18-661: Introduction to ML for Engineers

Distributed SGD

Spring 2025

ECE – Carnegie Mellon University

Outline

1. Stochastic Gradient Descent Convergence
2. Distributed SGD
3. Asynchronous Distributed SGD
4. Local-update SGD
5. From Distributed ML to Federated Learning

Stochastic Gradient Descent Convergence

SGD is at the core of Machine Learning!

We use it to train the model parameters $\mathbf{w}$ in

- Linear Regression: $y = \mathbf{w}^\top \mathbf{x}$
- Logistic Regression: $y = \sigma(\mathbf{w}^\top \mathbf{x})$
- Neural Networks: $y = NN(\mathbf{x}; \mathbf{w})$

For each problem, we define a loss function $F(\mathbf{w})$ to measure the error in the predicted output, and then update $\mathbf{w}$ according to:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta g(\mathbf{w})$$

SGD is at the core of Machine Learning!

For each problem, we define a loss function $F(\mathbf{w})$ to measure the error in the predicted output, and then update $\mathbf{w}$ according to:

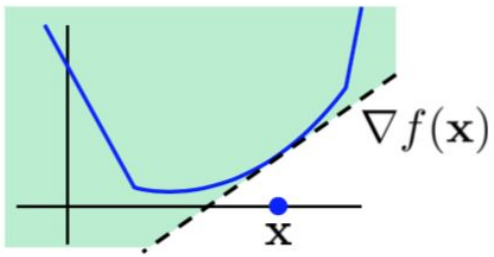
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta g(\mathbf{w})$$

The gradient $g(\mathbf{w})$ can be

- Full gradient, $\nabla F(\mathbf{w}_t) = \frac{1}{N} \sum_{i=1}^{N} \nabla f(\mathbf{w}_t; \xi)$ computed over the whole dataset
- Stochastic gradient $\nabla f(\mathbf{w}_t; \xi)$ for a randomly chosen sample ξ
- Mini-batch stochastic gradient $g(\mathbf{w}; \xi) = \frac{1}{b} \sum_{i=1}^{b} \nabla f(\mathbf{w}; \xi_i)$, computed using a batch ξ of b samples chosen at random

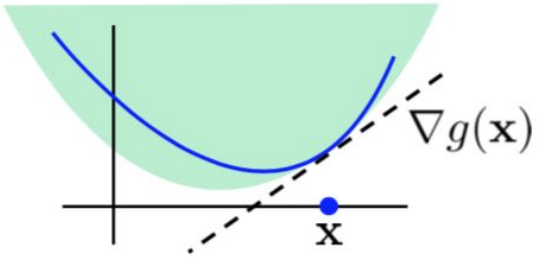
Let us analyze the time SGD takes to reach an ϵ error

A c-strongly Convex and L-Smooth Function



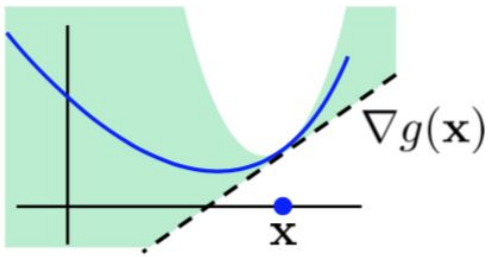
$f : \mathbb{R}^d \to \mathbb{R}$

CONVEX FUNCTION



$g : \mathbb{R}^d \to \mathbb{R}$

STRONGLY CONVEX FUNCTION



$g : \mathbb{R}^d \to \mathbb{R}$

STRONGLY SMOOTH CONVEX FUNCTION

Satisfies the upper and lower bounds given by

$$F(\mathbf{w}) \leq F(\mathbf{y}) + \nabla F(\mathbf{y})^\top (\mathbf{w} - \mathbf{y}) + \frac{L}{2}\|\mathbf{w} - \mathbf{y}\|^2$$

$$F(\mathbf{w}) \geq F(\mathbf{y}) + \nabla F(\mathbf{y})^\top (\mathbf{w} - \mathbf{y}) + \frac{1}{2}c\|\mathbf{w} - \mathbf{y}\|^2 \text{ for all } \mathbf{w}, \mathbf{y} \in \mathbb{R}^d$$

Convergence Analysis of GD

Convergence of GD

For a c-strongly convex and L-smooth function, if the learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_t)$ after t gradient descent iterations is bounded as

$$F(\mathbf{w}_t) - F(\mathbf{w}^*) \leq (1 - \eta c)^t (F(\mathbf{w}_0) - F(\mathbf{w}^*))$$

How many iterations do we need to converge to reach error $F(\mathbf{w}_t) - F(\mathbf{w}^*) = \epsilon$?

$$(1 - \eta c)^t (F(\mathbf{w}_0) - F(\mathbf{w}^*)) \leq \epsilon$$

$$t \log(1 - \eta c) + \log(F(\mathbf{w}_0) - F(\mathbf{w}^*)) \leq \log(\epsilon)$$

$$t \log(1/(1 - \eta c)) - \log(F(\mathbf{w}_0) - F(\mathbf{w}^*) \geq \log(\frac{1}{\epsilon})$$

$$t = O(\log(\frac{1}{\epsilon}))$$

Convergence Analysis of GD

Convergence of GD

For a c-strongly convex and L-smooth function, if the learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_t)$ after t gradient descent iterations is bounded as

$$F(\mathbf{w}_t) - F(\mathbf{w}^*) \leq (1 - \eta c)^t (F(\mathbf{w}_0) - F(\mathbf{w}^*))$$

How does the convergence speed depend on

- Learning rate η – Converges faster for larger η as long as $\eta < \frac{1}{L}$
- Lipschitz smoothness L – Converges faster for smaller L because we can set a higher η
- Strong convexity parameter c – Converges faster for larger c

Convergence Analysis of Mini-batch SGD

Assumptions on the Stochastic Gradients

Since we are using noisy gradients, we need the following assumptions on them

- Unbiased Gradients: The stochastic gradient $\nabla f(\mathbf{w}; \xi)$ is an unbiased estimate of $\nabla F(\mathbf{w})$, that is,

$$\mathbb{E}_{\xi}[\nabla f(\mathbf{w}; \xi)] = \nabla F(\mathbf{w})$$

- Bounded Variance: The stochastic gradient $\nabla f(\mathbf{w}; \xi)$ has bounded variance, that is,

$$\text{Var}(\nabla f(\mathbf{w}; \xi)) \leq \sigma^2$$

which implies that the variance of a mini-batch gradient is:

$$\text{Var}(g(\mathbf{w}; \xi)) \leq \frac{\sigma^2}{b}$$

Convergence Analysis of Mini-batch SGD

Convergence of Mini-batch SGD

For a c-strongly convex and L-smooth function, if the learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_t)$ after t gradient descent iterations is bounded as

$$\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \leq (1 - \eta c)^t \left(\mathbb{E}[F(\mathbf{w}_0)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \right)$$

- For batch GD, as $t \to \infty$, the objective $F(\mathbf{w}_t) \to F(\mathbf{w}^*)$
- For mini-batch SGD, as $t \to \infty$, we will be left with an error floor $\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) \to \frac{\eta L \sigma^2}{2cb}$.
- This is the price that we pay for noisy gradients, that is the variance bound being $\sigma^2 \geq 0$

Effect of Mini-batch Size on the Error Floor

Convergence of Mini-batch SGD

For a c-strongly convex and L-smooth function, if the learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_t)$ after t gradient descent iterations is bounded as

$$\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \leq (1 - \eta c)^t \left(\mathbb{E}[F(\mathbf{w}_0)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2bc} \right)$$

- Recall that
$$g(\mathbf{w}, \xi) = \frac{1}{b} \sum_{n \in S} \nabla f(\mathbf{w}),$$
- And the bounded variance assumption is $\text{Var}(g(\mathbf{w}; \xi)) \leq \sigma^2 / b$
- When we increase the mini-batch size b, the error floor $\frac{\eta L \sigma^2}{2cb}$ reduces.

Effect of Learning Rate on Convergence Speed and Error Floor

Convergence of Mini-batch SGD

For a c-strongly convex and L-smooth function, if the learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_t)$ after t gradient descent iterations is bounded as

$$\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \leq (1 - \eta c)^t \left(\mathbb{E}[F(\mathbf{w}_0)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \right)$$

- The convergence speed, represents by $(1 - \eta c)$ only depends on the strong convexity parameter c and learning rate η, not on the mini-batch size b
- As η increases, the algorithm converges faster
- But, as η increases, the error floor $\frac{\eta L \sigma^2}{2cb}$ also increases

How do we achieve zero error floor?

Convergence of Mini-batch SGD

For a c-strongly convex and L-smooth function, if the learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_t)$ after t mini-batch SGD iterations is bounded as

$$\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \leq (1 - \eta c)^t \left(\mathbb{E}[F(\mathbf{w}_0)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \right)$$

KEY IDEA: Decay the learning rate η

- If $\eta = \eta_0 / t$, then we can show that $\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) \leq O(1/t)$
- Thus, the number of iterations required an error ϵ is $O(1/\epsilon)$
- In contrast, with GD we need only $O(\log(1/\epsilon))$ iterations to reach an ϵ error

Outline

1. Stochastic Gradient Descent Convergence
2. Distributed SGD
3. Asynchronous Distributed SGD
4. Local-update SGD
5. From Distributed ML to Federated Learning

Distributed SGD

Why Distributed SGD?

- For large training datasets, it can be prohibitively slow to conduct training at a single node.
- Solution: Split the dataset across m nodes into partitions D_1, D_2, ... D_m and perform data-parallel distributed training, using an algorithm that is called Synchronous Distributed SGD

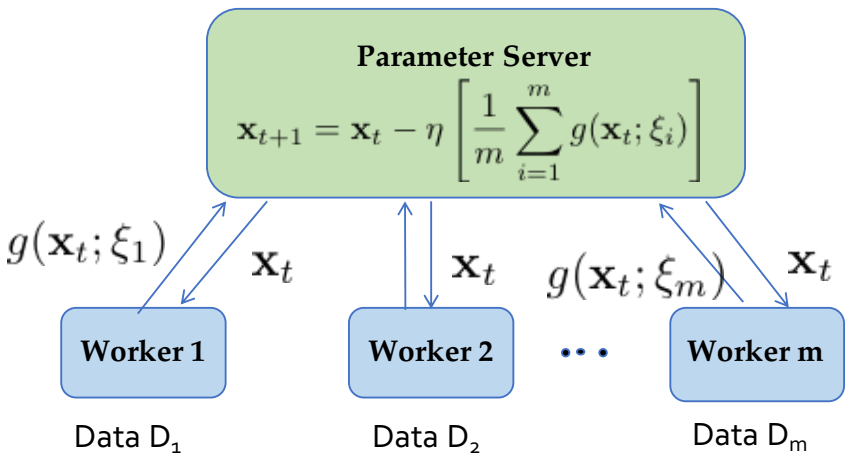


Figure 1: Replace **x** with **w** in this figure

Synchronous Distributed SGD

In each iteration

1. PS sends the current version of the parameter vector $\mathbf{w}_t$ to all the m workers
2. Each worker i computes a gradient $g(\mathbf{w}_t; \xi_i) = \frac{1}{b} \sum_{j=1}^{b} \nabla f(\mathbf{w}_t; \xi_{i,j})$ using a mini-batch of b samples drawn uniformly at random with replacement from its dataset partition D_i
3. The parameter server collects these m mini-batch gradients and updates the parameter vector as: $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{1}{m} \sum_{i=1}^{m} g(\mathbf{w}_t; \xi_i)$

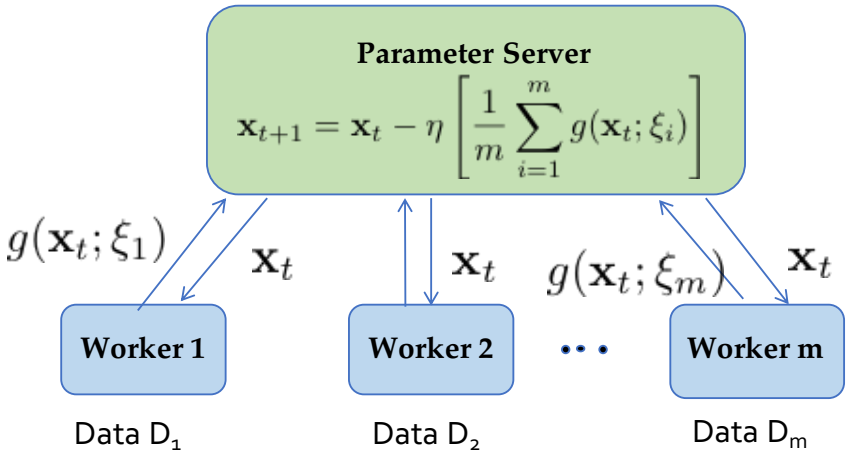


Figure 2: Replace **x** with **w** in this figure

Convergence Analysis of Synchronous SGD

- The update rule $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{1}{m} \sum_{i=1}^{m} g(\mathbf{w}_t; \xi_i)$ is equivalent to performing mini-batch SGD, but with a batch size of *mb* instead of just *b* samples!
- So, the convergence analysis of mini-batch SGD can be directly applied here.
- The only parameter that changes is the variance upper bound.

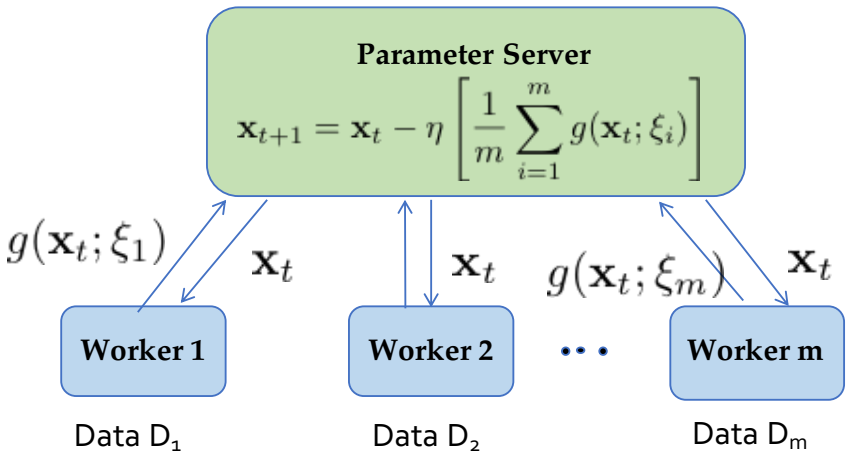


Figure 3: Replace **x** with **w** in this figure

Convergence Analysis of Synchronous SGD

We assume that the aggregated gradient $g(\mathbf{w}_t;\xi) = \frac{1}{m}\sum_{i=1}^{m} g(\mathbf{w};\xi_i) = \frac{1}{mb}\sum_{i=1}^{m}\sum_{j=1}^{b} \nabla f(\mathbf{w};\xi_{i,j})$ satisfies the following assumptions

- Unbiased Gradients: The stochastic gradient $g(\mathbf{w};\xi)$ is an unbiased estimate of $\nabla F(\mathbf{w})$, that is,

$$\mathbb{E}_{\xi}[g(\mathbf{w};\xi)] = \nabla F(\mathbf{w})$$

- Bounded Variance: The stochastic gradient $g(\mathbf{w};\xi)$ has bounded variance, that is,

$$\mathrm{Var}\left(\frac{1}{m}\sum_{i=1}^{m} g(\mathbf{w};\xi_i)\right) \leq \frac{\sigma^2}{mb}$$

where $\mathrm{Var}(\nabla f(\mathbf{w};\xi)) \leq \sigma^2$.

Convergence Analysis of Synchronous SGD

Convergence of Synchronous SGD

For a c-strongly convex and L-smooth objective function, if the learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_t)$ after t synchronous SGD iterations with m workers and mini-batch size b at each worker is bounded as

$$\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2mbc} \leq (1 - \eta c)^t \left(\mathbb{E}[F(\mathbf{w}_0)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2mbc} \right)$$

- For mini-batch SGD, as $t \to \infty$, we will be left with an error floor $\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) \to \frac{\eta L \sigma^2}{2mbc}$.
- When we increase the mini-batch size b, the error floor reduces.
- When we increase the number of workers m, the error floor reduces.

Is it always better to have more worker nodes?

The error versus iterations convergence improves as m increases, but what about the runtime per iteration?

Recall: Synchronous SGD Algorithm

1. Send the current parameter vector $\mathbf{w}$ to all the m workers
2. Each worker i computes a gradient $g(\mathbf{w}; \xi_i) = \frac{1}{b} \sum_{j=1}^{b} \nabla f(\mathbf{w}; \xi_{i,j})$ using a mini-batch of b samples drawn uniformly at random with replacement from its dataset partition D_i
3. The parameter server collects these m mini-batch gradients and updates the parameter vector as: $\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \frac{1}{m} \sum_{i=1}^{m} g(\mathbf{w}_t; \xi_i)$

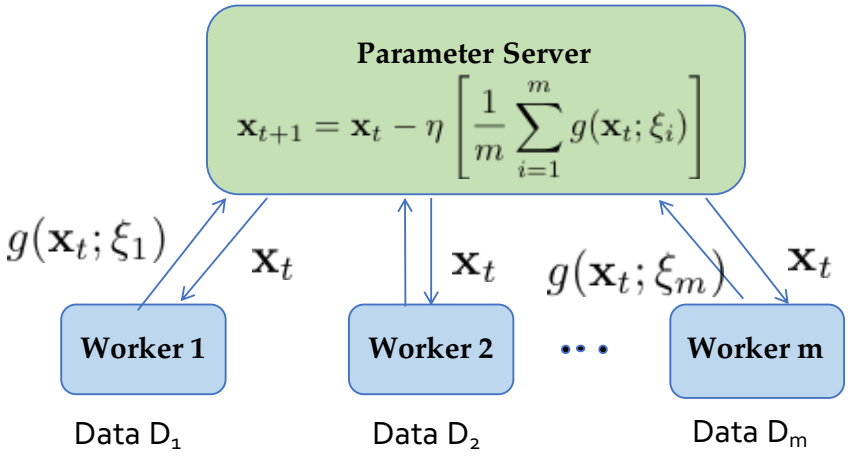


Figure 4: Replace **x** with **w** in this figure

Need to consider how m affects the runtime per iteration

Recall: Synchronous SGD Algorithm

1. Send the current parameter vector $\mathbf{w}_t$ to all the m workers
2. Each worker i computes a gradient $g(\mathbf{w}_t; \xi_i) = \frac{1}{b} \sum_{j=1}^{b} \nabla f(\mathbf{w}; \xi_{i,j})$ using a mini-batch of b samples drawn uniformly at random with replacement from its dataset partition D_i
3. The parameter server collects these m mini-batch gradients and updates the parameter vector as: $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{1}{m} \sum_{i=1}^{m} g(\mathbf{w}_t; \xi_i)$

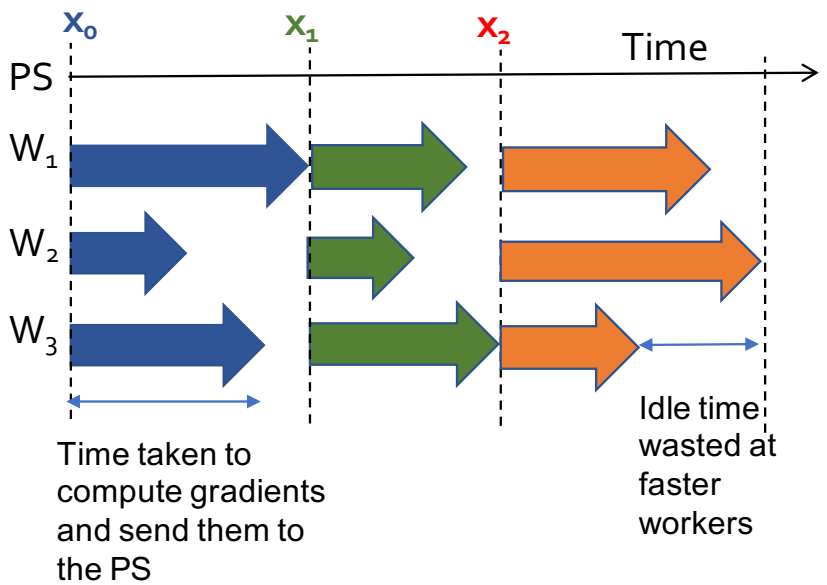


Figure 5: Replace **x** with **w** in this figure

Recall: Synchronous SGD Algorithm

The time taken by a worker to compute and send its mini-batch gradient to the PS can vary randomly across workers and iterations due to:

- Fluctuations in the worker's gradient computation speed
- The gradient computation time can vary across mini-batches
- Network outages can delay communication of the gradients

As the number of workers m increases, the expected time to wait for the slowest worker is longer than waiting for a specific worker

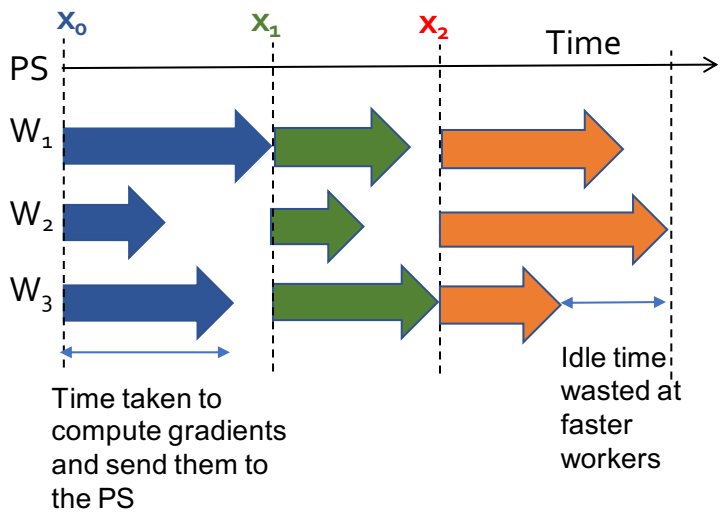


Figure 6: Replace **x** with **w** in this figure

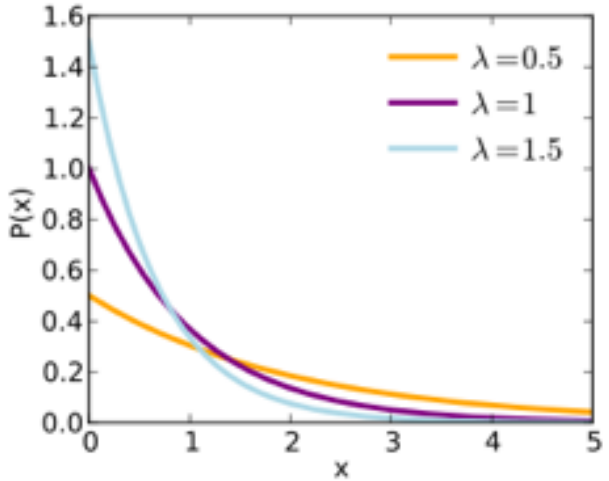
Let us discuss some order statistics concepts, and then use them to analyze the runtime per iteration

Exponential Random Variable

Probability density function of an exponential r.v. with rate λ is

$$f_X(x) = \lambda e^{-\lambda x} \text{ for all } x \geq 0$$

It is the continuous analogue of the geometric r.v.



Questions

- $\mathbb{E}[X] = \frac{1}{\lambda}$
- $\text{Var}[X] = \frac{1}{\lambda^2}$
- Memoryless property: $\Pr(X > x + s | X > s) = \Pr(X > x)$ for all $x, s \geq 0$

Order Statistics

- Suppose we have n independent and identically distributed random variables $X_1, X_2, \ldots, X_n$
- Order statistics are obtained by ordering the random variables

$$\underbrace{X_{1:n}}_{=\min(X_1,\ldots,X_n)} \leq X_{2:n} \leq \cdots \leq \underbrace{X_{n:n}}_{=\max(X_1,\ldots,X_n)}$$

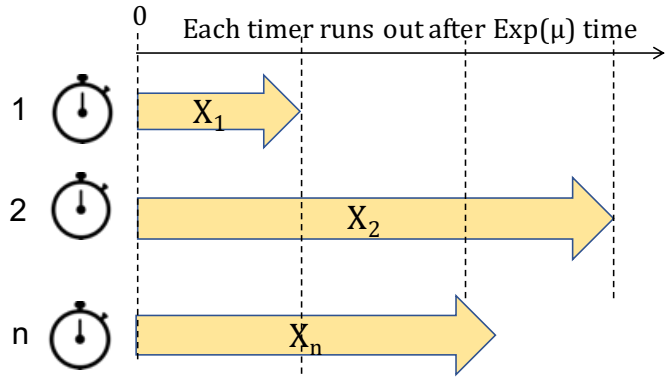
- $X_{k:n}$ is called the k-th order statistic
- Example: If $X_1 = 5$, $X_2 = 8$, and $X_3 = 1$, what is $X_{2:3}$? $= 5$

Order Statistics of the Exponential Distribution

Consider n i.i.d. exponential random variables $X_1, X_2, \ldots X_n$ with rate μ such that

$$f_X(x) = \lambda e^{-\mu x} \text{ for all } x \geq 0$$

- What is $\mathbb{E}[X_{1:n}] = \mathbb{E}[\min(X_1, \ldots, X_n)]$?
- What is $\mathbb{E}[X_{n:n}] = \mathbb{E}[\max(X_1, \ldots, X_n)]$?
- What is $\mathbb{E}[X_{k:n}]$?



Order Statistics of the Exponential Distribution

Consider n i.i.d. exponential random variables $X_1, X_2, \ldots X_n$ with rate μ such that

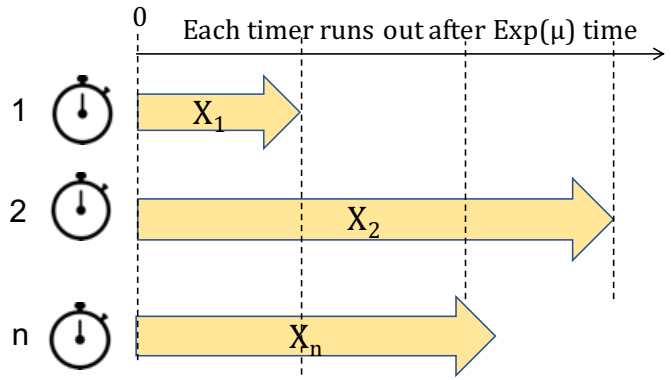
$$f_X(x) = \lambda e^{-\mu x} \text{ for all } x \geq 0$$

- What is $\mathbb{E}[X_{1:n}] = \mathbb{E}[\min(X_1, \ldots, X_n)]$?

  $$\Pr(\min(X_1, \ldots, X_n) > x) = \Pr(X > x)^n = e^{-n\mu x}$$

  Thus, $\min(X_1, \ldots, X_n)$ is an exponential r.v. with rate $n\mu$

Therefore, $\mathbb{E}[X_{1:n}] = \frac{1}{n\mu}$



Order Statistics of the Exponential Distribution

- What is $\mathbb{E}[X_{n:n}] = \mathbb{E}[\max(X_1, \ldots, X_n)]$?
- First timer runs out in $\mathbb{E}[X_{1:n}] = \frac{1}{n\mu}$ units of time
- Recall the memoryless property of the exponential distribution: $\Pr(X > x + s | X > s) = \Pr(X > x)$ for all $x, s \geq 0$.
- If a timer has already run for s seconds, its remaining time is still exponentially distributed as if it was just started
- Thus, after the first timer runs out, it is as if you started the remaining $n - 1$ timers from scratch. Hence,

$$\begin{aligned}\mathbb{E}[X_{n:n}] &= \mathbb{E}[X_{1:n}] + \mathbb{E}[X_{n-1:n-1}] \\ &= \frac{1}{n\mu} + \mathbb{E}[X_{n-1:n-1}]\end{aligned}$$

Order Statistics of the Exponential Distribution

- What is $\mathbb{E}[X_{n:n}] = \mathbb{E}[\max(X_1, \ldots, X_n)]$?
- After the first timer runs out, it is as if you started the remaining $n-1$ timers from scratch. Hence,

$$\begin{aligned}
\mathbb{E}[X_{n:n}] &= \mathbb{E}[X_{1:n}] + \mathbb{E}[X_{n-1:n-1}] \\
&= \frac{1}{n\mu} + \mathbb{E}[X_{n-1:n-1}] \\
&= \frac{1}{n\mu} + \frac{1}{(n-1)\mu} + \mathbb{E}[X_{n-2:n-2}] \\
&= \frac{1}{n\mu} + \frac{1}{(n-1)\mu} + \cdots + \frac{1}{\mu} \\
&= \frac{1}{\mu}\left(\frac{1}{n} + \frac{1}{(n-1)} + \cdots + \frac{1}{1}\right) \\
&= \frac{H_n}{\mu} \approx \frac{\log n}{\mu}
\end{aligned}$$

Order Statistics of the Exponential Distribution

- Exercise: What is $\mathbb{E}[X_{k:n}]$?
- After the first timer runs out, it is as if you started the remaining $n-1$ timers from scratch. Hence,

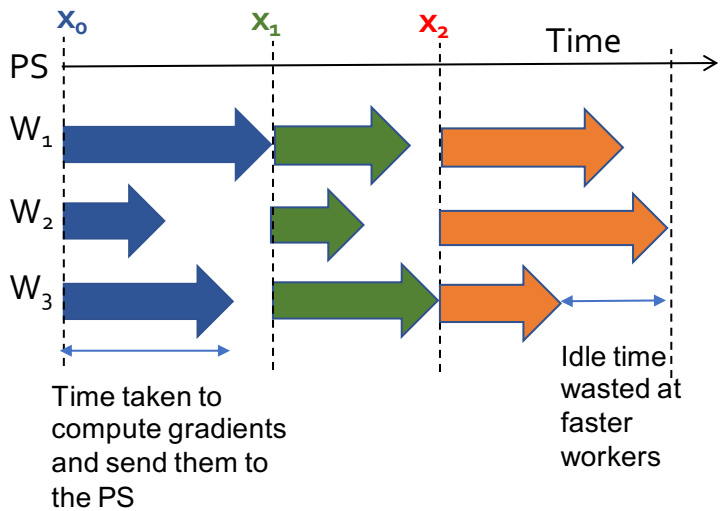
$$\begin{aligned}
\mathbb{E}[X_{k:n}] &= \mathbb{E}[X_{1:n}] + \mathbb{E}[X_{k-1:n-1}] \\
&= \frac{1}{n\mu} + \mathbb{E}[X_{k-1:n-1}] \\
&= \frac{1}{n\mu} + \frac{1}{(n-1)\mu} + \cdots + \frac{1}{(n-k+1)\mu} \\
&= \frac{H_n - H_{n-k}}{\mu} \approx \frac{1}{\mu} \log \frac{n}{(n-k)}
\end{aligned}$$

Now back to analyzing the runtime per iteration of synchronous SGD

Expected Runtime Per Iteration of Synchronous SGD

Suppose the time taken by a worker to compute and send its mini-batch gradient to the PS is a random variable X

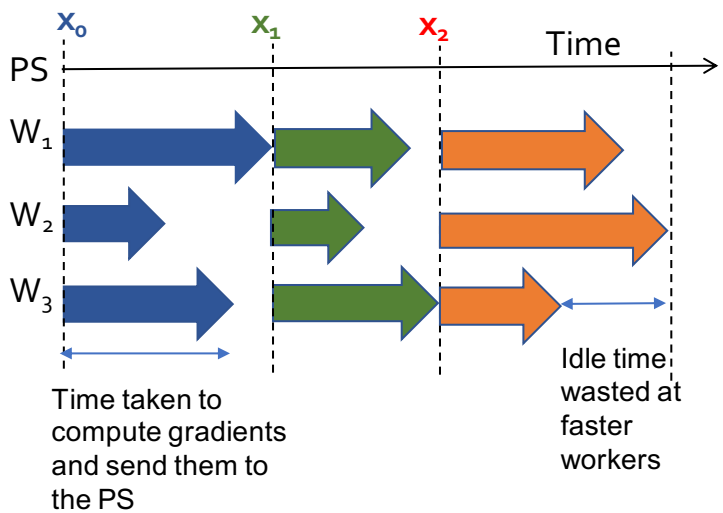
- Ques: What is the expected time taken to complete one distributed SGD iteration?
- Ans: $\mathbb{E}[X_{m:m}]$, which increases with the number of workers m
- Ques: How is $\mathbb{E}[X_{m:m}]$ affected by the variability in X, or more concretely, the probability distribution of X?
- Ans: Higher variability in X results in larger $\mathbb{E}[X_{m:m}]$.



Expected Runtime Per Iteration of Synchronous SGD

Suppose the time taken by a worker to compute and send its mini-batch gradient to the PS is an exponential random variable $X \sim \exp(\mu)$

- Ques: What is the expected time taken by a worker to compute and send its gradient?
- Ans: $\mathbb{E}[X] = \frac{1}{\mu}$
- Ques: What is the expected time taken to complete one distributed SGD iteration?
- Ans: $\mathbb{E}[X_{m:m}] \sim \frac{\log m}{\mu}$, which increases with the number of workers m



Error versus Iterations Convergence of Synchronous SGD

Convergence of Synchronous SGD

For a c-strongly convex and L-smooth objective function, with learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$, $F(\mathbf{w}_t)$ after t sync. SGD iterations with m workers and mini-batch size b is bounded as

$$\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2mbc} \leq (1 - \eta c)^t \left(\mathbb{E}[F(\mathbf{w}_0)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2mbc} \right)$$

Better Convergence when we have more workers m

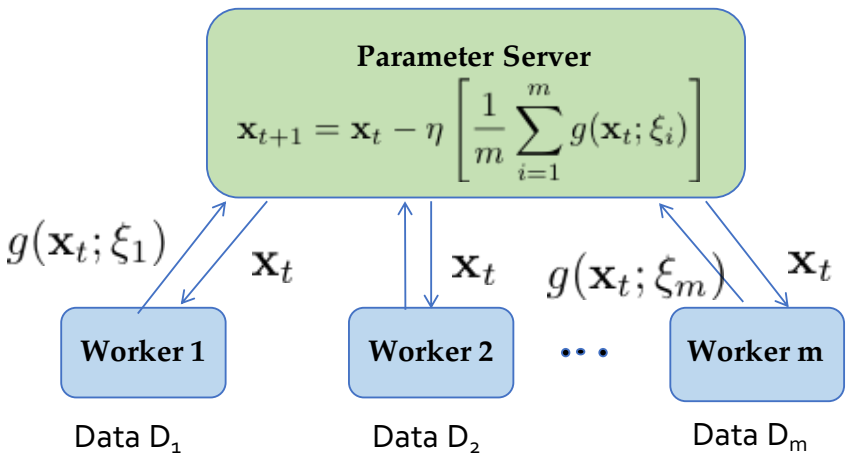
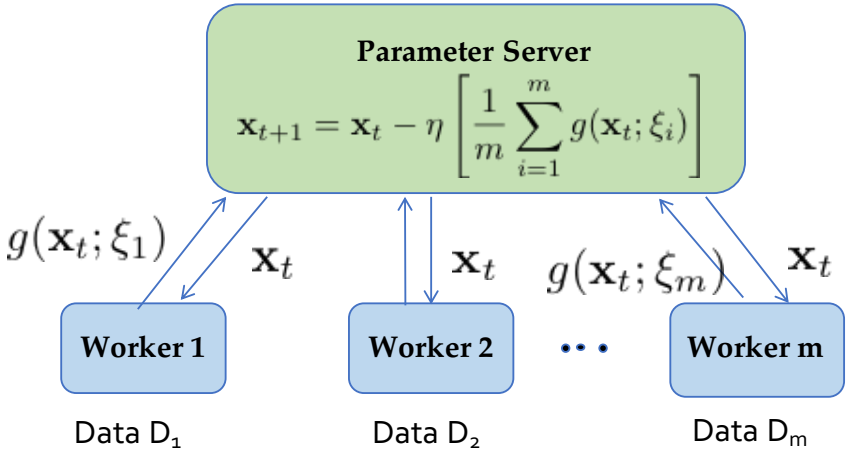


Figure 7: Replace **x** with **w** in this figure

Expected Runtime Per Iteration of Synchronous SGD

Suppose the time taken by a worker to compute and send its mini-batch gradient to the PS is an exponential random variable $X \sim \exp(\mu)$

- Ques: What is the expected time taken to complete one distributed SGD iteration?
- Ans: $\mathbb{E}[X_{m:m}] \sim \frac{\log m}{\mu}$, which increases with the number of workers m



Putting this together with the error versus iterations convergence result, we can get the error versus runtime time convergence

Error Versus Runtime Convergence of Synchronous SGD

Convergence of Synchronous SGD

For a c-strongly convex and L-smooth objective function, with learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_T)$ at time T with m workers and mini-batch size b is bounded as

$$\mathbb{E}[F(\mathbf{w}_T)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2mbc} \lesssim (1 - \eta c)^{\frac{T\mu}{\log m}} \left(\mathbb{E}[F(\mathbf{w}_0)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2mbc} \right)$$

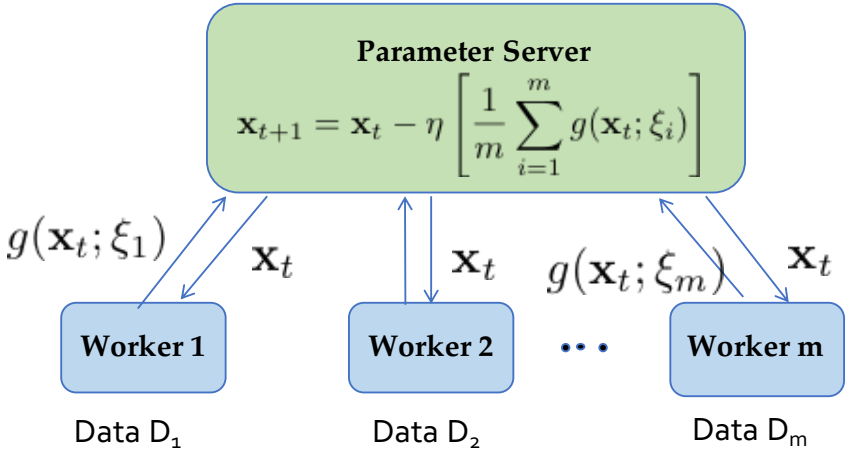


Figure 8: Replace **x** with **w** in this figure

Trade-off between convergence speed and error floor as we vary m

Outline

1. Stochastic Gradient Descent Convergence
2. Distributed SGD
3. Asynchronous Distributed SGD
4. Local-update SGD
5. From Distributed ML to Federated Learning

Asynchronous Distributed SGD

Drawbacks of synchronous SGD and its variants

- Gradients already computed by workers are discarded
- Faster workers can have idle time

What if we remove the synchronization barrier altogether and run asynchronous distributed SGD?

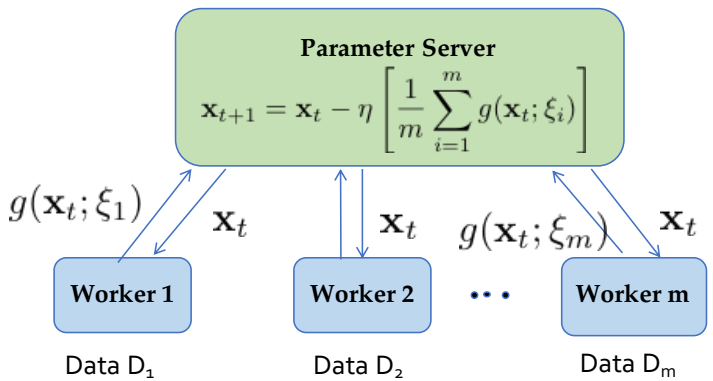


Figure 9: Replace **x** with **w** in this figure

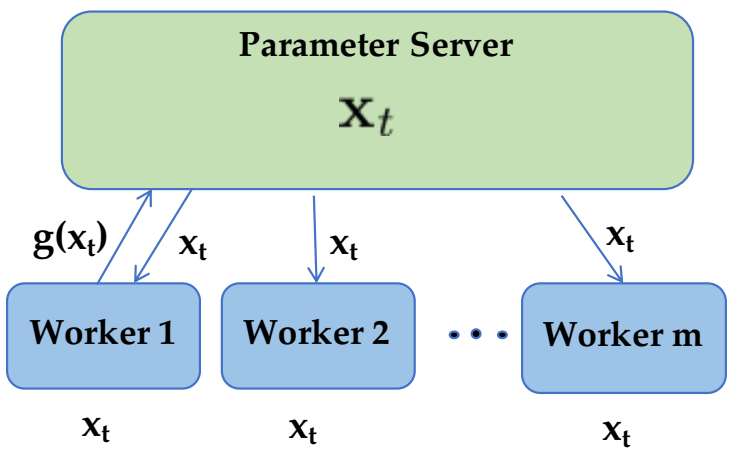
Asynchronous Distributed SGD

Each worker asynchronously does the following:

1. Pulls the current version $\mathbf{w}_t$ of the model
2. Computes a mini-batch gradient $g(\mathbf{w}_t)$ and sends it to the PS

Each time the PS receives a gradient $g(\mathbf{w}_{\tau(t)})$ where $\tau(t) < t$ from a worker it updates the model as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta g(\mathbf{w}_{\tau(t)}; \xi_i)$$



Asynchronous Distributed SGD

Each worker asynchronously does the following:

1. Pulls the current version $\mathbf{w}_t$ of the model
2. Computes a mini-batch gradient $g(\mathbf{w}_t)$ and sends it to the PS

Each time the PS receives a gradient $g(\mathbf{w}_{\tau(t)})$ where $\tau(t) < t$ from a worker it updates the model as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta g(\mathbf{w}_{\tau(t)}; \xi_i)$$

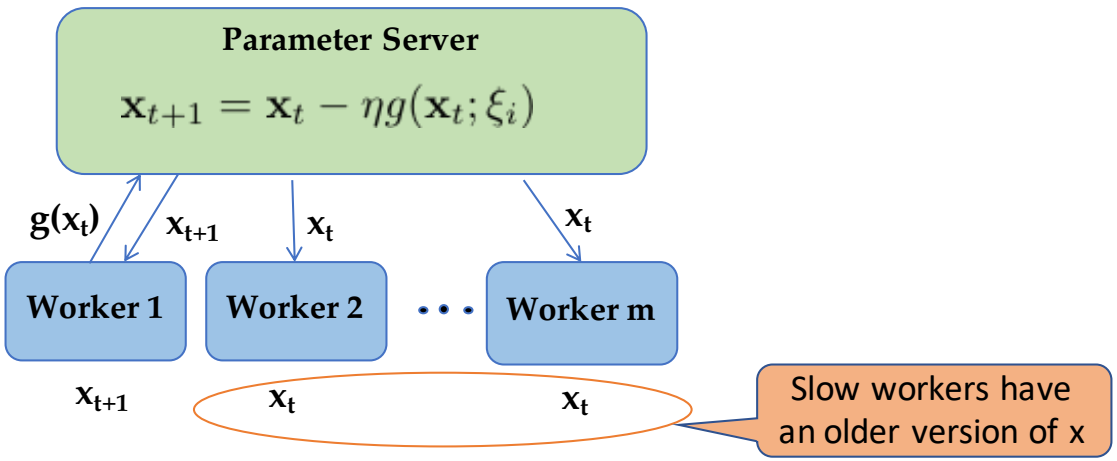


Figure 10: Replace **x** with **w** in this figure

Asynchronous Distributed SGD

Each worker asynchronously does the following:

1. Pulls the current version $\mathbf{w}_t$ of the model
2. Computes a mini-batch gradient $g(\mathbf{w}_t)$ and sends it to the PS

Each time the PS receives a gradient $g(\mathbf{w}_{\tau(t)})$ where $\tau(t) < t$ from a worker it updates the model as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta g(\mathbf{w}_{\tau(t)}; \xi_i)$$

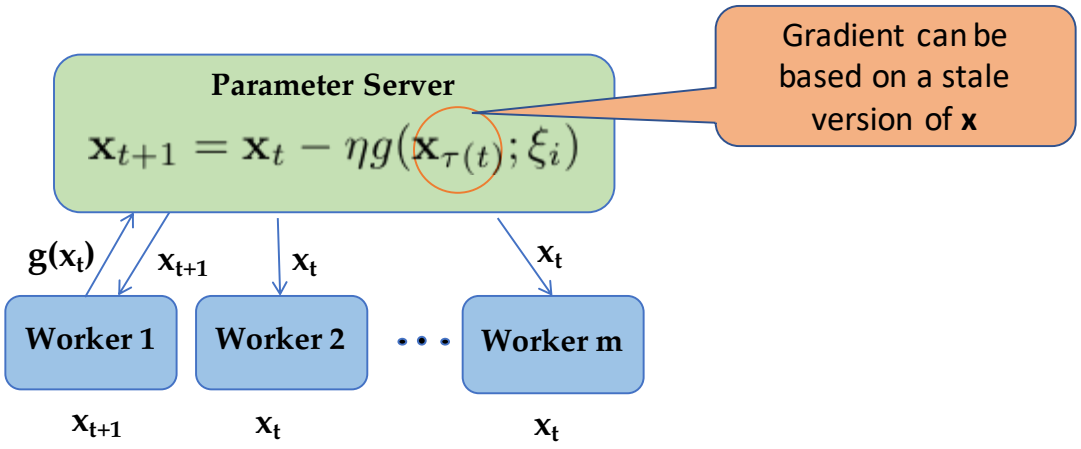


Figure 11: Replace **x** with **w** in this figure

Runtime Analysis of Asynchronous SGD

Suppose each gradient computation takes time $X \sim exp(\mu)$

- The parameter vector **w** is updated every time any one of the m workers returns a gradient
- What is expected time between two successive gradients received by the PS?

$$\mathbb{E}[T_{async}] = \mathbb{E}[X_{1:m}]$$
$$= \frac{1}{m\mu}$$

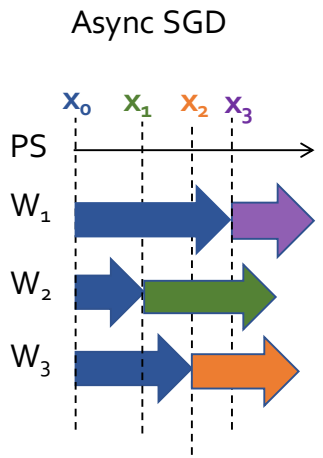
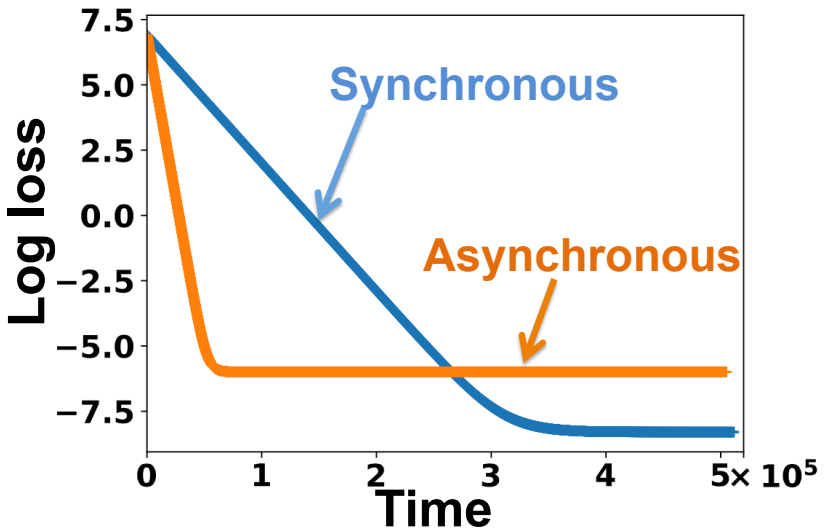


Figure 12: Replace **x** with **w** in this figure

Error-Runtime Trade-off Comparison of Sync and Async SGD

- Asynchronous SGD is much faster because the runtime per iteration is smaller than that of synchronous SGD
- But asynchronous SGD reaches a higher error floor



Are synchronous/asynchronous SGD used in practice?

- Yes! Synchronous/asynchronous SGD are at the backbone of industrial ML implementations today
- One of the earliest industrial implementations is Google's DistBelief framework described in this paper: `https://static.googleusercontent.com/media/research.google.com/en//archive/large_deep_networks_nips2012.pdf`.
- Distbelief popularized the data-parallel and model-parallel parameter server framework. Their algorithm Downpour SGD is based on asynchronous SGD

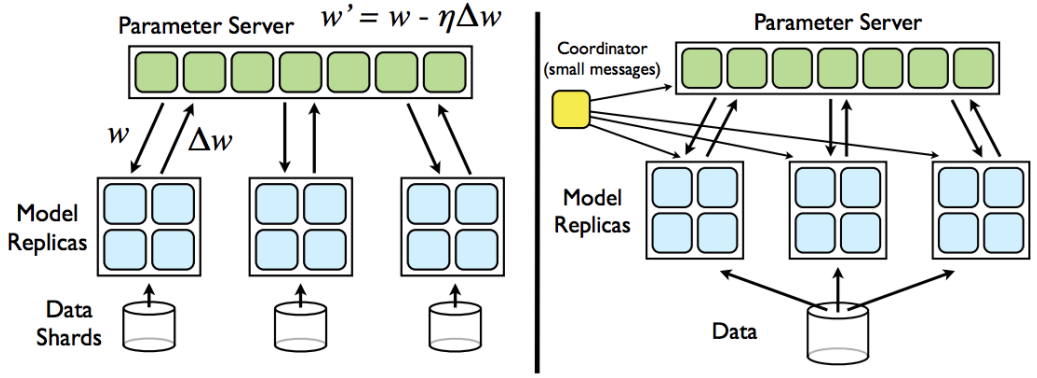


Figure 2: Left: Downpour SGD. Model replicas asynchronously fetch parameters w and push gradients Δw to the parameter server. Right: Sandblaster L-BFGS. A single 'coordinator' sends small messages to replicas and the parameter server to orchestrate batch optimization.

Are synchronous/asynchronous SGD used in practice?

- More recently, specialized hardware such as Tensor Processing Units (TPUs) removes straggling and synchronization delays.
- Thus, industrial implementations are moving from asynchronous SGD back to synchronous SGD
- But asynchronous SGD is still a useful paradigm when you are using cheap consumer hardware and low-bandwidth networks

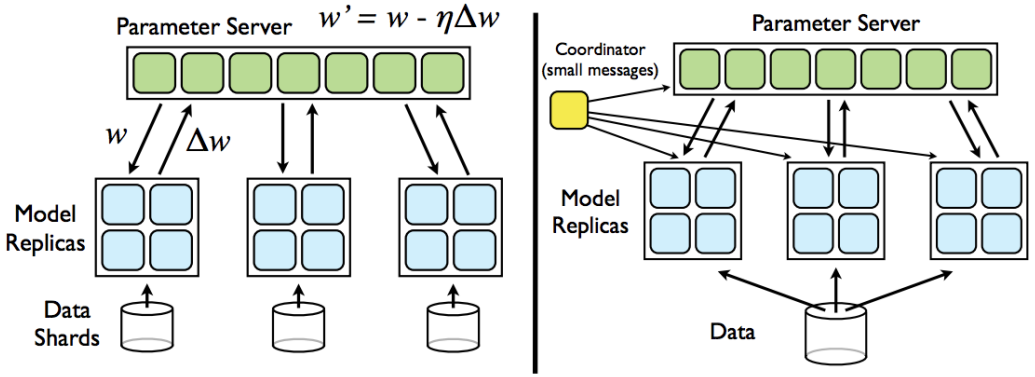


Figure 2: Left: Downpour SGD. Model replicas asynchronously fetch parameters w and push gradients Δw to the parameter server. Right: Sandblaster L-BFGS. A single 'coordinator' sends small messages to replicas and the parameter server to orchestrate batch optimization.

Outline

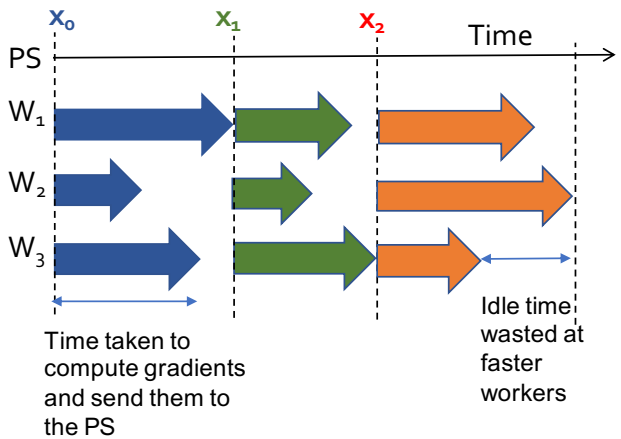
1. Stochastic Gradient Descent Convergence
2. Distributed SGD
3. Asynchronous Distributed SGD
4. Local-update SGD
5. From Distributed ML to Federated Learning

Local-update SGD

Recall: Synchronous SGD Algorithm

In each iteration

1. Send the current parameter vector $\mathbf{w}_t$ to all the m workers
2. Each worker i computes a gradient $g(\mathbf{w}_t; \xi_i) = \frac{1}{b} \sum_{j=1}^{b} \nabla f(\mathbf{w}; \xi_{i,j})$ using a mini-batch of b samples drawn uniformly at random with replacement from its dataset partition D_i
3. The parameter server collects these m mini-batch gradients and updates the parameter vector as: $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{1}{m} \sum_{i=1}^{m} g(\mathbf{w}_t; \xi_i)$



In this picture, we don't account for the gradient communication delays

Communication Delay Per Iteration

The pink colored communication delay block includes

- Time taken to receive gradients $g(\mathbf{w}_t)$ from the workers
- Time taken to update $\mathbf{w}_t$ to $\mathbf{w}_{t+1}$ (this is typically small)
- Time taken to send the updated $\mathbf{w}_{t+1}$ back to the workers

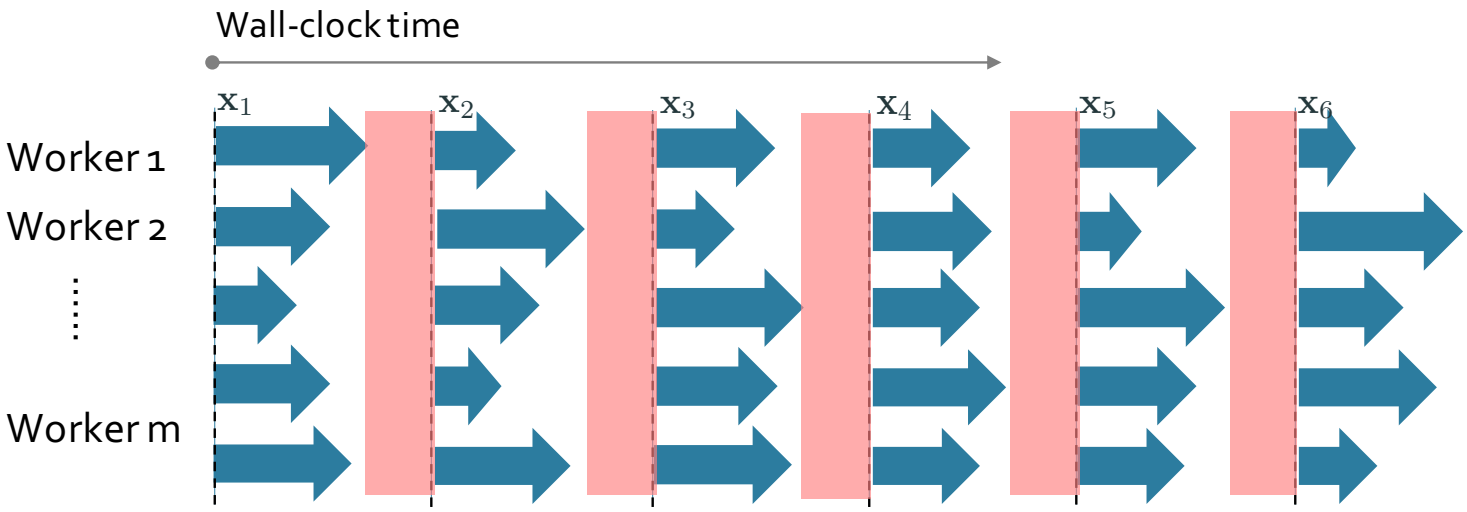
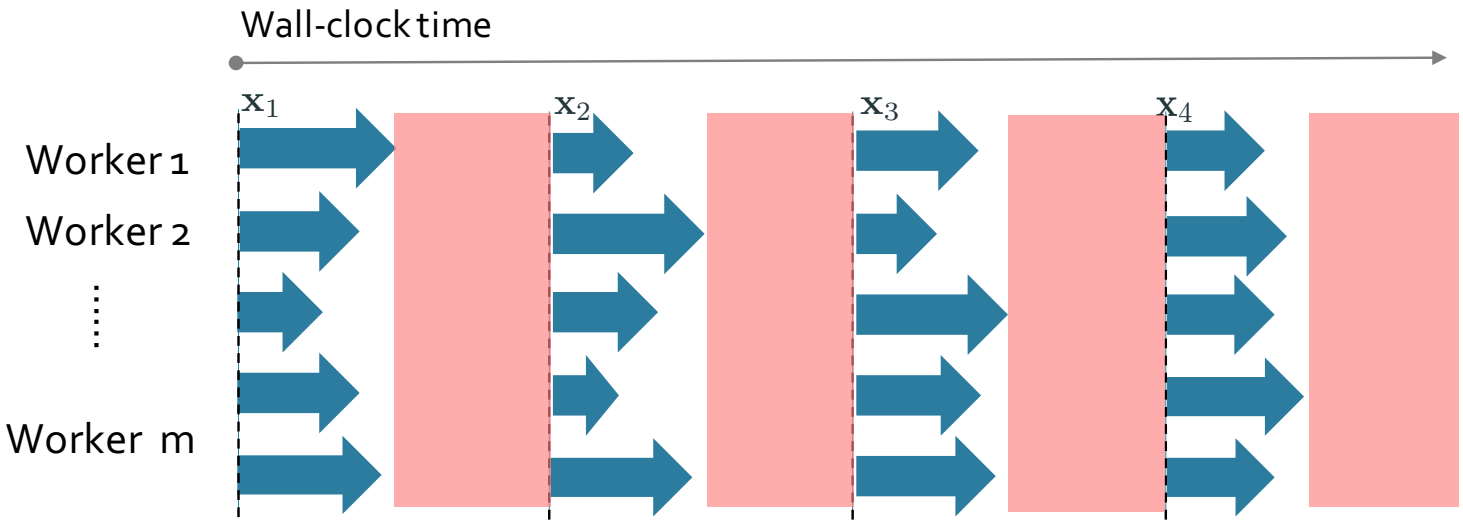


Figure 13: Replace **x** with **w** in this figure

Communication Delay Per Iteration

The pink colored communication delay block includes

- Time taken to receive gradients $g(\mathbf{w}_t)$ from the workers
- Time taken to update $\mathbf{w}_t$ to $\mathbf{w}_{t+1}$ (this is typically small)
- Time taken to send the updated $\mathbf{w}_{t+1}$



The communication delay increases with the size of the model, that is, the size of vector $\mathbf{w}$

Solution: Perform Multiple Local Updates

- Perform τ local updates at each worker and then send the updated model to the parameter server for aggregation
- By reducing the frequency of communication, the communication delay gets amortized over τ iterations
- Can finish many more SGD iterations in the same time

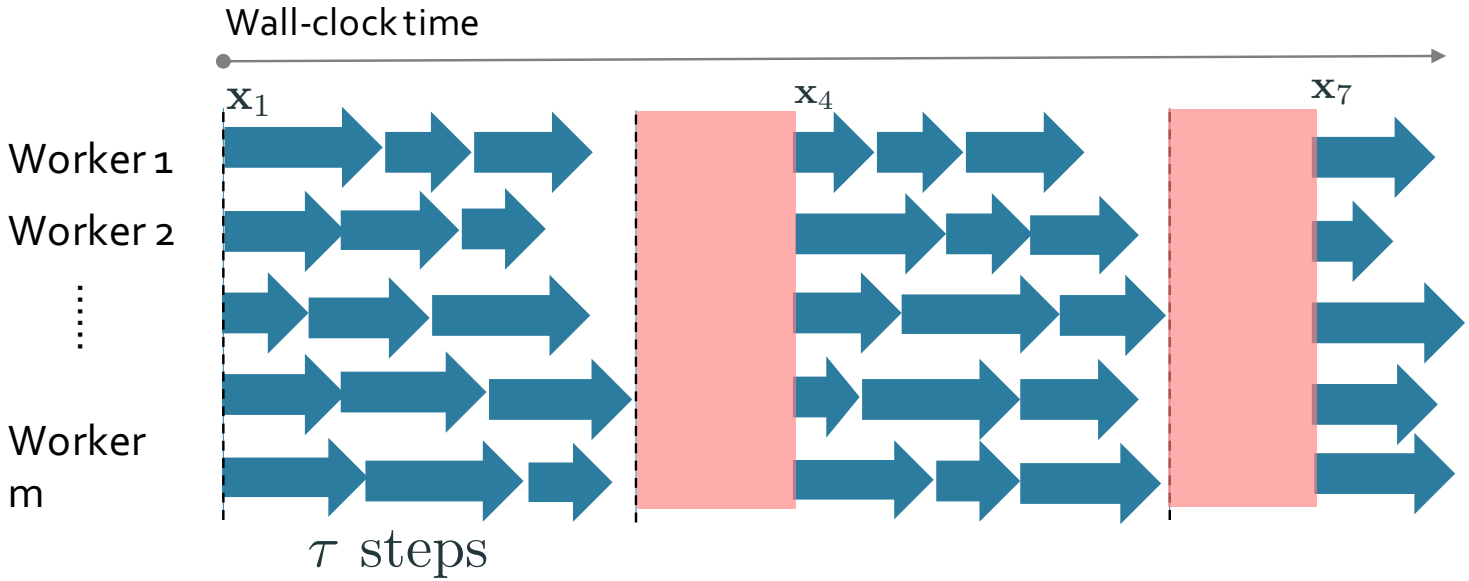


Figure 14: Replace **x** with **w** in this figure

The Local Update SGD Algorithm

The local update SGD algorithm operates in 'communication rounds' or 'training rounds'. In each round, it does the following:

- Worker i fetches the current version of the model $\mathbf{w}_k$ from the parameter server
- It performs τ local SGD updates using its local data D_i
- Due to difference in D_i across workers, the resulting models $\mathbf{w}_{k+\tau}^{(i)}$ will be different
- The resulting models $\mathbf{w}_{k+\tau}^{(i)}$ are aggregated by the parameter server as per

$$\mathbf{w}_{k+\tau} = \sum_{i=1}^{m} \frac{\mathbf{w}_{k+\tau}^{(i)}}{m}$$

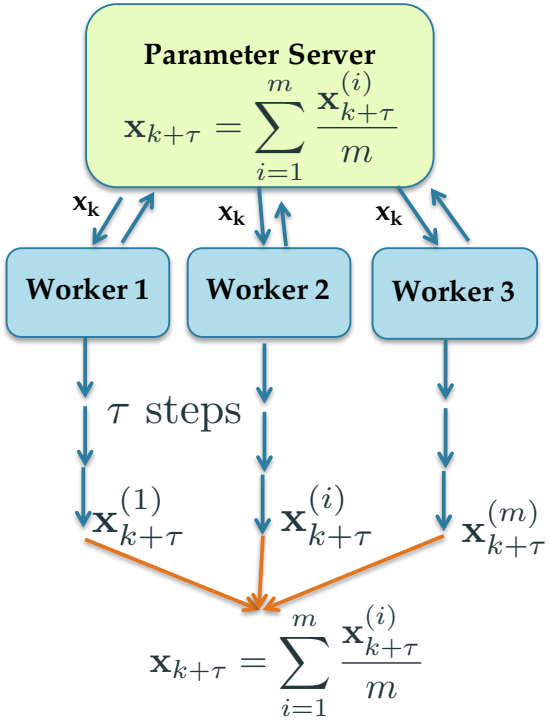


Figure 15: Replace **x** with **w** in this figure

The Local Update SGD Algorithm

- Thus, the effective update rule of local update SGD at the i-th worker is given by

$$\mathbf{w}_{k+1}^{(i)} = \begin{cases} \frac{1}{m}\sum_{j=1}^{m}[\mathbf{w}_k^{(j)} - \eta g(\mathbf{w}_k^{(j)}; \xi_k^{(j)})], & k \bmod \tau = 0 \\ \mathbf{w}_k^{(i)} - \eta g(\mathbf{w}_k^{(i)}; \xi_k^{(i)}), & \text{otherwise} \end{cases}$$

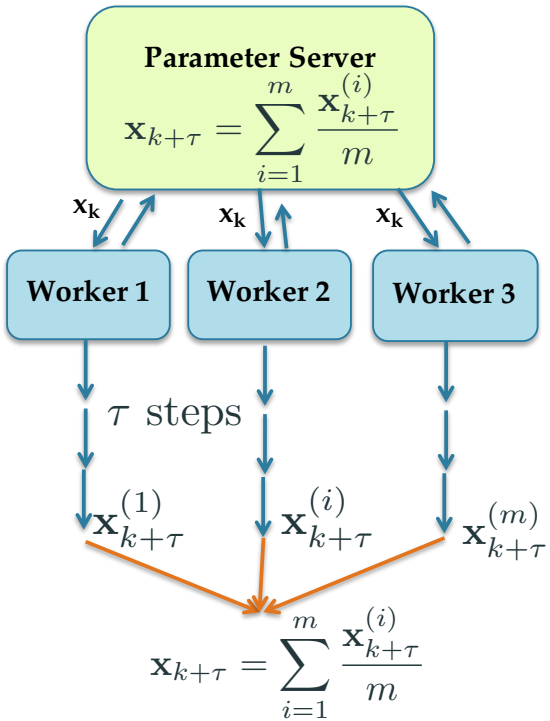
- When the communication period $\tau = 1$ this reduces to the update rule of fully synchronous SGD

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \left[\frac{1}{m} \sum_{i=1}^{m} g(\mathbf{w}_k; \xi_k^{(i)}) \right].$$

where $\mathbf{w}_k^{(i)}$ is the model parameter vector at the i-th worker at the end of the k-th iteration, η is the learning rate, and m is the number of workers

How does τ affect error convergence?

- Larger τ can give inferior error convergence because the local models at the nodes diverge from each other
- We are going to obtain an error convergence bound in terms of τ



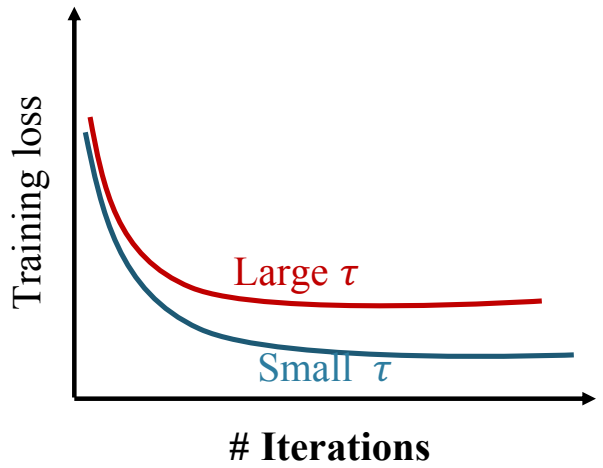


Figure 16: Replace **x** with **w** in this figure

Convergence of Local-update SGD: Assumptions

- Lipschitz Smoothness: The objective function $F(\mathbf{w})$ is differentiable and L-Lipschitz smooth, that is,

$$\|\nabla F(\mathbf{w}) - \nabla F(\mathbf{y})\| \leq L\|\mathbf{w} - \mathbf{y}\|$$

- Lower Bound on F_{inf}: The objective function $F(\mathbf{w})$ is bounded below by F_{inf}, that is,

$$F(\mathbf{w}) \geq F_{\text{inf}} \text{ for all } \mathbf{w}$$

- Unbiased Gradients: The stochastic gradient $g(\mathbf{w}; \xi)$ is an unbiased estimate of $\nabla F(\mathbf{w})$, that is,

$$\mathbb{E}_{\xi}[g(\mathbf{w}; \xi)] = \nabla F(\mathbf{w})$$

- Bounded Variance: The mini-batch stochastic gradient $g(\mathbf{w}; \xi)$ has bounded variance, that is,

$$\text{Var}(g(\mathbf{w}; \xi)) \leq \frac{\sigma^2}{b}$$

$$\mathbb{E}_{\xi}[\|g(\mathbf{w}; \xi)\|^2] \leq \|\nabla F(\mathbf{w})\|^2 + \frac{\sigma^2}{b}$$

Convergence of Local-update SGD: Error Bound

Convergence of Local-update SGD

For a L-smooth function, $\eta L + \eta^2 L^2 \tau(\tau - 1) \leq 1$, and if the starting point is $\mathbf{w}_1$ then $F(\mathbf{w}_t)$ after t iterations of local update SGD is bounded as

$$\mathbb{E}\left[\frac{1}{t}\sum_{k=1}^{t}\|\nabla F(\mathbf{w}_k)\|^2\right] \leq \frac{2\left[F(\mathbf{w}_1) - F_{\text{inf}}\right]}{\eta t} + \frac{\eta L \sigma^2}{mb} + \frac{\eta^2 L^2 \sigma^2 (\tau - 1)}{b}$$

where $\mathbf{w}_k$ denotes the averaged model at the k^{th} iteration.

- We are not assuming strong convexity above – that is why the left hand side is the average of gradients rather than the optimality gap.
- For non-convex functions we can only show convergence towards F_{inf} of the objective function rather than the true optimum F^*.

Convergence of Local-update SGD: Error Bound

Convergence of Local-update SGD

For a L-smooth function, $\eta L + \eta^2 L^2 \tau(\tau - 1) \leq 1$, and if the starting point is $\mathbf{w}_1$ then $F(\mathbf{w}_t)$ after t iterations of local update SGD is bounded as

$$\mathbb{E}\left[\frac{1}{t}\sum_{k=1}^{t}\|\nabla F(\mathbf{w}_k)\|^2\right] \leq \frac{2\left[F(\mathbf{w}_1) - F_{\text{inf}}\right]}{\eta t} + \frac{\eta L \sigma^2}{mb} + \frac{\eta^2 L^2 \sigma^2 (\tau - 1)}{b}$$

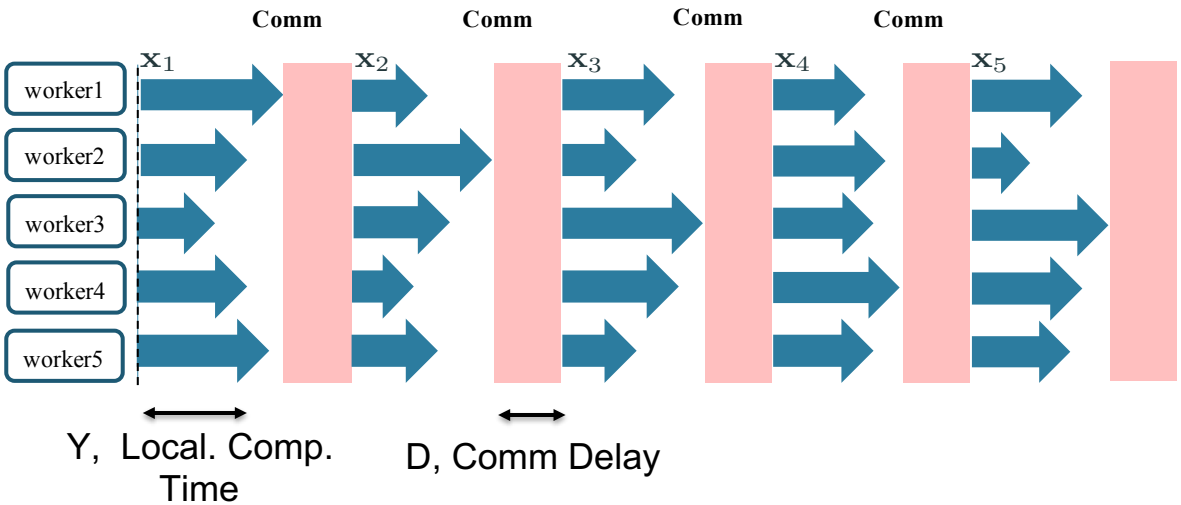
where $\mathbf{w}_k$ denotes the averaged model at the k^{th} iteration.

- When the communication period $\tau = 1$, this bound reduces the convergence bound for mini-batch SGD
- As τ increases, the last term in the error floor grows, corroborating our intuition that the error floor increases with τ

Runtime Per Iteration of Synchronous SGD

- Communication delay is modeled by a random variable D
- Time to perform 1 local update is modeled by random variable Y and assumed to be i.i.d across workers and mini-batches
- Expected Runtime Per Iteration is:

$$\mathbb{E}[T_{sync}] = \mathbb{E}[Y_{m:m}] + \mathbb{E}[D]$$



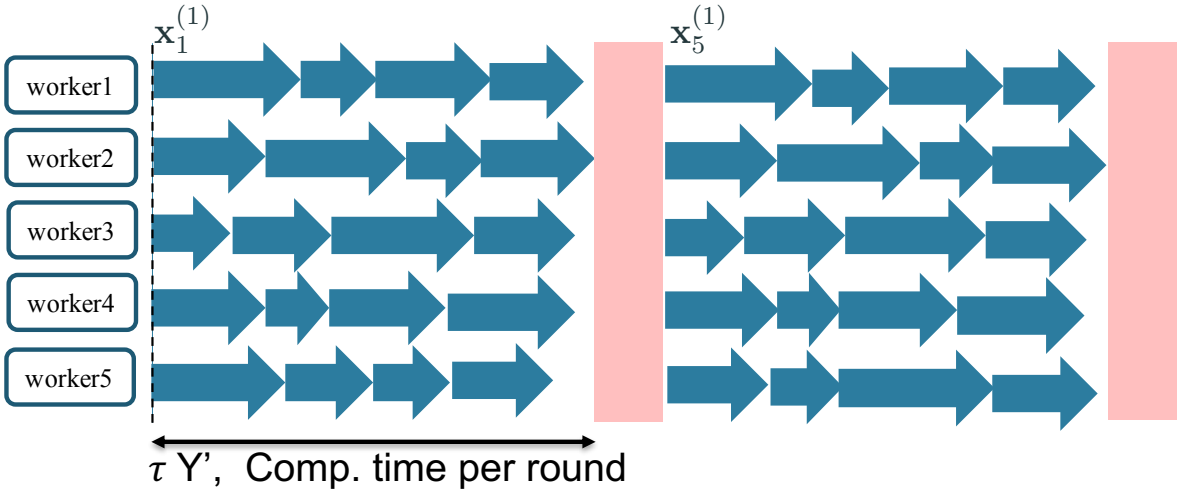
Runtime Per Iteration of Local-update SGD

- Time to perform 1 local update is modeled by random variable Y and assumed to be i.i.d across workers and mini-batches
- Total local computation time per round is

$$T_{comp} = \tau Y' = (Y_1 + Y_2 + \cdots + Y_\tau)$$

$$Y' = \frac{(Y_1 + Y_2 + \cdots + Y_\tau)}{\tau}, \text{average of } \tau \text{ i.i.d. realizations of } Y$$

- Q1: What is the expected value of Y'? $\mathbb{E}[Y'] = \mathbb{E}[Y]$
- Q2: What is the variance of Y'? $Var[Y'] = Var[Y]/\tau$
- Y' has the same mean but a lower variance than Y



Runtime Per Iteration of Local-update SGD

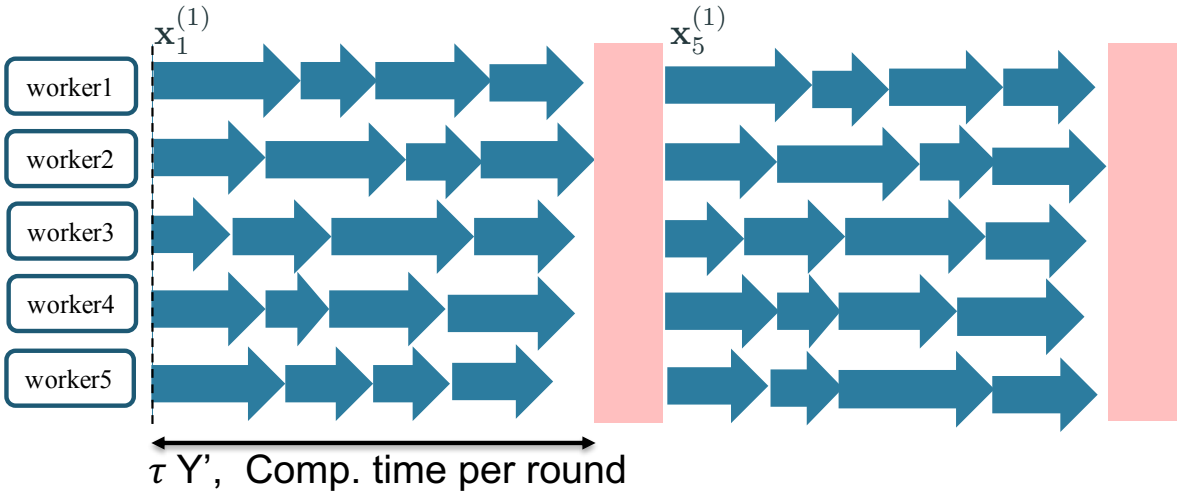
- Expected Runtime Per Iteration is:

$$\mathbb{E}[T_{local}] = \mathbb{E}[Y'_{m:m}] + \frac{\mathbb{E}[D]}{\tau}$$

- Let us compare this with the runtime of synchronous SGD

$$\mathbb{E}[T_{sync}] = \mathbb{E}[Y_{m:m}] + \mathbb{E}[D]$$

- The expected runtime is reduced in two ways:
  - the comm. delay $\mathbb{E}[D]$ gets divided by τ
  - $\mathbb{E}[Y'_{m:m}]$ is smaller than $\mathbb{E}[Y_{m:m}]$ because Y' has lower variance – straggler mitigation

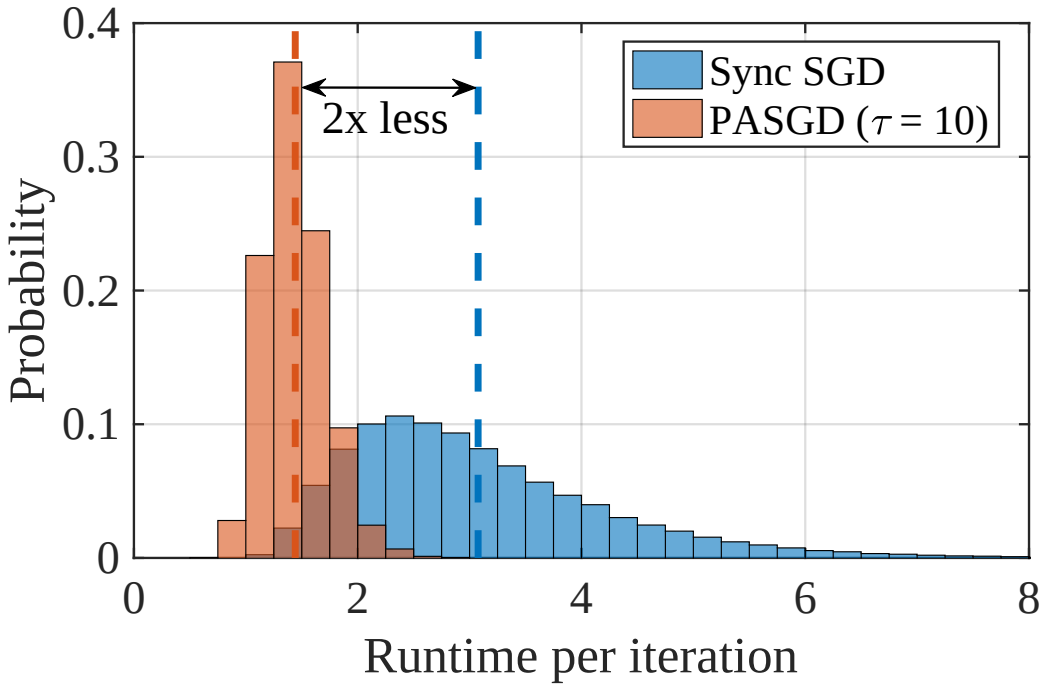


Runtime Per Iteration of Local-update SGD

- Expected Runtime Per Iteration is:

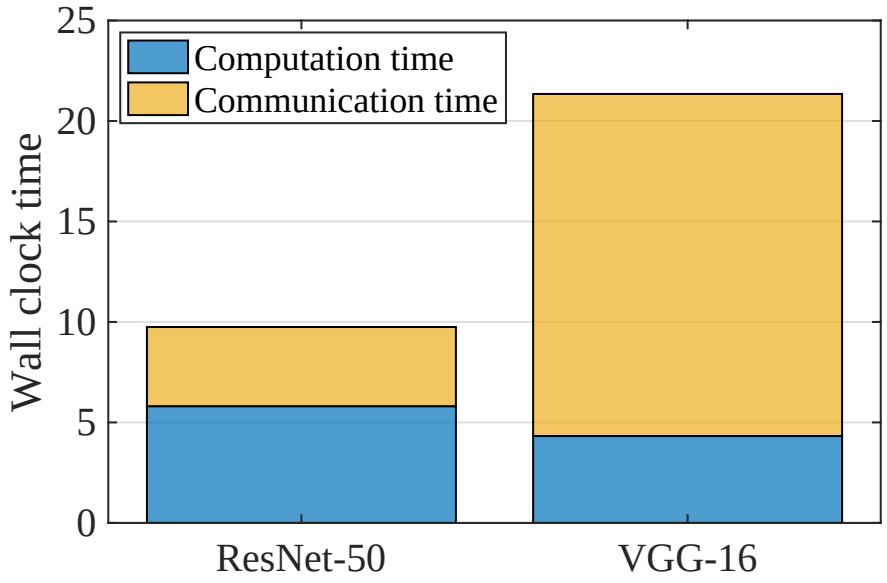
$$\mathbb{E}[T_{local}] = \mathbb{E}[Y'_{m:m}] + \frac{\mathbb{E}[D]}{\tau}$$

- Local SGD reduces both the mean and variance of the runtime



Speed-up Over Synchronous SGD

- Suppose Y and D are constants, and their ratio, that communication to computation time ratio is $\alpha = D/Y$
- The value of α depends on the size of model being trained, the available communication bandwidth and other system parameters
- As we see below, the ratio α for VGG16 is more than that of ResNet16 because VGG16 is a larger neural network

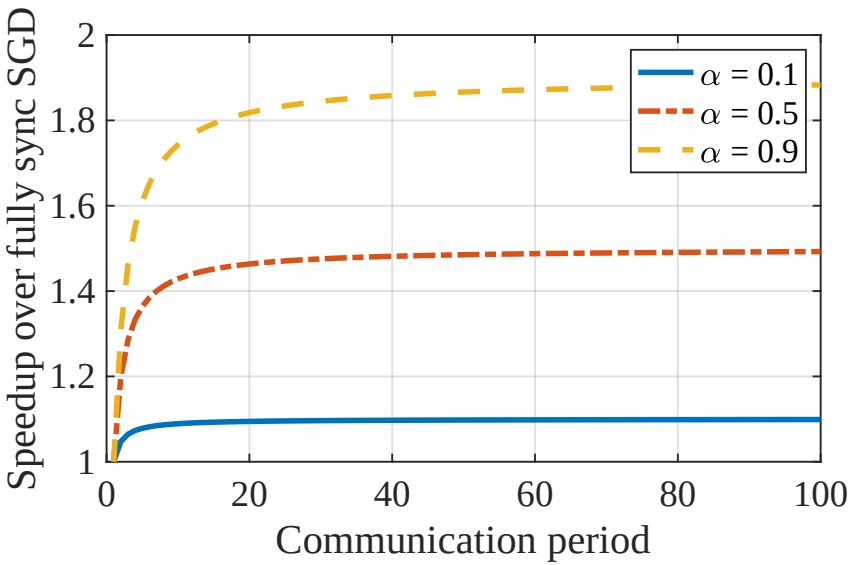


Speed-up Over Synchronous SGD

- Suppose Y and D are constants, and their ratio, that communication to computation time ratio is $\alpha = D/Y$
- The speed-up of local-update SGD over synchronous SGD is given by

$$\frac{\mathbb{E}[T_{sync}]}{\mathbb{E}[T_{local}]} = \frac{Y + D}{Y + D/\tau} = \frac{1 + \alpha}{1 + \alpha/\tau}$$

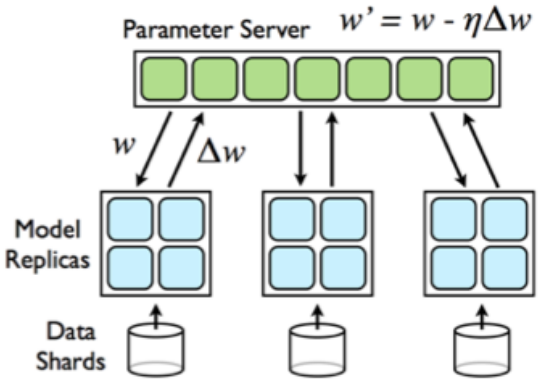
- As α increases, local-update SGD gives an even larger speed-up over synchronous SGD

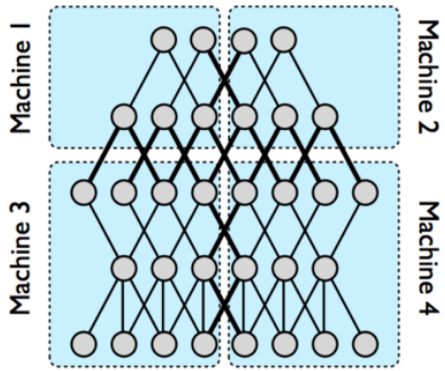


From Distributed ML to Federated Learning

Distributed ML in the Data-center Setting

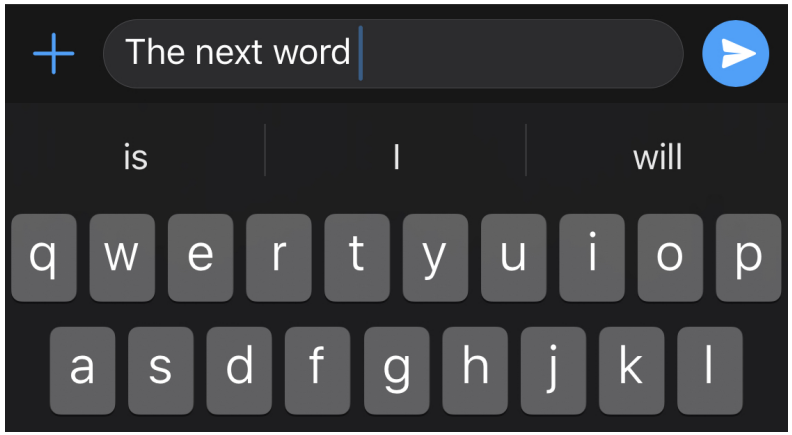
- We have a massive training dataset, which is shuffled and split across multiple nodes (servers in the cloud, often equipped with GPUs)
- A parameter server aggregates gradients from them using synchronous, asynchronous, local-update and/or gradient compression methods that we learned so far





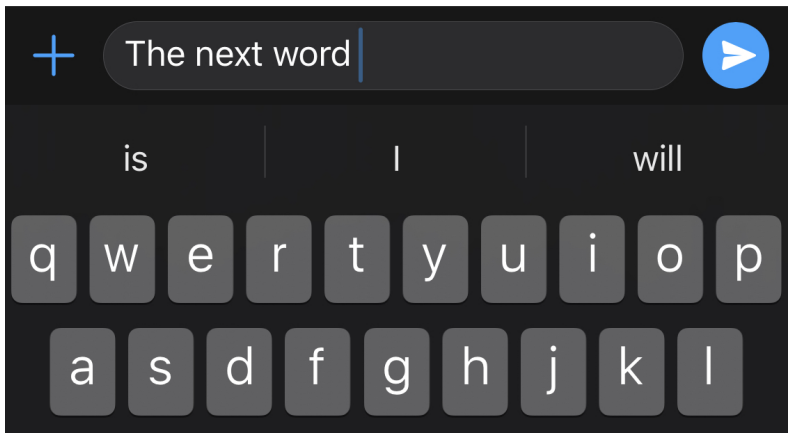
Data Collection at Edge Clients

- Edge clients such as cell phone and IoT devices collect massive amounts of data that can be used to train informative ML models
- Consider the next word prediction service on cell phone keyboards
- Training data – What each user types on their phone
- These data can be used to train language models that can accurately predict the next word



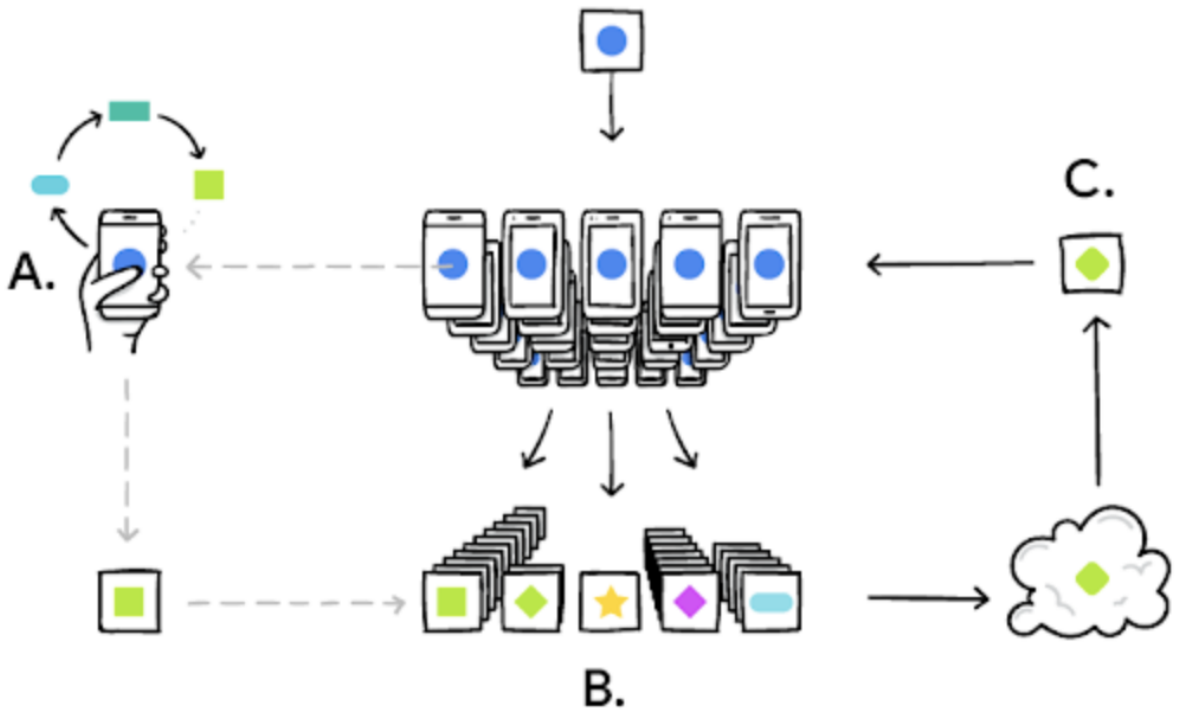
Standard ML Training on Edge Data

- How would you train a single ML model on such edge data using the parameter server framework that we studied?
- Each edge client (there can be millions) sends its data to the cloud, and then we use parameter server based training
- What are some problems with this strategy?
  1. Because of the size of raw training data, the communication cost (in terms of time and energy) can be prohibitively high
  2. The clients may only have an intermittent network connectivity
  3. Raw data may contain private and sensitive user information



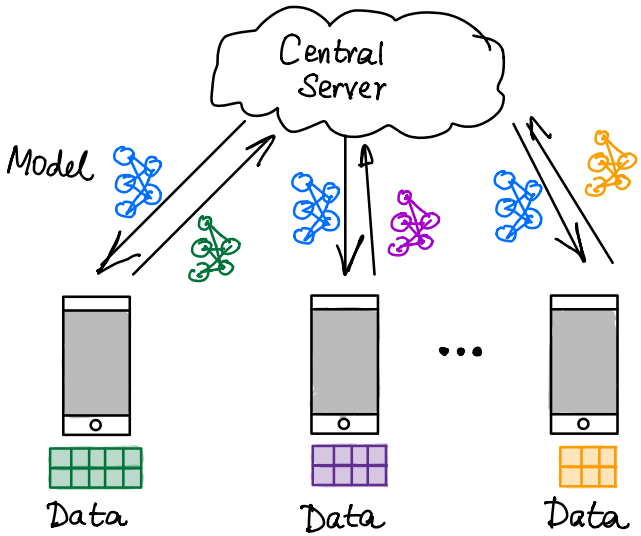
Solution: Federated Learning

- Proposed in the 2017 paper by McMahan et al at Google Seattle
- Main Idea: Bring the model training to the edge data
- Already used for next-word prediction on Android cell phones, when the phone is plugged in for charging
- Read more in this comic book: `http://federated.withgoogle.com/`



Solution: Federated Learning

- Main Idea: Keep the data at the edge client, and bring the model training to the edge
- Sketch of the Algorithm:
  1. The aggregating server sends the current version of the model to available clients
  2. The clients train the model locally for a few iterations and send it back
  3. The server aggregates the models and goes back to step 1



Does this look familiar to an algorithm that we studied in the parameter server setting? Local-update SGD

Differences from Local-update SGD

Local-update SGD

1. Tens or at most hundreds of worker nodes
2. All workers are available throughout training
3. Training data are well-shuffled and split equally across the workers
4. Worker nodes typically have homogeneous speeds, and only experience short-term straggling
5. Data can be re-organized across workers

Federated Learning

1. Thousands or millions of geo-distributed edge clients.
2. Only a subset of clients can participate in each round
3. Data are highly heterogeneous across clients in size and composition
4. Computation speeds can vary widely across clients depending on the device model, type etc.
5. Privacy constraints prevent data sharing between clients

Summary

Today we covered

- Convergence of GD and mini-batch SGD
- Distributed Synchronous SGD
- Distributed Asynchronous SGD
- Local-update SGD
- Federated Learning

I will teach 18-667 (Algorithms for Large-Scale Distributed ML) in Fall 2025, which will cover these and other distributed ML algorithms in more detail.