Distributed Optimization for Machine Learning

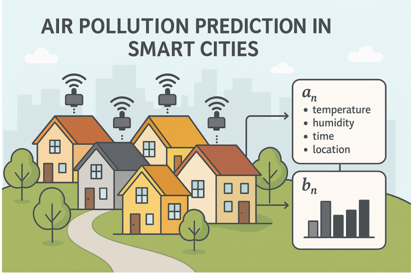
Lecture 14 - Communication-efficient Distributed Training - Part I

Tianyi Chen

School of Electrical and Computer Engineering
Cornell Tech, Cornell University

October 15, 2025

Example: air pollution prediction in smart cities



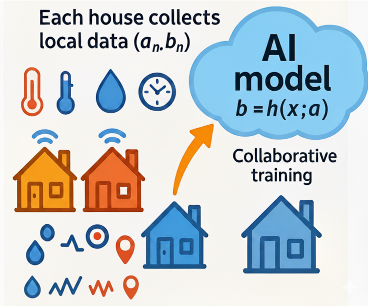
There is a community of multiple houses, where each house has a smart sensor that records environmental information.

Each house collects data pairs $\boldsymbol{\xi}_n = \{\boldsymbol{a}_n, \boldsymbol{b}_n\}$ over time, where:

$$\boldsymbol{a}_n = (\text{temperature, humidity, time, location, etc.})$$

$$\boldsymbol{b}_n = \text{concentration of a particular pollutant.}$$

Motivation of distributed training



Goal: All houses want to collaboratively train a machine learning model to predict future $\boldsymbol{b}$ given $\boldsymbol{a}$:

$$\boldsymbol{b} = h(\boldsymbol{x}; \boldsymbol{a})$$

where $\boldsymbol{x}$ denotes the model parameters to be learned.

Example: next-word prediction on smart keyboards

Each smartphone collects sequences of words typed by the user.

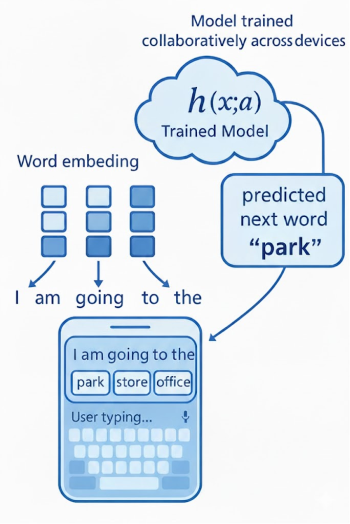
$$\boldsymbol{a}_n = \mathrm{Vec}\begin{pmatrix} \text{current word} \\ \vdots \\ \text{past word} \end{pmatrix},$$

$$\boldsymbol{b}_n = \mathrm{Vec}(\text{next word})\,.$$

Here, $\mathrm{Vec}(\cdot)$ denotes the Word-to-vector embedding operation.

Goal: Learn a model $h(\boldsymbol{x};\boldsymbol{a})$ to predict the next word embedding:

$$\boldsymbol{b} = h(\boldsymbol{x};\boldsymbol{a})$$



Why perform distributed training?

Key question: Why *distributed* data?

Main reason: Privacy!

- Each house may not want to share its raw sensor data with others or with a central server.
- Instead, they exchange only model updates or gradients to preserve local data confidentiality.

Secondary reason: Bandwidth and latency!

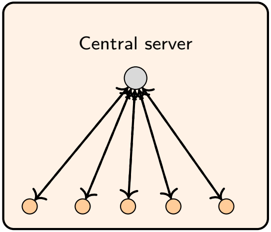
- Reduce communication overhead of transferring large datasets.
- Enable real-time, edge-level learning across smart devices.

Optimization formulation of data parallelism

A network of n nodes (such as mobile devices) collaborate to solve:

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x}), \quad \text{where } \boxed{f_i(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{\xi}_i\sim D_i}[F(\boldsymbol{x};\boldsymbol{\xi}_i)]}$$

- Each component $f_i : \mathbb{R}^d \to \mathbb{R}$ is local and private to node i.
- Random variable $\boldsymbol{\xi}_i$ denotes local data following distribution D_i.
- D_i may be different $\Rightarrow$ **data heterogeneity**.



Local data on nodes

Parallel SGD: compute locally, communicate globally

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x}), \qquad \text{where } f_i(\mathbf{x}) = \mathbb{E}_{\xi_i\sim D_i}[F(\mathbf{x};\xi_i)]\,.$$

PSGD

$$g_i^k = \nabla F\big(\mathbf{x}^k;\, \xi_i^k\big) \qquad \text{(Local compt.)}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \frac{\eta}{n}\sum_{i=1}^{n} g_i^k \qquad \text{(Global comm.)}$$

- Each node i samples mini-batch ξ_i^k and computes $\nabla F\big(\mathbf{x}^k;\xi_i^k\big)$.
- All nodes synchronize (i.e., *globally average*) to update $\mathbf{x}$.

Communication overhead of distributed training

Communication overhead:

- Each entry of a d-dimensional vector (model or gradient) requires 32 bits by default float32 (IEEE 754 single-precision floating-point).
- Each upload or download of the vector incurs:

$$\text{Communication cost} = 32 \times d \times n$$

where

- 32: bits per entry,
- d: number of dimensions ($10^6 \sim 10^{11}$),
- n: number of workers ($10^3 \sim 10^4$).

$\Rightarrow$ Total communication per round $= \mathcal{O}(10^{10}$ to $10^{16})$ bits.

Solutions to overcome communication overhead

Goal: Reduce the total communication cost per iteration:

$$32 \times d \times n$$

Possible solutions:

S1: **Reduce the communication rounds:**
e.g., Local SGD: perform τ local updates before synchronization to reduce communication frequency while maintaining accuracy.

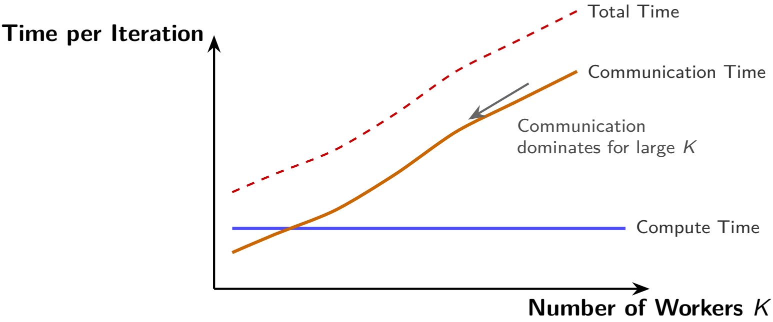
S2: **Reduce the number of bits via quantization/sparsification:**
e.g., Stochastic or deterministic quantization, threshold-based or Top-k sparsification.

S3: **Reduce the number of workers:**
e.g., Randomized / cyclic and adaptive worker selection (LAG).

Table of Contents

Communication bottleneck in Parallel SGD

- In Parallel SGD, workers **synchronize after every step**.
- Comm. dominates runtime when n is large or network is slow.



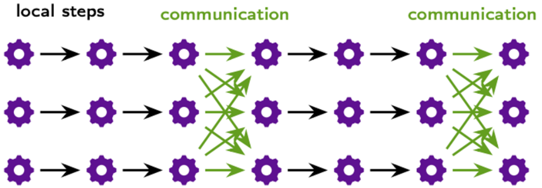
Idea: Local updates before synchronization

Key idea: Each node i performs several SGD steps before averaging.

$$\boldsymbol{x}_i^{(s+1)} = \boldsymbol{x}_i^{(s)} - \eta \nabla F(\boldsymbol{x}_i^{(s)}; \xi_i^{(s)}), \quad s = 0, \dots, \tau - 1$$

where $\boldsymbol{x}_i^{(0)} = \boldsymbol{x}^k$. After every τ steps:

$$\boldsymbol{x}^{k+1} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i^{(\tau)}$$



Benefit: Reduces communication by a factor of τ.

Mini-batch SGD vs. Local SGD

Mini-batch or Parallel SGD:

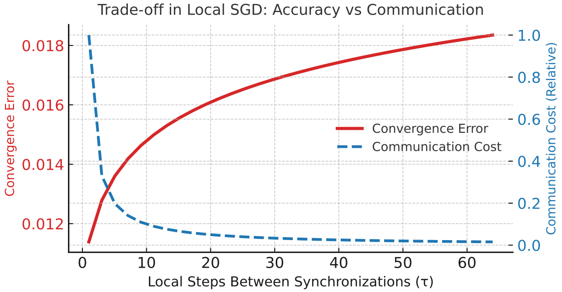
$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t - \eta_t \frac{1}{n\tau} \sum_{i=1}^{n} \sum_{s=1}^{\tau} \nabla F(\boldsymbol{x}^t; \xi_i^{t,s})$$

Local SGD:

$$\boldsymbol{x}_i^{t+1} = \begin{cases} \boldsymbol{x}_i^t - \eta_t \nabla F(\boldsymbol{x}_i^t; \boldsymbol{\xi}_i^t), & t \bmod \tau \neq 0 \\ \frac{1}{n} \sum_{i=1}^{n} \left(\boldsymbol{x}_i^t - \eta_t \nabla F(\boldsymbol{x}_i^t; \boldsymbol{\xi}_i^t) \right), & t \bmod \tau = 0 \end{cases}$$

Method	Mini-batch SGD	Local SGD
# Comm. rounds	K	K
Batch size	$n\tau$	n
# Model updates	K	τK
# Gradient calcs	$n\tau K$	$n\tau K$

Communication vs. computation trade-off



$$\text{Runtime per iteration} = \text{Compute} + \frac{1}{\tau}\text{Comm.}$$

Insight: Increasing τ improves efficiency but risks model drift.

Why Local SGD works under homogeneous data?

If $f(\boldsymbol{x})$ is convex and all workers start synchronized ($\boldsymbol{x}_i^t = \bar{\boldsymbol{x}}^t$):

$$f_i(\boldsymbol{x}_i^{t+\tau}) \leq f_i(\bar{\boldsymbol{x}}^t) - (\text{descent term for worker } i).$$

Thus, synchronization preserves global descent.

$$f(\bar{\boldsymbol{x}}^{t+\tau}) \leq \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x}_i^{t+\tau}) \leq f(\bar{\boldsymbol{x}}^t) - (\text{averaged progress}).$$

Not true in general if f_i differ across workers (non-i.i.d.).

Quadratic objectives: analytical insight

For local quadratic objectives $f_i(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{A}_i \boldsymbol{x} - \boldsymbol{b}_i^\top \boldsymbol{x}$:

$$\boldsymbol{x}_i^{k+1} = \boldsymbol{x}_i^k - \eta_k \nabla f_i(\boldsymbol{x}_i^k).$$

Averaging yields:

$$\mathbb{E}[\bar{\boldsymbol{x}}^{k+1} | \mathcal{F}^k] = \bar{\boldsymbol{x}}^k - \eta_k \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\boldsymbol{x}_i^k) \approx \bar{\boldsymbol{x}}^k - \eta_k \nabla f(\bar{\boldsymbol{x}}^k).$$

Hence, Local SGD mimics global descent dynamics.

Local SGD improves efficiency in quadratic setting

Theorem 1 (Local SGD under smooth and convex loss)

The error bound for Local SGD with τ local updates equals the bound for Mini-batch SGD with batch size n and $K\tau$ rounds:

$$\epsilon_{\text{L-SGD}} := \frac{1}{K}\sum_{k=1}^{K}\mathbb{E}[||\nabla f(\boldsymbol{x}^k)||_2^2] = \boldsymbol{\Theta}\left(\frac{1}{K\tau} + \frac{\sigma}{\sqrt{nK\tau}}\right)$$

- More local updates τ always help convergence.
- Mini-batch SGD: $\epsilon_{\text{MB-SGD}} = \boldsymbol{\Theta}\left(\frac{1}{K} + \frac{\sigma}{\sqrt{nK\tau}}\right)$
- Local SGD *can be better* given the same computation budget.

Performance for general convex objectives*

Upper and lower bounds for Local SGD:

$$\text{Upper:} \qquad \epsilon_{L\text{-}SGD} = \mathcal{O}\left(\frac{\sigma^{2/3}}{K^{2/3}\tau^{1/3}} + \frac{\sigma}{\sqrt{nK\tau}}\right)$$

$$\text{Lower:} \qquad \Omega\left(\frac{\sigma^{2/3}}{K^{2/3}\tau^{2/3}} + \frac{\sigma}{\sqrt{nK\tau}}\right)$$

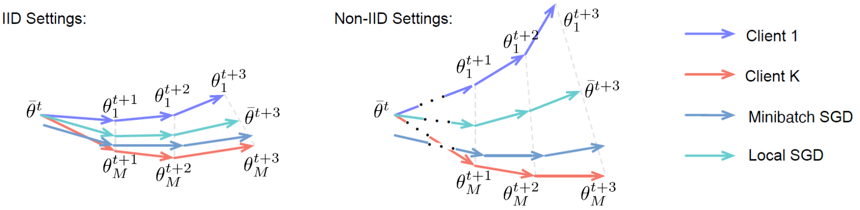
Mini-batch SGD: $\boldsymbol{\Theta}\left(\frac{1}{K} + \frac{\sigma}{\sqrt{nK\tau}}\right)$

Local SGD better when $K \lesssim \tau$, worse when $K \gtrsim \tau$.

Woodworth et al. "Is Local SGD Better than Mini-batch SGD?", *ICML 2020*

Why local SGD may fail under heterogeneous data?

In heterogeneous (non-i.i.d.) data settings, local gradients are misaligned

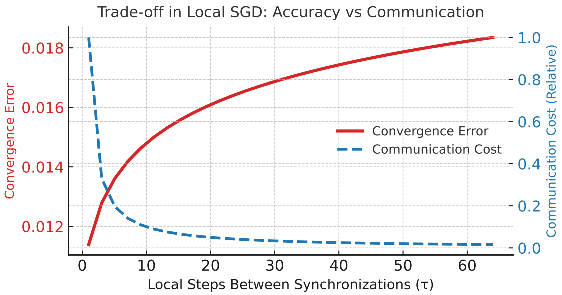


- Local updates diverge due to heterogeneous data (Γ^2).
- Need additional assumptions to control gradient dissimilarity.
- Larger $\tau \Rightarrow$ greater deviation from the global model.

Summary: Parallel SGD vs local SGD

Aspect	**Parallel SGD**	**Local SGD**
Communication	Every iteration	Every τ iterations
Local computation	1 gradient step	τ local steps
Speed	Communication-limited	Compute-efficient
Convergence rate	Stable	Slower (($\eta^2\tau\Gamma^2$) bias)
Best for	Data centers, i.i.d. data	Federated settings

Takeaway: Communication - accuracy trade-off



- $\tau = 1$: fully synchronized (Parallel SGD)
- $\tau > 1$: fewer syncs $\Rightarrow$ faster but drift grows
- Choose τ based on network bandwidth and data heterogeneity

Rule of thumb: $\tau^* \propto \sqrt{\frac{c_{\mathrm{comm}}}{c_{\mathrm{comp}}}}$

When to use local SGD?

Recommended if:

- Communication cost $c_{\mathrm{comm}} \gg c_{\mathrm{comp}}$
- Data across workers are relatively homogeneous
- Occasional synchronization suffices for convergence

Avoid if:

- Highly non-i.i.d. data (strong gradient heterogeneity)
- Models are unstable to small parameter changes

Table of Contents

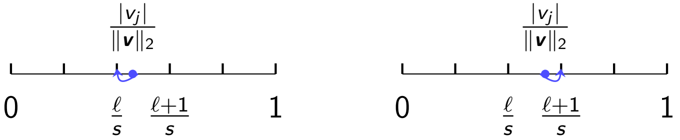
Deterministic quantization

Definition: For any vector $\boldsymbol{v} = [v_1, v_2, \ldots, v_d]^\top \in \mathbb{R}^d$, the j-th entry of the s-level quantized vector $Q_s(\boldsymbol{v})$ is defined as:

$$[Q_s(\boldsymbol{v})]_j := \|\boldsymbol{v}\|_2 \cdot \mathrm{sign}(v_j) \cdot \zeta_j(\boldsymbol{v}, s),$$

Let $0 \leq \ell < s$ be an integer such that $\frac{|v_j|}{\|\boldsymbol{v}\|_2} \in \left[\frac{\ell}{s}, \frac{\ell+1}{s}\right]$. Then:

$$\zeta_j(\boldsymbol{v}, s) = \begin{cases} \frac{\ell}{s}, & \text{if } \frac{|v_j|}{\|\boldsymbol{v}\|_2} - \frac{\ell}{s} \leq \frac{1}{2s}, \\ \frac{\ell+1}{s}, & \text{otherwise} \end{cases}$$

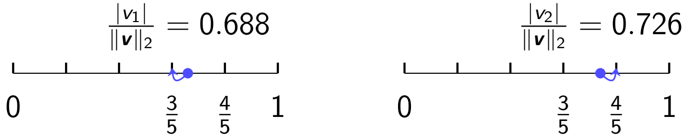


Example: deterministic quantization ($s = 5$)

Example: Consider a 2-D vector $\boldsymbol{v} = [0.36,\, 0.38]$. Let $s = 5$. Its ℓ_2-norm is $\|\boldsymbol{v}\|_2 = \sqrt{0.36^2 + 0.38^2} \approx 0.523$. Thus,

$$\frac{|v_1|}{\|\boldsymbol{v}\|_2} = 0.688, \qquad \frac{|v_2|}{\|\boldsymbol{v}\|_2} = 0.726.$$

Both values fall into the same quantization interval $\left[\frac{3}{5}, \frac{4}{5}\right] = [0.6, 0.8]$.



According to the rule: $[Q_s(\boldsymbol{v})]_j = \|\boldsymbol{v}\|_2 \cdot \mathrm{sign}(v_j) \cdot \zeta_j(\boldsymbol{v}, s)$,
we obtain: $Q_5(\boldsymbol{v}) = 0.523\,[0.6,\, 0.8] = [0.314,\, 0.418]$.

Loss of deterministic quantization

Problem with this strategy: Higher quantization error for values that are further away from the center of the interval.

Lemma. For any vector $\boldsymbol{v} \in \mathbb{R}^d$, we have:

(i) $\|Q_s(\boldsymbol{v}) - \boldsymbol{v}\|_\infty \leq \frac{1}{s}\|\boldsymbol{v}\|_2$ (bias)

(ii) $\|Q_s(\boldsymbol{v}) - \boldsymbol{v}\|_2^2 \leq \frac{d^2}{s}\|\boldsymbol{v}\|_2^2$

Stochastic quantization

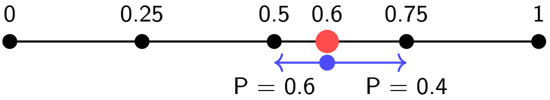
For any vector $\boldsymbol{v} = [v_1, \ldots, v_d]^\top \in \mathbb{R}^d$, the j-th entry of the s-level quantized vector $Q_s(\boldsymbol{v})$ is:

$$[Q_s(\boldsymbol{v})]_j := \|\boldsymbol{v}\|_2 \, \mathrm{sign}(v_j) \, \zeta_j(\boldsymbol{v}, s),$$

where the random variable $\zeta_j(\boldsymbol{v}, s)$ is:

$$\zeta_j(\boldsymbol{v}, s) = \begin{cases} \dfrac{\ell + 1}{s}, & \text{with probability } s\left(\dfrac{|v_j|}{\|\boldsymbol{v}\|_2} - \dfrac{\ell}{s}\right), \\ \dfrac{\ell}{s}, & \text{otherwise} \end{cases}$$

See example for $s = 4$ levels below:



Lemma: properties of stochastic quantization

Lemma: For any vector $\boldsymbol{v} \in \mathbb{R}^d$, if we apply stochastic quantization $Q_s(\boldsymbol{v})$, then we have:

(i) **Unbiasedness:**

$$\mathbb{E}[Q_s(\boldsymbol{v})] = \boldsymbol{v}$$

(ii) **Bounded variance:**

$$\mathbb{E}\left[\|Q_s(\boldsymbol{v}) - \boldsymbol{v}\|_2^2\right] \leq \min\left(\frac{d}{s^2}, \frac{\sqrt{d}}{s}\right) \|\boldsymbol{v}\|_2^2$$

The proof of the second property is given in Appendix 1 of the QSGD paper `https://arxiv.org/pdf/1610.02132`.

Convergence guarantees for QSGD: error bound

If $Q(\boldsymbol{v}_i^{(t)})$ is an unbiased stochastic estimator of $\nabla F_i(\boldsymbol{x}^t)$, then the quantized update is equivalent to a stochastic gradient update, and the standard SGD analysis can be applied.

Theorem 2 (Convergence of QSGD)

Let f be L-smooth and $\eta_k \equiv \eta = 1/\sqrt{K}$. Then the following holds:

$$\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\big[\|\nabla f(\boldsymbol{x}^k)\|_2^2\big] \le \mathcal{O}\left(\frac{\sigma}{\sqrt{nK}}\sqrt{1+\min\left(\frac{d}{s^2}, \frac{\sqrt{d}}{s}\right)}\right).$$

- The error versus iterations convergence becomes worse if we use fewer quantization levels s.

Proof of QSGD error convergence bound

By combining the variance upper bound with the bounded estimation error property of the stochastic quantizer, we have:

$$\mathbb{E}_Q\left[\|Q(g(\boldsymbol{x};\boldsymbol{\xi})) - g(\boldsymbol{x};\boldsymbol{\xi})\|_2^2\right] \le \min\left(\frac{d}{s^2}, \frac{\sqrt{d}}{s}\right) \|g(\boldsymbol{x};\boldsymbol{\xi})\|_2^2$$

$$\Rightarrow \quad \mathbb{E}_Q\left[\|Q(g(\boldsymbol{x};\boldsymbol{\xi}))\|_2^2\right] \le \|g(\boldsymbol{x};\boldsymbol{\xi})\|_2^2 + \min\left(\frac{d}{s^2}, \frac{\sqrt{d}}{s}\right) \|g(\boldsymbol{x};\boldsymbol{\xi})\|_2^2$$

$$\mathbb{E}_\xi\left[\mathbb{E}_Q\left[\|Q(g(\boldsymbol{x};\boldsymbol{\xi}))\|_2^2\right]\right] \le \mathbb{E}_\xi\left[\|g(\boldsymbol{x};\boldsymbol{\xi})\|_2^2\right] + \min\left(\frac{d}{s^2}, \frac{\sqrt{d}}{s}\right) \mathbb{E}_\xi\left[\|g(\boldsymbol{x};\boldsymbol{\xi})\|_2^2\right]$$

$$\mathbb{E}\left[\|Q(g(\boldsymbol{x};\boldsymbol{\xi}))\|_2^2\right] \le \left(1 + \min\left(\frac{d}{s^2}, \frac{\sqrt{d}}{s}\right)\right) \left(\|\nabla F(\boldsymbol{x})\|_2^2 + \sigma^2\right)$$

Implementation of stochastic quantization

After quantization, we transmit $Q_s(\boldsymbol{v})$ instead of the full vector $\boldsymbol{v}$.

The quantized vector $Q_s(\boldsymbol{v})$ can be represented by the tuple:

$$Q_s(\boldsymbol{v}) = \Big(\underbrace{\|\boldsymbol{v}\|_2}_{\text{32 bits}}, \underbrace{\operatorname{sign}(v_j)_{j=1}^d}_{d \text{ bits}}, \underbrace{\zeta_j(\boldsymbol{v}, s)_{j=1}^d}_{d \log_2 s \text{ bits}} \Big)$$

Total:

$$32 + d(1 + \log_2 s) \quad \text{vs.} \quad 32d \text{ (full precision)}$$

Conclusion: Quantization effectively reduces communication cost while introducing a moderate increase in the error versus during convergence.

Recap and fine-tuning

- What we have talked about today?

  ⇒ **Local SGD** reduces communication rounds by allowing each worker to perform multiple local updates before synchronization.

  ⇒ **Quantization** reduces communication cost by transmitting gradients with fewer bits.

Welcome anonymous survey!