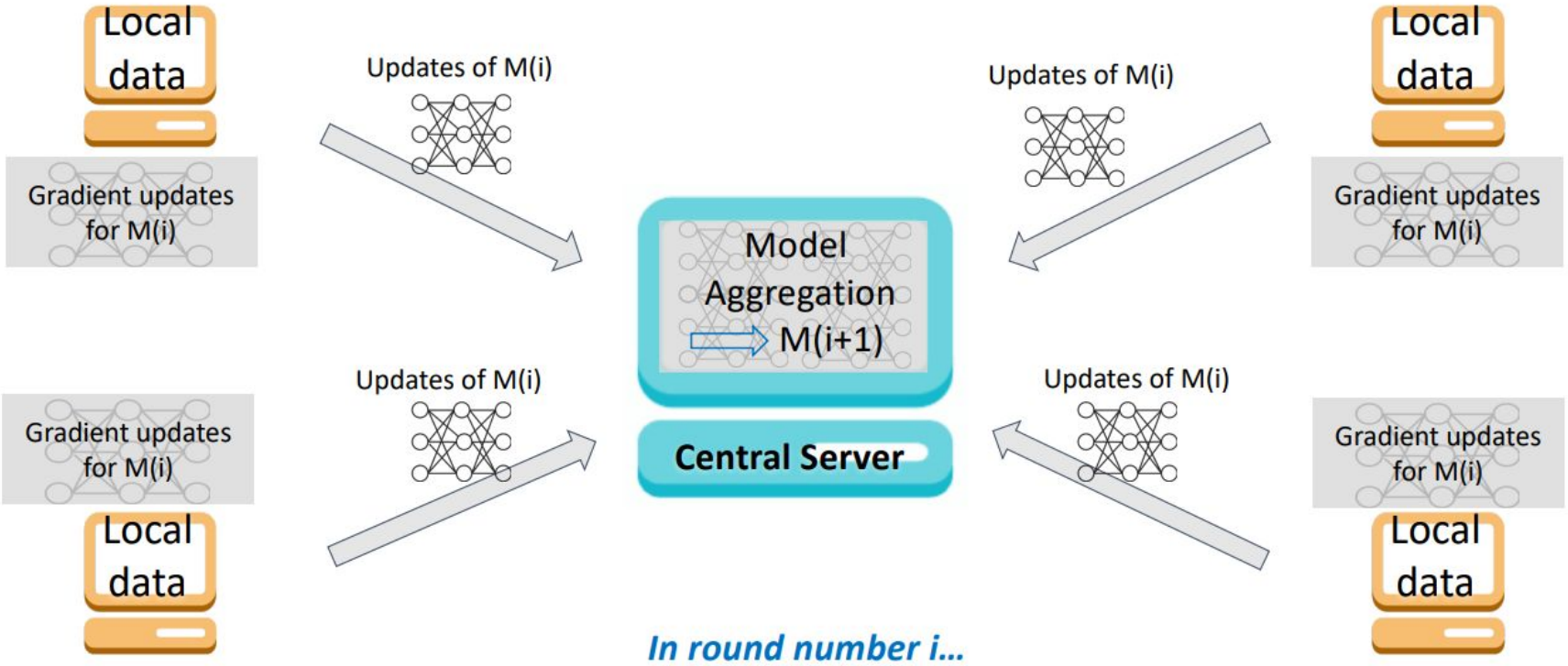
Example

Next Word Prediction

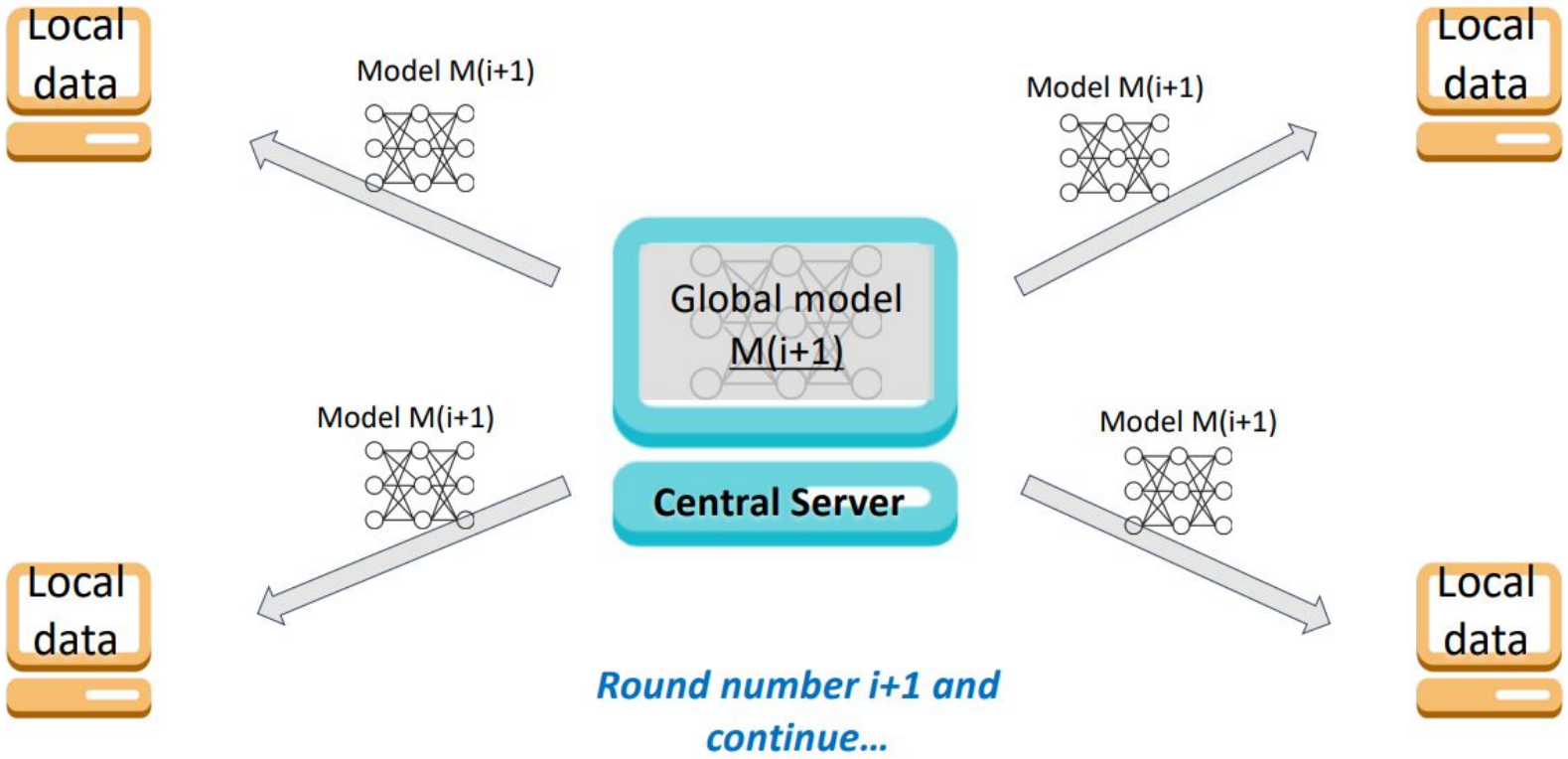
Federated Learning Overview



Federated Learning Overview



Federated Learning Overview



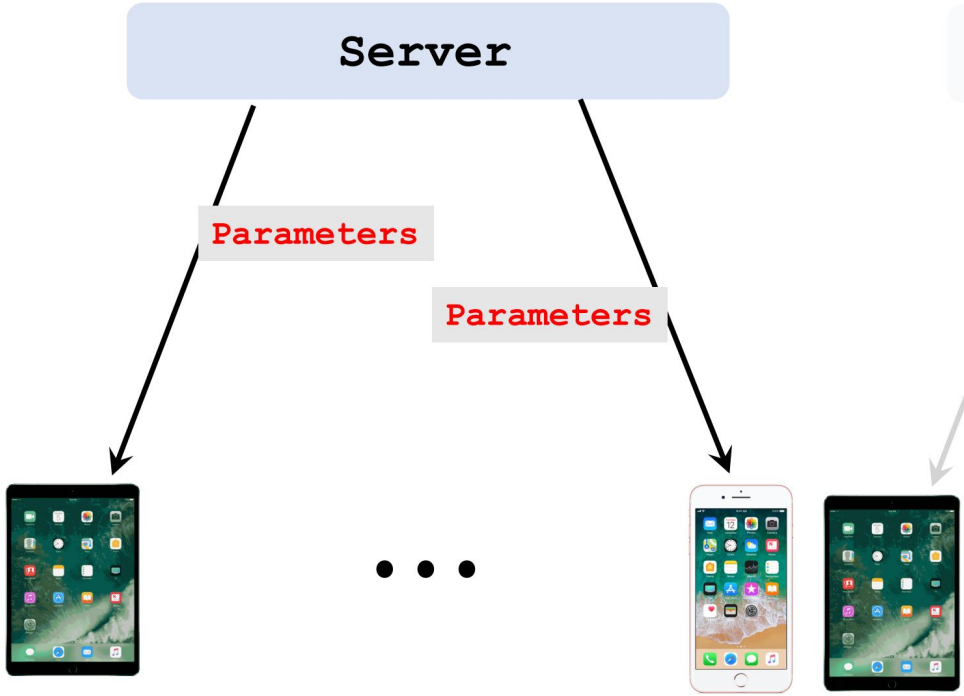
What is Federated Learning?

Federated Learning is a kind of distributed learning

How does Federated Learning different from traditional distributed learning?

1. Users have control over their device and data.
2. Worker nodes are unstable.
3. **Communication cost is higher than computation cost.**
4. Data stored on worker nodes are not IID (independent and identically distributed)
5. The amount of data on each node is severely imbalance.

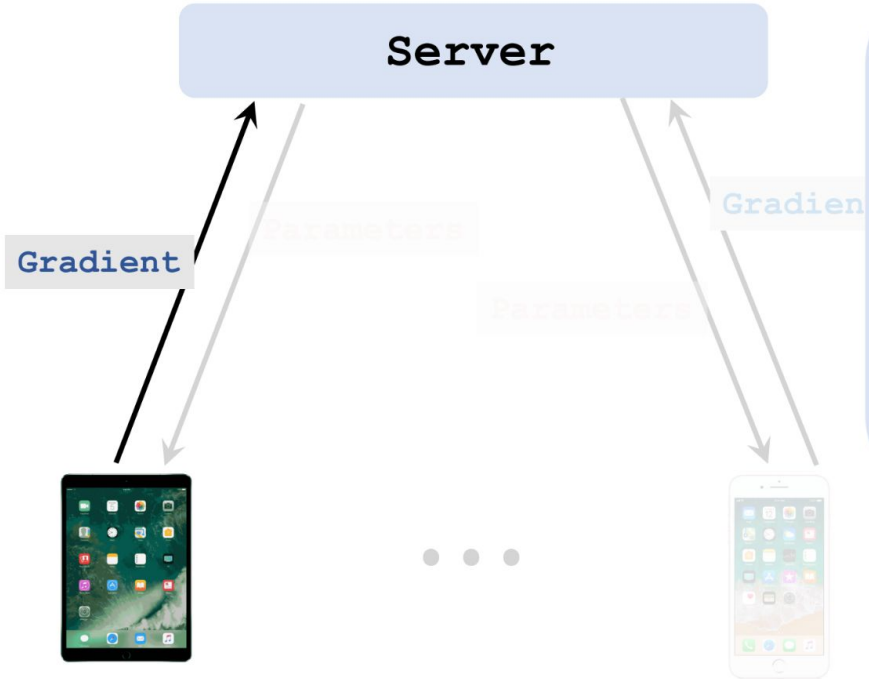
Distributed ML



The i-th worker performs:

1. Receiving model parameters $\mathbf{w}$ from the server.
2. Using $\mathbf{w}$ and its local data to compute gradient $\mathbf{g}_i$.
3. Sending $\mathbf{g}_i$ to the server.

Distributed ML



The server performs:

1. Receiving gradients $\mathbf{g}_1, \cdots, \mathbf{g}_m$ from all the m workers.
2. Computing $\mathbf{g} = \mathbf{g}_1 + \cdots + \mathbf{g}_m$.
3. Updating model parameters:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \mathbf{g}.$$

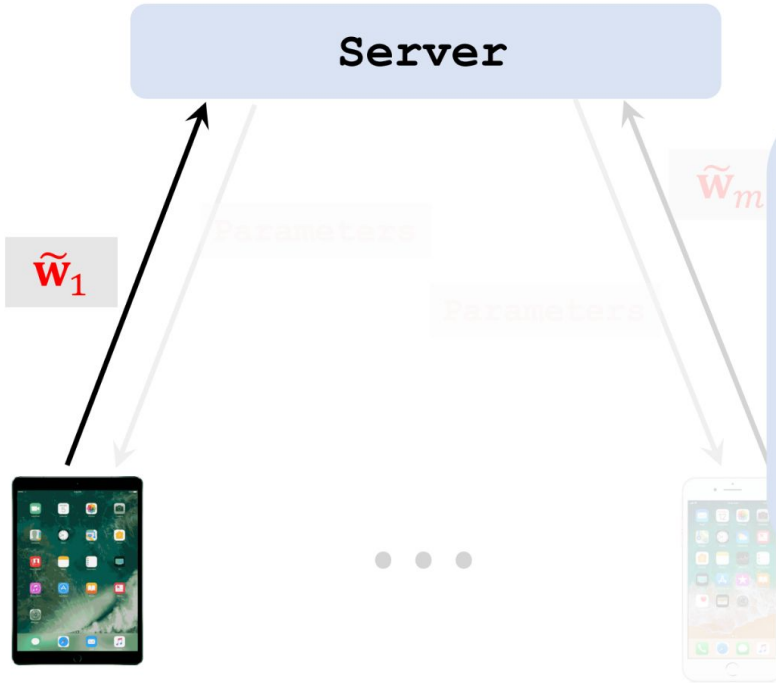
FedAvg

Epoch

The i-th worker performs:

1. Receiving model parameters $\mathbf{w}$ from the server.
2. Repeating the followings:
   a) Using $\mathbf{w}$ and its local data to compute gradient $\mathbf{g}$.
   b) Local update: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \mathbf{g}$.
3. Sending $\widetilde{\mathbf{w}}_i = \mathbf{w}$ to the server.

FedAvg



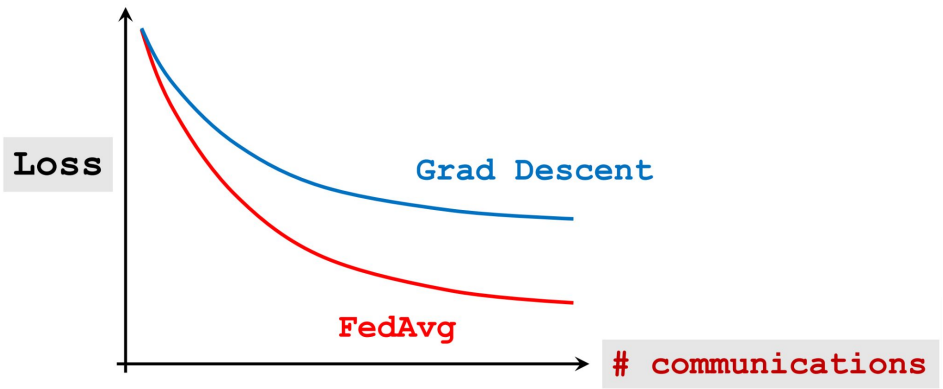
The server performs:

1. Receiving $\widetilde{\mathbf{w}}_1, \cdots, \widetilde{\mathbf{w}}_m$ from all the m workers.
2. Updating model parameters:

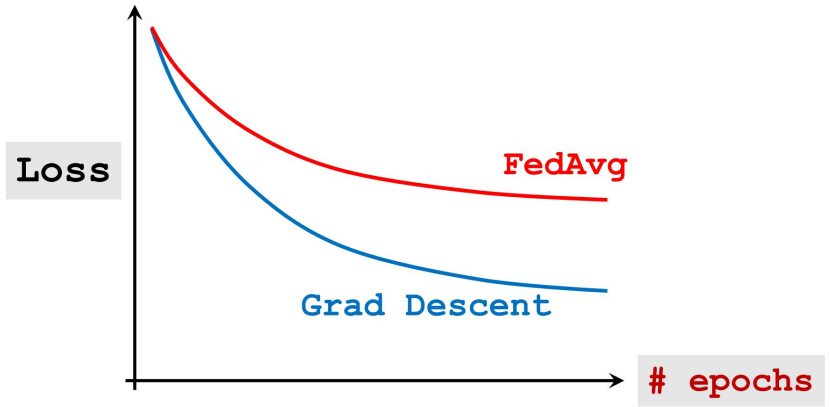
$$\mathbf{w} \leftarrow \frac{1}{m}(\widetilde{\mathbf{w}}_1 + \cdots + \widetilde{\mathbf{w}}_m).$$

FedAvg Vs. Gradient Descent

Measured by # communications, Federated Averaging is faster.



Measured by # epochs, Federated Averaging is slower.



Federated learning: Basics and application to the mobile keyboard