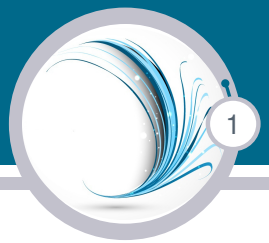
Distributed Statistial Machine Learning in Adversarial Settings: Byzantine Gradient Descent

Project for Convex Optimization SI215

Wang Zhuoli Li Weijing

June 20, 2018

Content



Purpose

- Design robust algorithms such that the system can learn the underlying true parameter, despite the interruption of the Byzantine attacks.

Application

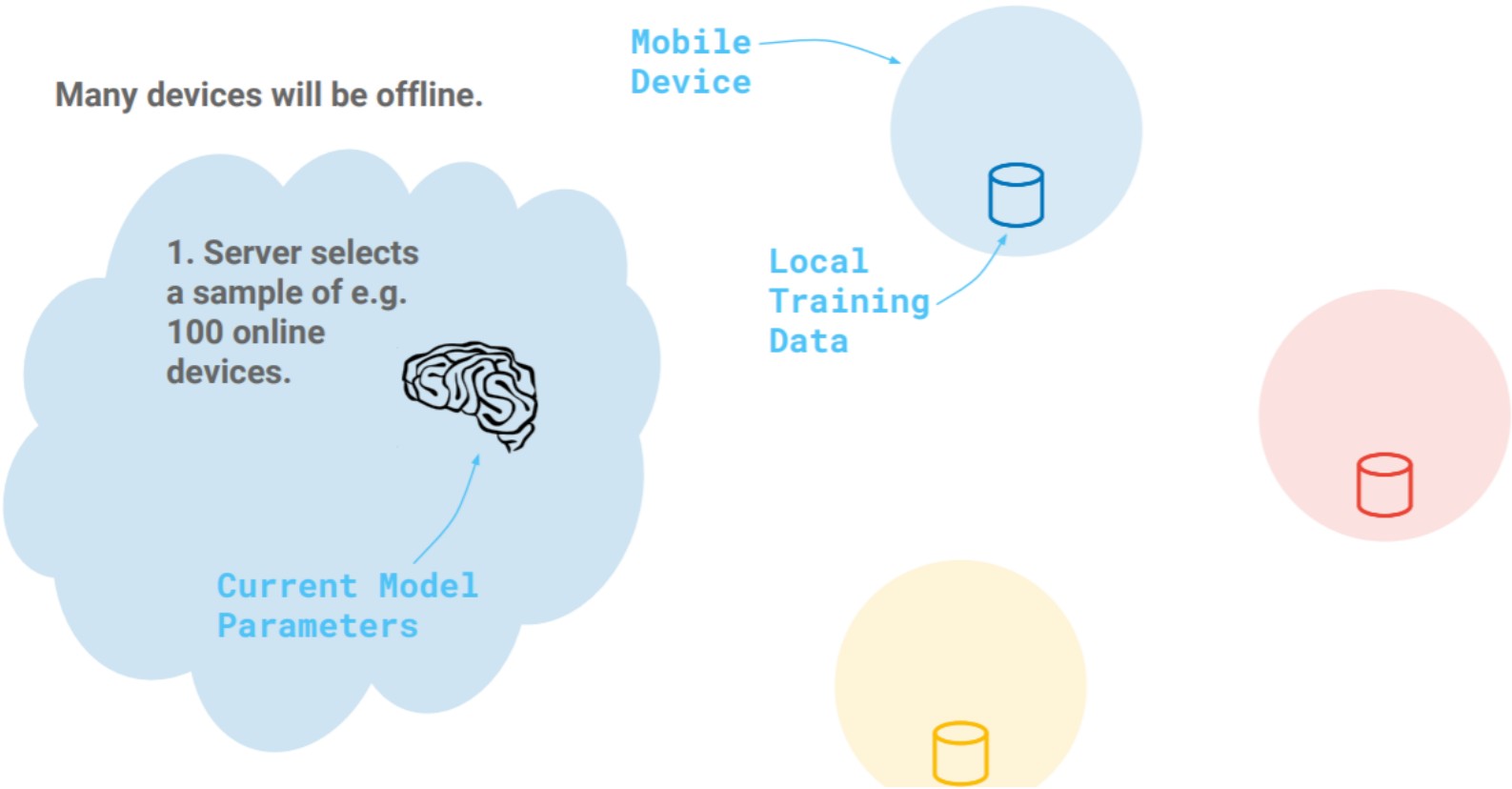
- Federated learning

Byzantine fault tolerance (BFT)

The dependability of a fault-tolerant computer system, particularly distributed computing systems, where components may fail and there is imperfect information on whether a component is failed. In a "Byzantine failure", a component such as a server can inconsistently appear both failed and functioning to failure-detection systems, presenting different symptoms to different observers.

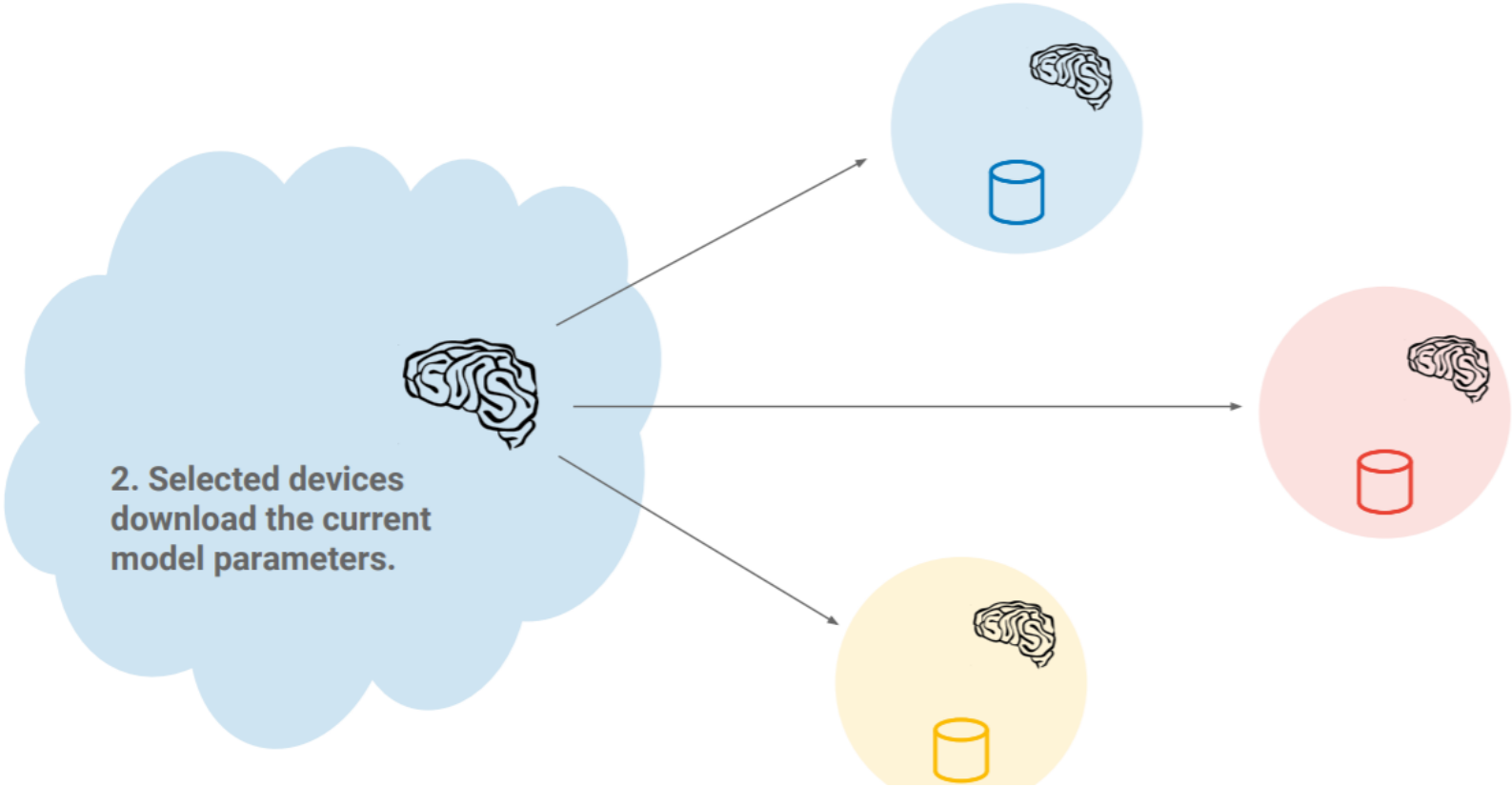
Introduction of Federated Learning

Federated Learning



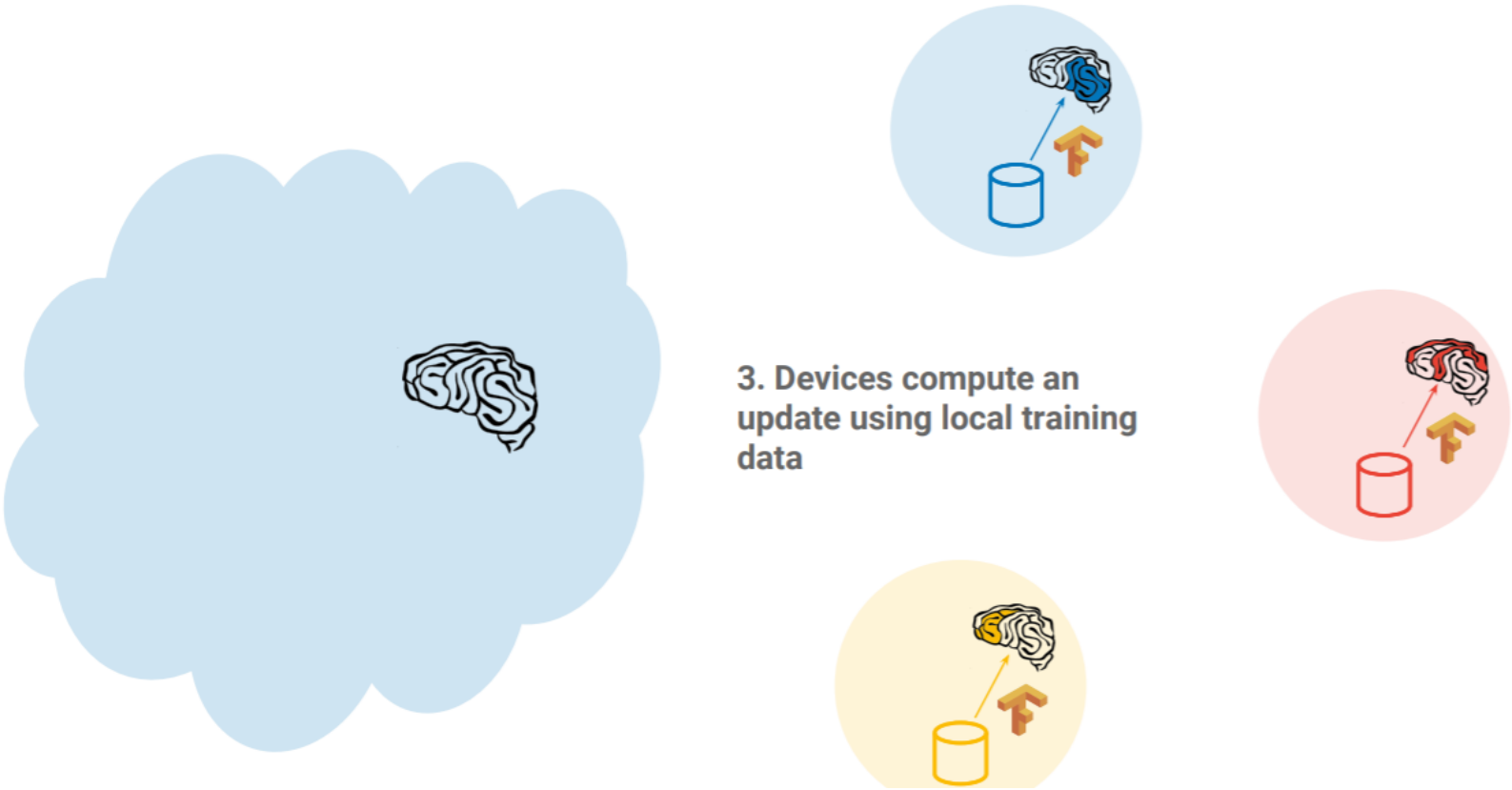
Introduction of Federated Learning

Federated Learning



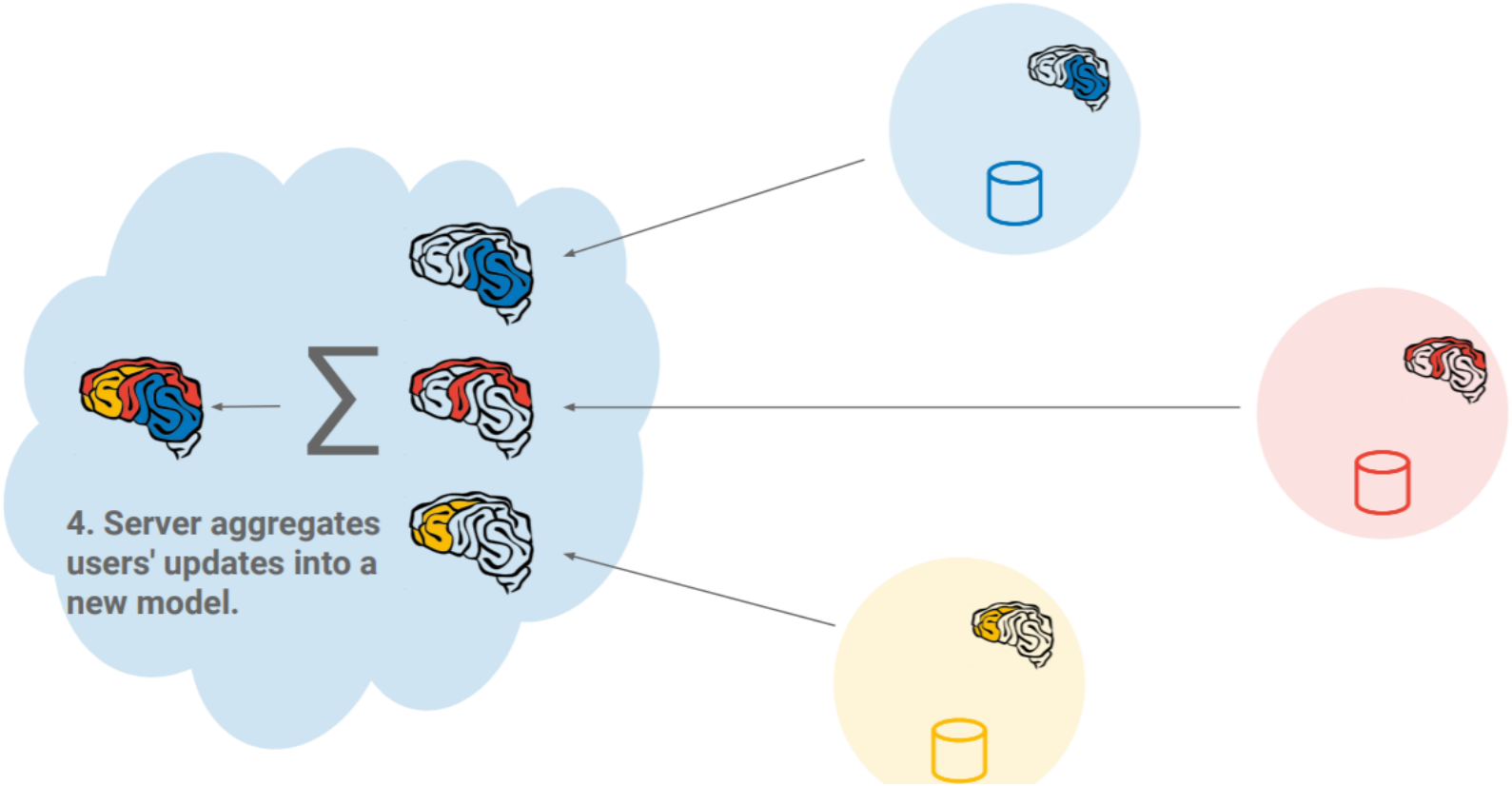
Introduction of Federated Learning

Federated Learning



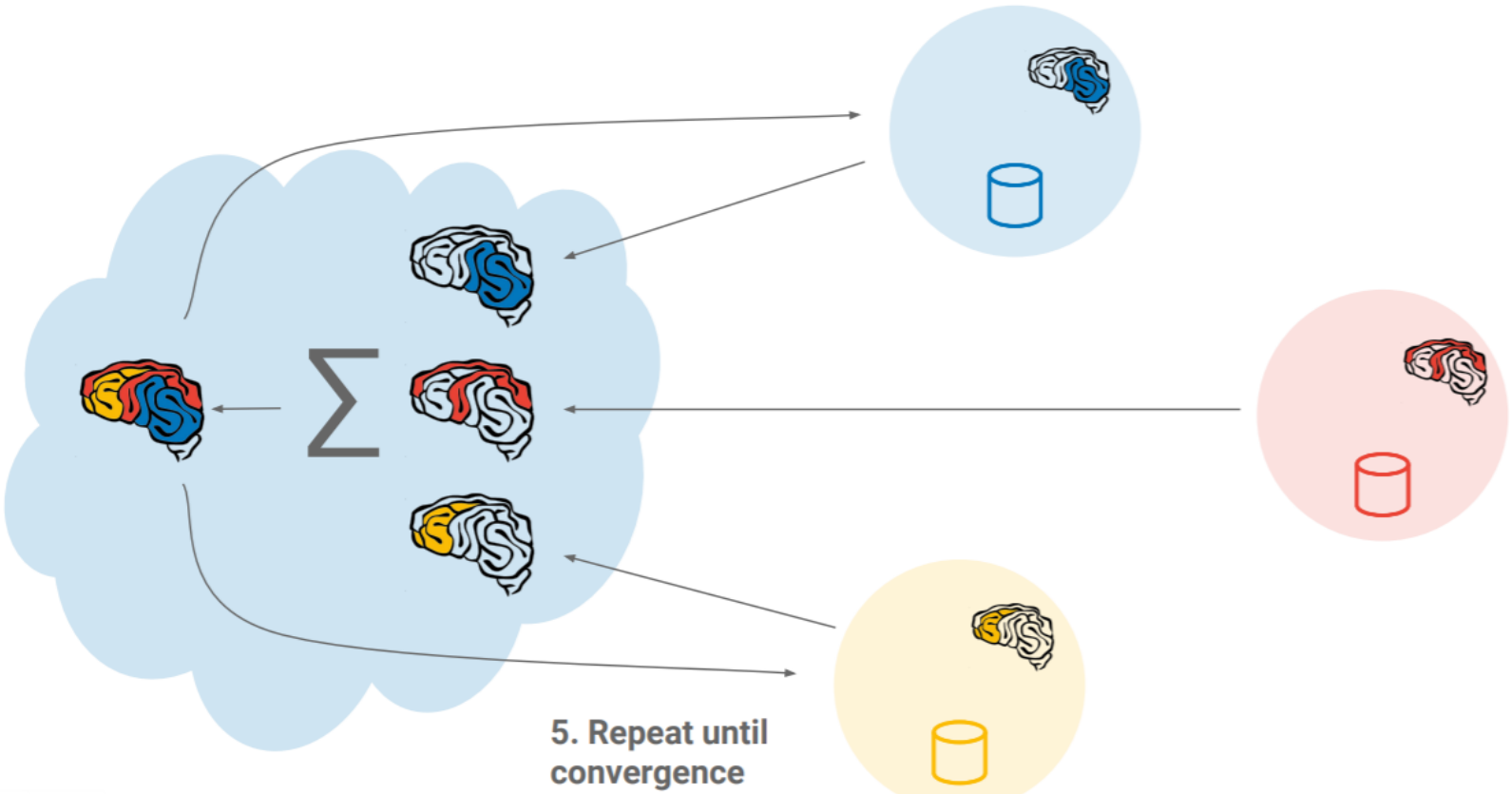
Introduction of Federated Learning

Federated Learning

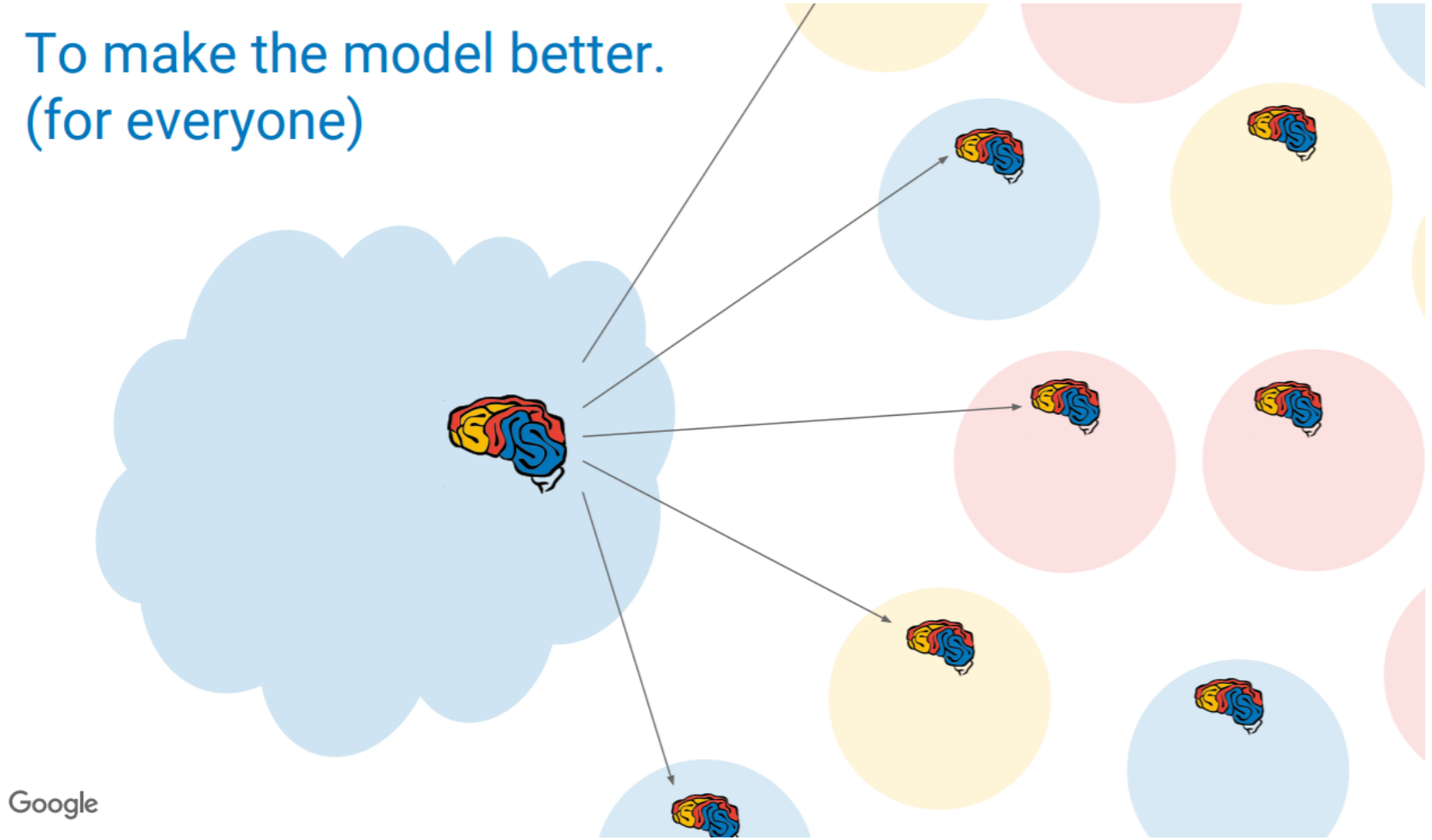


Introduction of Federated Learning

Federated Learning



Introduction of Federated Learning



Motivation
Drawback of Federated Learning

Federated learning three problems:

- Security
- Small local datasets versus high model complexity
- Communication constraints

- $X \in \mathcal{X}$: input data generated according to some unknown distribution μ.
- $\Theta \in \mathbb{R}^d$: set of all choices of model parameters.
- $f(x, \theta)$: measures the risk induced by a realization x of the data under the model parameter choice θ.

We are interested in learning the model choice θ^* that minimizes the population risk, i.e.

$$\theta^* \in \arg\min_{\theta \in \Theta} F(\theta) \triangleq \mathbb{E}\left[f(X, \theta)\right]$$

ASSUMPTION 1. *The population risk function $F : \Theta \to \mathbb{R}$ is L-strongly convex, and differentiable over Θ with M-Lipschitz gradient. That is, for all $\theta, \theta' \in \Theta$,*

$$F(\theta') \geq F(\theta) + \langle \nabla F(\theta), \theta' - \theta \rangle + \frac{L}{2}\|\theta' - \theta\|^2,$$

and

$$\|\nabla F(\theta) - \nabla F(\theta')\| \leq M\|\theta - \theta'\|.$$

Under Assumption 1, it is well-known [4] that using the standard gradient descent update

$$\theta_t = \theta_{t-1} - \eta \times \nabla F(\theta_{t-1}), \tag{2}$$

where η is some fixed stepsize, θ_t approaches θ^* exponentially fast. In particular, choosing $\eta = L/(2M^2)$, it holds that

$$\|\theta_t - \theta^*\| \leq \left(1 - \left(\frac{L}{2M}\right)^2\right)^{t/2} \|\theta_0 - \theta^*\|.$$

Nevertheless, when the distribution μ is unknown, as assumed in this paper, the population gradient ∇F can only be approximated using sample gradients, if they exist.

Assumption
Standard gradient descent

Algorithm 1 Standard Gradient Descent: Iteration $t \geq 1$

Parameter server:

1: *Initialize:* Let θ_0 be an arbitrary point in Θ.
2: Broadcast the current model parameter estimator θ_{t-1} to all working machines;
3: Wait to receive all the gradients reported by the m machines; Let $g_t^{(j)}(\theta_{t-1})$ denote the value received from machine j.
If no message from machine j is received, set $g_t^{(j)}(\theta_{t-1})$ to be some arbitrary value;
4: Update: $\theta_t \leftarrow \theta_{t-1} - \eta \times \left(\frac{1}{m}\sum_{j=1}^{m} g_t^{(j)}(\theta_{t-1})\right)$;

Working machine j:

1: Compute the gradient $\nabla \bar{f}^{(j)}(\theta_{t-1})$;
2: Send $\nabla \bar{f}^{(j)}(\theta_{t-1})$ back to the parameter server;

Assumption
Standard gradient descent

Algorithm 1 Standard Gradient Descent: Iteration $t \geq 1$

Parameter server:

1: *Initialize:* Let θ_0 be an arbitrary point in Θ.
2: Broadcast the current model parameter estimator θ_{t-1} to all working machines;
3: Wait to receive all the gradients reported by the m machines; Let $g_t^{(j)}(\theta_{t-1})$ denote the value received from machine j.
If no message from machine j is received, set $g_t^{(j)}(\theta_{t-1})$ to be some arbitrary value;
4: Update: $\theta_t \leftarrow \theta_{t-1} - \eta \times \left(\frac{1}{m}\sum_{j=1}^{m} g_t^{(j)}(\theta_{t-1})\right)$;

Working machine j:

1: Compute the gradient $\nabla \bar{f}^{(j)}(\theta_{t-1})$;
2: Send $\nabla \bar{f}^{(j)}(\theta_{t-1})$ back to the parameter server;

NOT GOOD!

Assumption
Standard gradient descent

However, a single Byzantine failure can completely skew the average value of the gradients received by the parameter server, and thus foils the algorithm.

Advantages
Byzantine Gradient Descent

In this paper, we address the above challenges by developing a simple variant of the gradient descent method that can

- tolerate the arbitrary and adversarial failures
- accurately learn a highly complex model with low local data volume
- converge exponentially fast using logarithmic communication rounds

Advantages
Byzantine Gradient Descent

In Byzantine gradient descent method, the parameter server aggregates the local gradients reported by the working machines in three steps:

- it partitions all the received local gradients into k batches and computes the mean for each batch
- it computes the geometric median of the k batch means
- it performs a gradient descent step using the geometric median

Geometric median: it has been widely used in robust statistics

$$med\{y_1, ..., y_n\} \triangleq \arg\min_{y \in \mathbb{R}^d} \sum_{i=1}^{n} ||y - y_i||$$

Algorithm 2 Byzantine Gradient Descent: Iteration $t \geq 1$

Parameter server:

1: *Initialize:* Let θ_0 be an arbitrary point in Θ; group the m machines into k batches, with the ℓ-th batch being $\{(\ell-1)b+1, \ldots, \ell b\}$ for $1 \leq \ell \leq k$.

2: Broadcast the current model parameter estimator θ_{t-1} to all working machines;

3: Wait to receive all the gradients reported by the m machines; If no message from machine j is received, set $\nabla \widetilde{g}_j(\theta_{t-1})$ to be some arbitrary value;

4: *Robust Gradient Aggregation*

$$\mathcal{A}_k(\boldsymbol{g}_t(\theta_{t-1})) \leftarrow \text{med}\left\{\frac{1}{b}\sum_{j=1}^{b} g_t^{(j)}(\theta_{t-1}), \cdots, \frac{1}{b}\sum_{j=n-b+1}^{n} g_t^{(j)}(\theta_{t-1})\right\}. \tag{8}$$

5: Update: $\theta_t \leftarrow \theta_{t-1} - \eta \times \mathcal{A}_k\Big(\boldsymbol{g}_t(\theta_{t-1})\Big)$;

Working machine j:

1: Compute the gradient $\nabla \bar{f}^{(j)}(\theta_{t-1})$;

2: Send $\nabla \bar{f}^{(j)}(\theta_{t-1})$ back to the parameter server;

Algorithm 2 Byzantine Gradient Descent: Iteration $t \geq 1$

Parameter server:

1: *Initialize:* Let θ_0 be an arbitrary point in Θ; group the m machines into k batches, with the ℓ-th batch being $\{(\ell-1)b+1, \ldots, \ell b\}$ for $1 \leq \ell \leq k$.
2: Broadcast the current model parameter estimator θ_{t-1} to all working machines;
3: Wait to receive all the gradients reported by the m machines; If no message from machine j is received, set $\nabla\widetilde{g}_j(\theta_{t-1})$ to be some arbitrary value;
4: *Robust Gradient Aggregation*

$$\mathcal{A}_k(\boldsymbol{g}_t(\theta_{t-1})) \leftarrow \text{med}\left\{\frac{1}{b}\sum_{j=1}^{b} g_t^{(j)}(\theta_{t-1}), \cdots, \frac{1}{b}\sum_{j=n-b+1}^{n} g_t^{(j)}(\theta_{t-1})\right\}. \tag{8}$$

5: Update: $\theta_t \leftarrow \theta_{t-1} - \eta \times \mathcal{A}_k\big(\boldsymbol{g}_t(\theta_{t-1})\big)$;

Working machine j:

1: Compute the gradient $\nabla\bar{f}^{(j)}(\theta_{t-1})$;
2: Send $\nabla\bar{f}^{(j)}(\theta_{t-1})$ back to the parameter server;

GOOD!

Result
Byzantine Gradient Desent

To implement the byzantine gradient descent algorithm,we choose:

- Twelve machines, divided into four batches, and one of them is byzantine.
- The truth model parameter is (3,4), and the initialize value is(9,10)
- After 100 itertions, the BGD get(2.70, 3.54), while the GD get(1.62, -9.67)
- We found that the error reduced obviously by BGD

Reference

- https://github.com/rmahfuz/Fault-tolerant-distributed-optimization

Thank you!