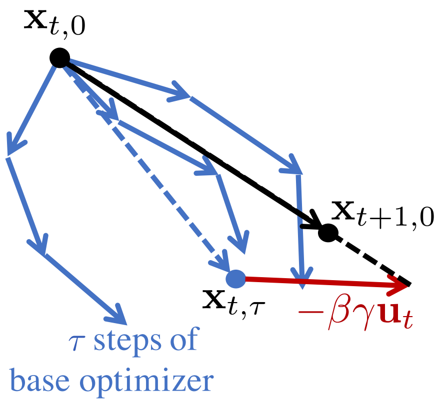
SlowMo

Improving Communication-Efficient Distributed SGD with *Slow Momentum*



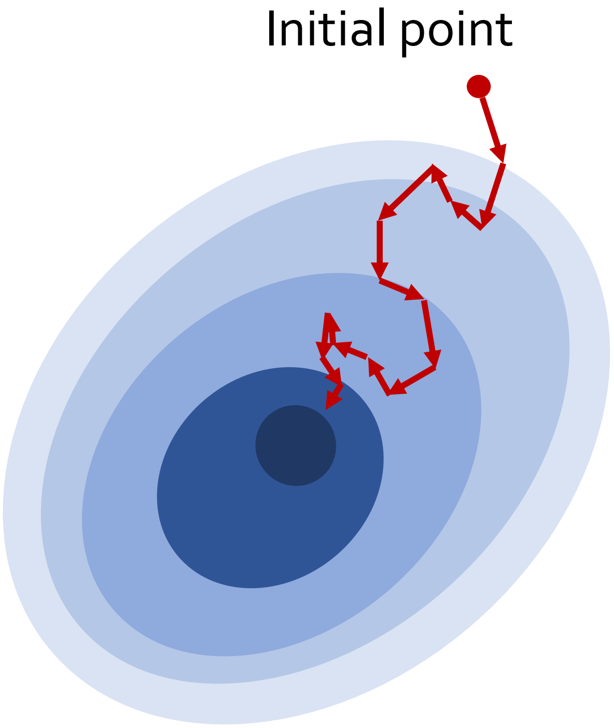
Jianyu Wang[1], Vinayak Tantia[2], Nicolas Ballas[2], Michael Rabbat[2]

[1]*Carnegie Mellon University*

[2]*Facebook AI Research*

Stochastic Gradient Descent

Stochastic gradient descent (SGD) is the backbone of ML, especially deep learning



Loss incurred by the i-th sample

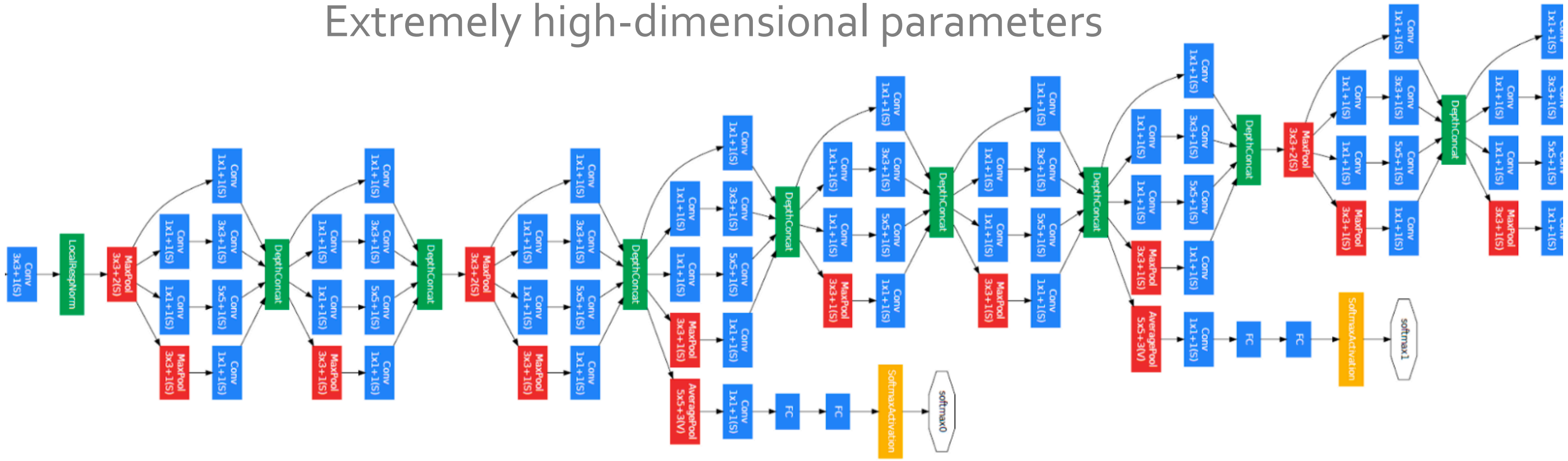
Empirical Risk $F(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \boxed{f_i(\mathbf{x})}$

Mini-batch SGD $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \cdot \boxed{\frac{1}{|\xi_k|}\sum_{j\in\xi_k} \nabla f_j(\mathbf{x}_k)}$

Stochastic gradient

Big Model, Big Data

Extremely high-dimensional parameters



IMAGENET

Extremely large training datasets

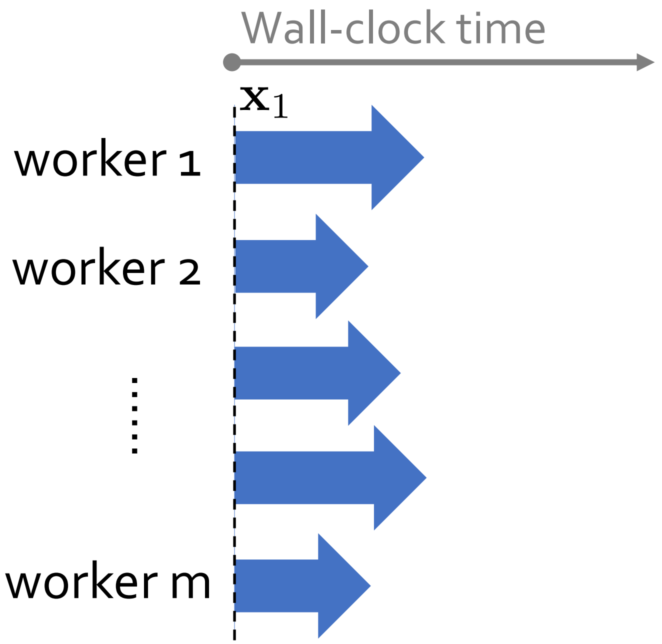
Training on a single machine
can takes several days or even weeks.

It is imperative to distribute SGD across multiple machines!

Classic Method: Fully Synchronous SGD

Execution pipeline:

1. Local stochastic gradients computation



Gradient at **k-th** iteration and **i-th** worker:

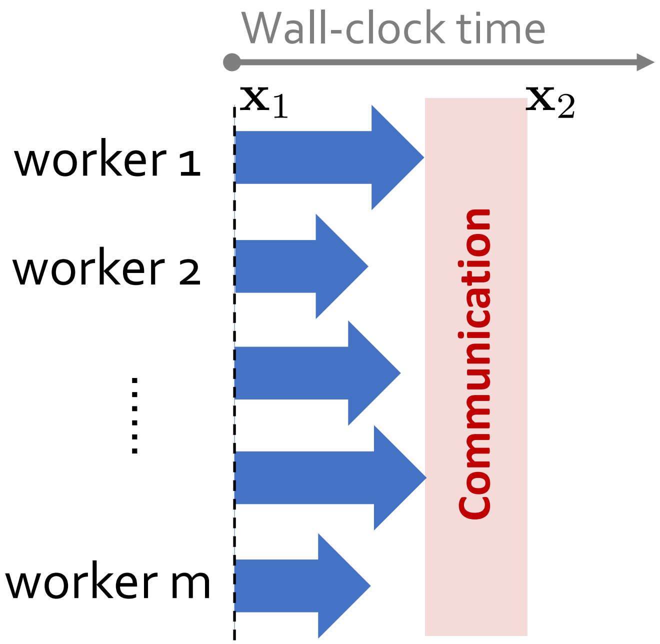
$$g(\mathbf{x}_k; \xi_k^{(i)}) = \frac{1}{|\xi_k^{(i)}|} \sum_{j \in \xi_k^{(i)}} \nabla f_j(\mathbf{x}_k)$$

- **Blue arrows:** gradient computation time

Classic Method: Fully Synchronous SGD

Execution pipeline:

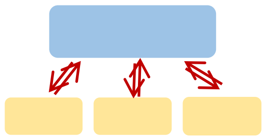
1. Local stochastic gradients computation
2. Average local models across all nodes



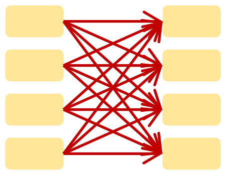
$$\frac{1}{m}\sum_{i=1}^{m}\mathbf{x}^{(i)}$$

Communication can be implemented via:

Parameter Server



All-Reduce



Li et al. **Scaling Distributed Machine Learning with the Parameter Server**, In *OSDI 2014*

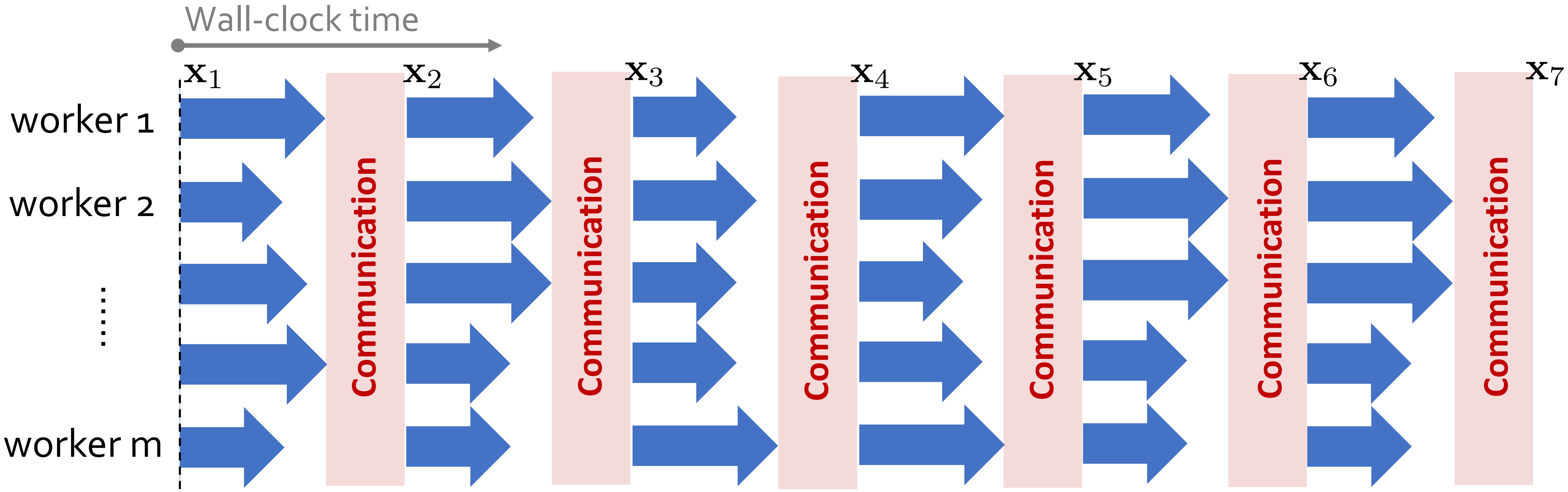
Goyal et al. **Accurate, Large Mini-Batch SGD: Training ImageNet in 1 Hour**, *ArXiv preprint 2017*

- **Blue arrows:** gradient computation time
- **Red blocks:** communication time

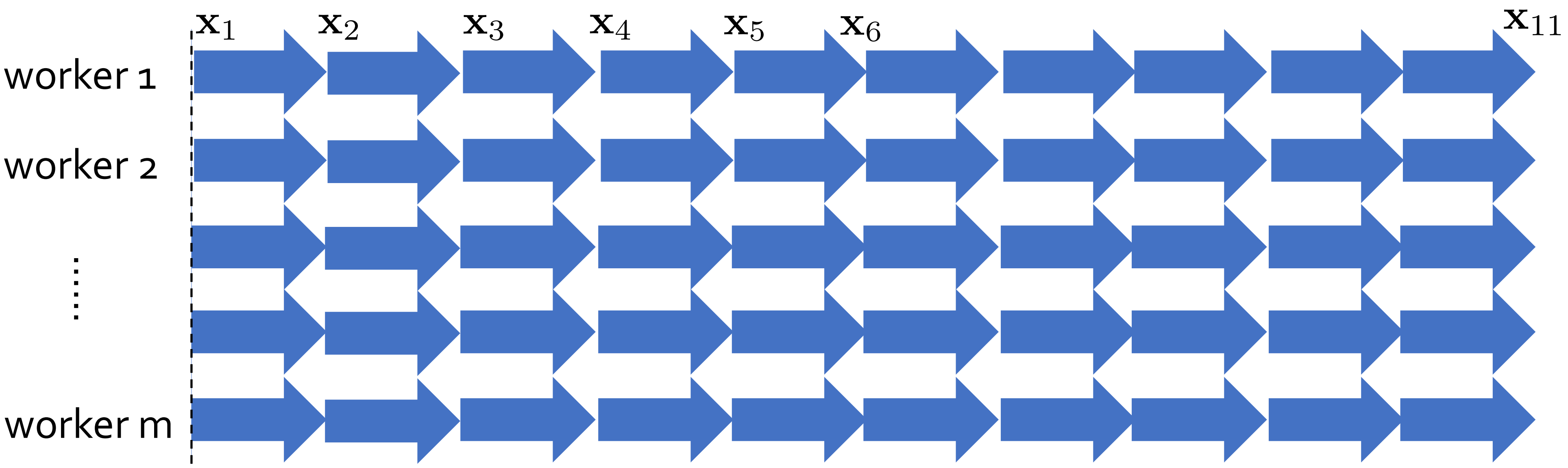
Classic Method: Fully Synchronous SGD

Execution pipeline:

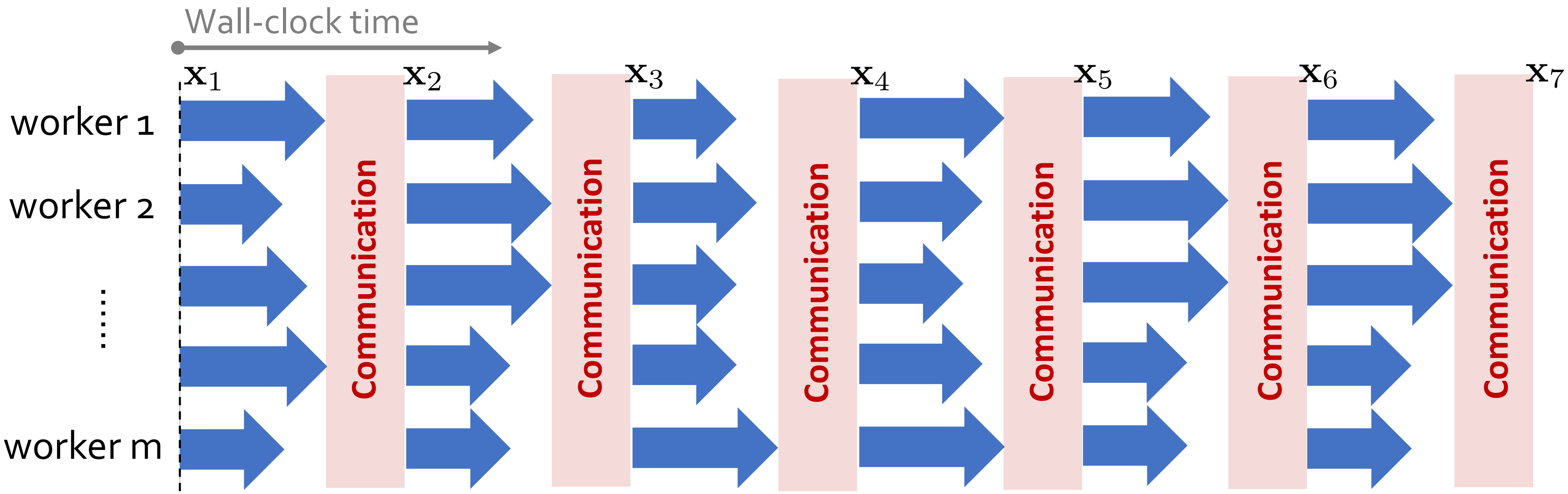
1. Local stochastic gradients computation
2. Average local models across all nodes
3. Repeat the above steps until convergence



- **Blue arrows:** gradient computation time
- **Red blocks:** communication time



Ideal:
11 iterations

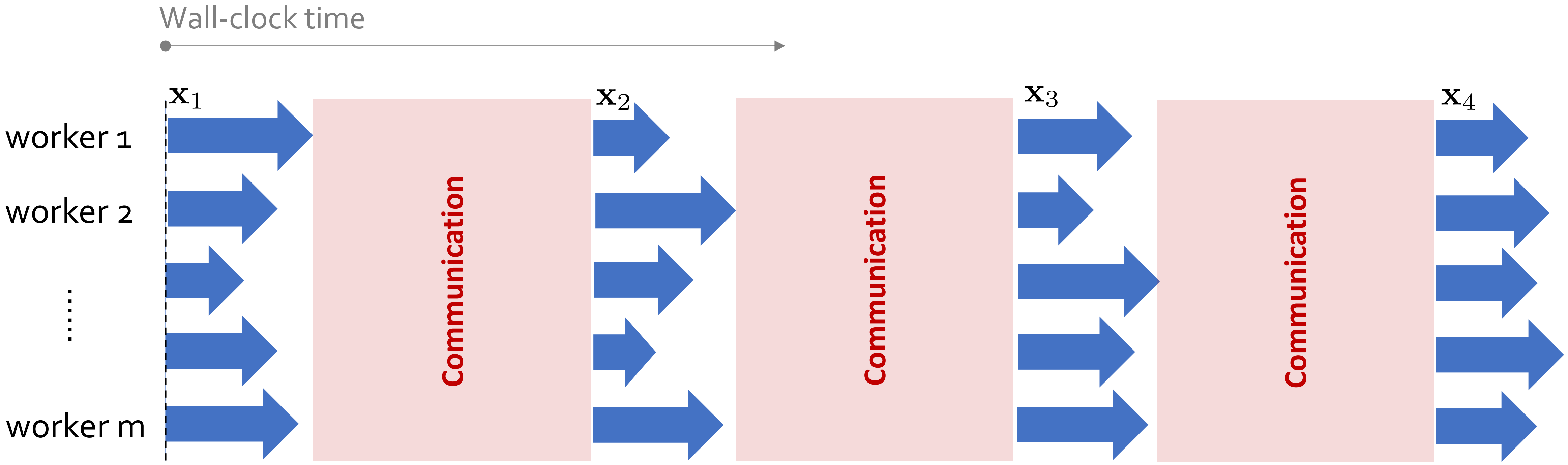


Practice:
7 iterations

- **Blue arrows:** gradient computation time
- **Red blocks:** communication time

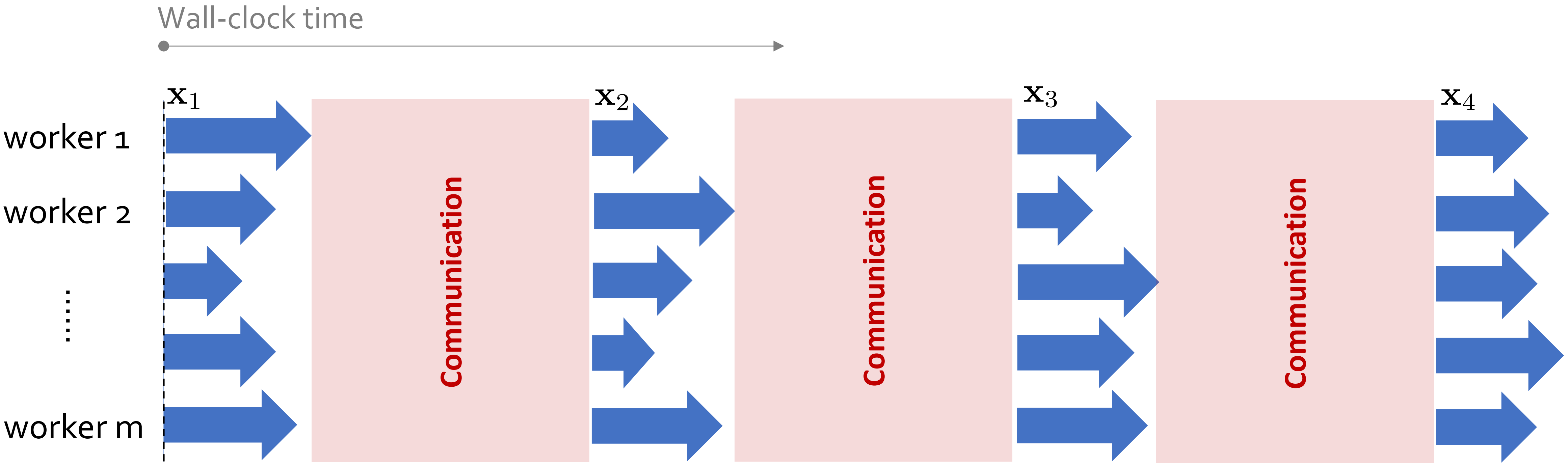
Communication is the Bottleneck in DNN Training

In deep neural nets training, the **communication time can be even larger than computation time**. [Harlap et al. ArXiv preprint 2018; Wang and Joshi, SysML 2019]



Communication is the Bottleneck in DNN Training

It is critical to develop **communication-efficient distributed SGD**



Background: Communication-Efficient Training

Motivation

Update rule of fully synchronous SGD (i.e., AllReduce SGD/AR-SGD)

$$\boldsymbol{X}_{k+1} = [\boldsymbol{X}_k - \eta \boldsymbol{G}_k]\boldsymbol{J}$$

Local model at one node

where $\boldsymbol{X}_k = [\boldsymbol{x}_k^{(1)}, \boldsymbol{x}_k^{(2)}, \ldots, \boldsymbol{x}_k^{(m)}] \in \mathbb{R}^{d \times m}$

$$\boldsymbol{G}_k = [g(\boldsymbol{x}_k^{(1)}; \xi_k^{(1)}), g(\boldsymbol{x}_k^{(2)}; \xi_k^{(2)}), \ldots, g(\boldsymbol{x}_k^{(m)}; \xi_k^{(m)})] \in \mathbb{R}^{d \times m}$$

$\boldsymbol{J} = \mathbf{1}\mathbf{1}^\top / m$ Fully synchronization (AllReduce) matrix

After preform AllReduce operation ($\boldsymbol{J}$), all local models (columns in X) are the same

Is it necessary? Can we replace J by other matrices?

Background: Communication-Efficient Training

Key Ideas:

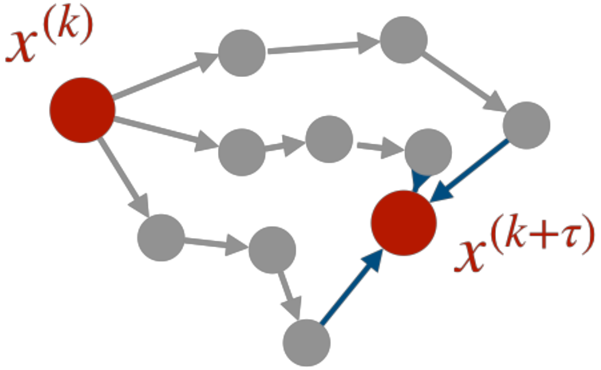
- Reduce communication by allowing inconsistent local models
- Let synchronization matrix $\boldsymbol{J}$ to be sparse: $\boldsymbol{J} \to \boldsymbol{S}_k$

Example Algorithms

- **Local SGD:** *"temporal" sparse synchronization*
    - reduce the communication frequency

$$\boldsymbol{X}_{k+1} = [\boldsymbol{X}_k - \eta \boldsymbol{G}_k]\boldsymbol{S}_k$$

$$\boldsymbol{S}_k = \begin{cases} \boldsymbol{J} & k \bmod \tau = 0 \\ \boldsymbol{I} & \text{otherwise} \end{cases}$$



Stich et al. **Local SGD Converges Fast and Communicates little**, in *ICLR 2019;*
Wang and Joshi. ***Cooperative SGD***, in *ICML CodML workshop 2019*
Hao et al. **Parallel Restarted SGD**, in *AAAI 2019*

Background: Communication-Efficient Training

Key Ideas:

- Reduce communication by allowing inconsistent local models
- Let synchronization matrix $\boldsymbol{J}$ to be sparse: $\boldsymbol{J} \to \boldsymbol{S}_k$

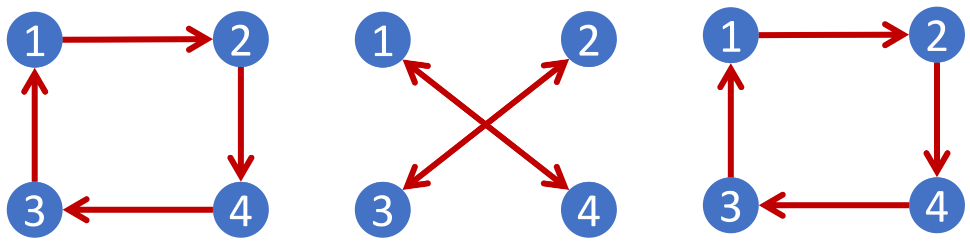
Example Algorithms

- **Stochastic Gradient Push:** *"spatial" sparse synchronization*
  - Only synchronize with one neighbor instead of all

$$\boldsymbol{X}_{k+1} = [\boldsymbol{X}_k - \eta \boldsymbol{G}_k]\boldsymbol{S}_k$$

Row-stochastic matrix

- Elements on each row sum to 1
- Each row has only two non-zero elements



Iteration 1 Iteration 2 Iteration 3

$$\mathbf{x}^{(1)}_{\text{new}} = \frac{1}{2}(\mathbf{x}^{(3)} + \mathbf{x}^{(1)}) \quad \mathbf{x}^{(1)}_{\text{new}} = \frac{1}{2}(\mathbf{x}^{(4)} + \mathbf{x}^{(1)}) \quad \mathbf{x}^{(1)}_{\text{new}} = \frac{1}{2}(\mathbf{x}^{(3)} + \mathbf{x}^{(1)})$$

Assran et al. **Stochastic Gradient Push for Distributed Deep Learning**, in *ICML 2019*

Background: Communication-Efficient Training

Key Ideas:

- Reduce communication by allowing inconsistent local models
- Let synchronization matrix $\boldsymbol{J}$ to be sparse: $\boldsymbol{J} \rightarrow \boldsymbol{S}_k$

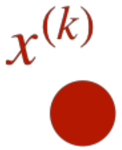
Example Algorithms

- **Stochastic Gradient Push:** *"spatial" sparse synchronization*

$$\boldsymbol{X}_{k+1} = [\boldsymbol{X}_k - \eta \boldsymbol{G}_k]\boldsymbol{S}_k$$

Row-stochastic matrix

- Elements on each row sum to 1
- Each row has only two non-zero elements



Assran et al. **Stochastic Gradient Push for Distributed Deep Learning**, in *ICML 2019*

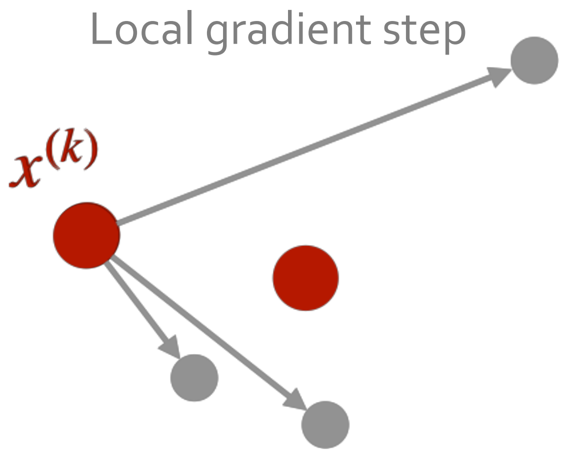
Background: Communication-Efficient Training

Key Ideas:

- Reduce communication by allowing inconsistent local models
- Let synchronization matrix $\boldsymbol{J}$ to be sparse: $\boldsymbol{J} \rightarrow \boldsymbol{S}_k$

Example Algorithms

- **Stochastic Gradient Push:** *"spatial" sparse synchronization*



$$\boldsymbol{X}_{k+1} = [\boldsymbol{X}_k - \eta \boldsymbol{G}_k]\boldsymbol{S}_k$$

Row-stochastic matrix

- Elements on each row sum to 1
- Each row has only two non-zero elements

Assran et al. **Stochastic Gradient Push for Distributed Deep Learning**, in *ICML 2019*

Background: Communication-Efficient Training

Key Ideas:

- Reduce communication by allowing inconsistent local models
- Let synchronization matrix $\boldsymbol{J}$ to be sparse: $\boldsymbol{J} \rightarrow \boldsymbol{S}_k$

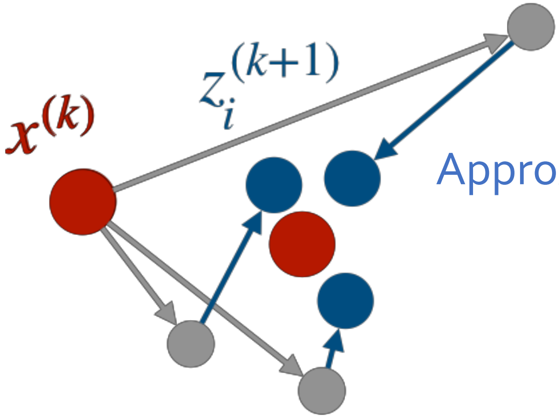
Example Algorithms

- **Stochastic Gradient Push:** *"spatial" sparse synchronization*

$$\boldsymbol{X}_{k+1} = [\boldsymbol{X}_k - \eta \boldsymbol{G}_k]\boldsymbol{S}_k$$

Row-stochastic matrix

- Elements on each row sum to 1
- Each row has only two non-zero elements



Approximate distributed averaging

Assran et al. **Stochastic Gradient Push for Distributed Deep Learning**, in *ICML 2019*

Background: Communication-Efficient Training

Key Ideas:

- Reduce communication by allowing inconsistent local models
- Let synchronization matrix $\boldsymbol{J}$ to be sparse: $\boldsymbol{J} \rightarrow \boldsymbol{S}_k$

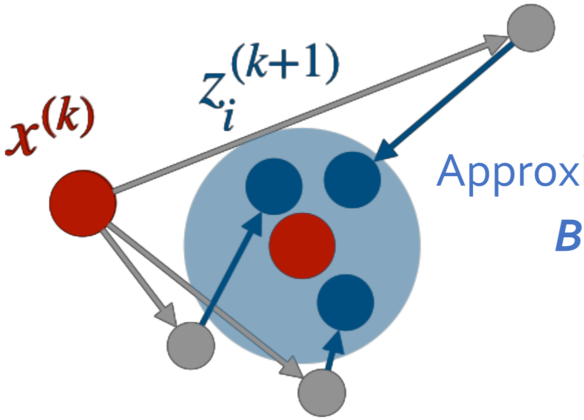
Example Algorithms

- **Stochastic Gradient Push:** *"spatial" sparse synchronization*

$$\boldsymbol{X}_{k+1} = [\boldsymbol{X}_k - \eta \boldsymbol{G}_k] \boldsymbol{S}_k$$

Row-stochastic matrix

- Elements on each row sum to 1
- Each row has only two non-zero elements



Approximate distributed averaging

Bounded disagreement

Assran et al. **Stochastic Gradient Push for Distributed Deep Learning**, in *ICML 2019*

Background: Communication-Efficient Training

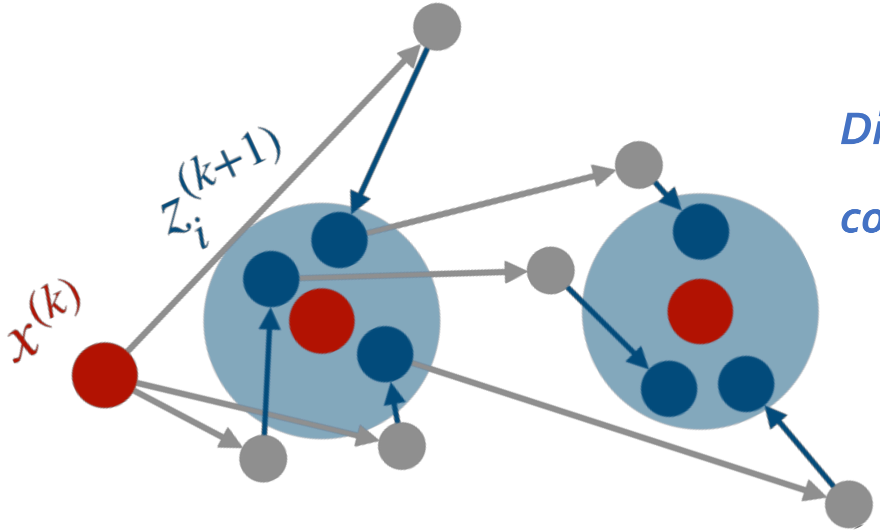
Key Ideas:

- Reduce communication by allowing inconsistent local models
- Let synchronization matrix $\boldsymbol{J}$ to be sparse: $\boldsymbol{J} \rightarrow \boldsymbol{S}_k$

Example Algorithms

- **Stochastic Gradient Push:** *"spatial" sparse synchronization*

$$\boldsymbol{X}_{k+1} = [\boldsymbol{X}_k - \eta \boldsymbol{G}_k] \boldsymbol{S}_k$$



Disagreement provably converges to 0

Row-stochastic matrix

- Elements on each row sum to 1
- Each row has only two non-zero elements

Assran et al. **Stochastic Gradient Push for Distributed Deep Learning**, in *ICML 2019*

Background: Communication-Efficient Training

Key Ideas:

- Reduce communication by allowing inconsistent local models
- Let synchronization matrix $\boldsymbol{J}$ to be sparse: $\boldsymbol{J} \to \boldsymbol{S}_k$

Example Algorithms

- **Stochastic Gradient Push:** *"spatial" sparse synchronization*
  - Only synchronize with one neighbor instead of all

Algorithm	# handshakes	Transferred data size
AR-SGD	O(m) or O(log m)	O(1)
SGP	**O(1)**	**O(1)**

$$\boldsymbol{X}_{k+1} = [\boldsymbol{X}_k - \eta \boldsymbol{G}_k]\boldsymbol{S}_k$$

Row-stochastic matrix

- Elements on each row sum to 1
- Each row has only two non-zero elements

Assran et al. **Stochastic Gradient Push for Distributed Deep Learning**, in *ICML 2019*

Distributed Momentum Scheme

The momentum scheme for communication-efficient training methods have not been formally studied

- **Local momentum Scheme:**
  - By default, SGP and Local SGD let workers maintain unsynchronized local momentum buffers
- **Double-Averaging Scheme:** [Yu et al. ICML 2019]
  - Average momentum buffers as well as model parameters
  - Doubled/tripled communication cost

Algorithm	Time/iteration	Best Validation Acc.
AR-SGD	420 ms	76.00%
SGP	304 ms	75.15%
SGP-double-avg	402 ms	75.54%
Local SGD	294 ms	69.94%

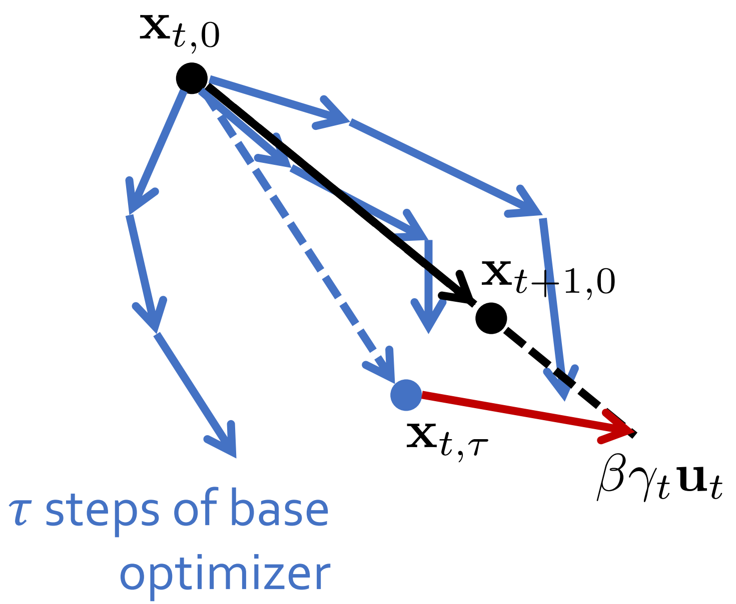
Reset50, ImageNet Training

- 8k mini-batch size
- 10Gbps Ethernet

We propose *Slow Momentum (SlowMo)*:

- 👍 ***A Novel Distributed Momentum Scheme***
  - Improve performance of communication-efficient distributed SGD
  - Negligible additional overhead
  - Convergence guarantee for non-convex loss functions
- 👍 ***A General Framework***
  - Can be applied on top of various distributed optimizers, such as SGP, Overlap-SGP, Local SGD, D-PSGD, etc.

Our Solution: Slow Momentum (SlowMo)



Step 1

starting from $\mathbf{x}_{t,0}$, perform multiple steps of base optimizer

- Base optimizer can have local momentum --> two-layer momentum
- Base optimizer can be SGP, Local SGD, D-PSGD, etc.

Step 2

Average all local models and obtain $\mathbf{x}_{t,\tau}$

Treat $\frac{1}{\gamma_t}(\mathbf{x}_{t,0} - \mathbf{x}_{t,\tau})$ as a *pseudo-gradient* for $\mathbf{x}_{t,0}$

Step 3

Update slow momentum buffer $\mathbf{u}_{t+1} = \beta\mathbf{u}_t + \frac{1}{\gamma_t}(\mathbf{x}_{t,0} - \mathbf{x}_{t,\tau})$

Update initial point $\mathbf{x}_{t+1,0} = \mathbf{x}_{t,0} - \alpha\gamma_t\mathbf{u}_{t+1}$

- "Slow" because updated after every τ steps

Our Solution: Slow Momentum (SlowMo)

Step 1

starting from $\mathbf{x}_{t,0}$, perform multiple steps of base optimizer

- Base optimizer can have local momentum --> two-layer momentum
- Base optimizer can be SGP, Local SGD, D-PSGD, etc.

Step 2

Average all local models and obtain $\mathbf{x}_{t,\tau}$

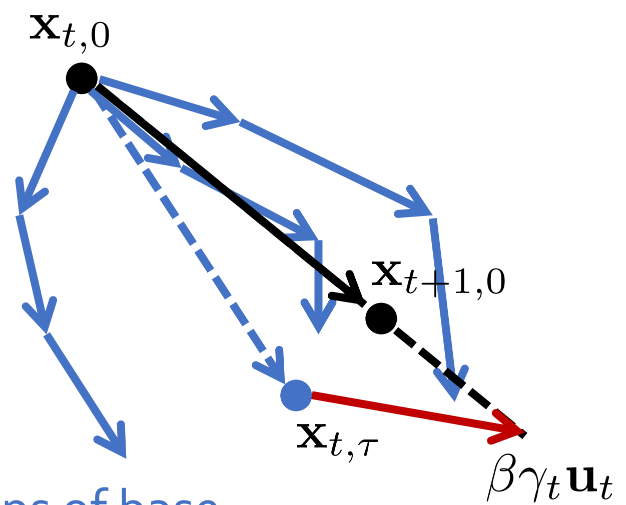
Treat $\frac{1}{\gamma_t}(\mathbf{x}_{t,0} - \mathbf{x}_{t,\tau})$ as a *pseudo-gradient* for $\mathbf{x}_{t,0}$

Step 3

Update slow momentum buffer $\mathbf{u}_{t+1} = \beta\mathbf{u}_t + \frac{1}{\gamma_t}(\mathbf{x}_{t,0} - \mathbf{x}_{t,\tau})$

Update initial point $\mathbf{x}_{t+1,0} = \mathbf{x}_{t,0} - \alpha\gamma_t\mathbf{u}_{t+1}$

- "Slow" because updated after every τ steps



τ steps of base optimizer

α Global learning rate
β Slow momentum factor
τ Sync. Period

$$\boxed{\mathcal{A}(\alpha, \beta, \tau, \text{optimizer})}$$

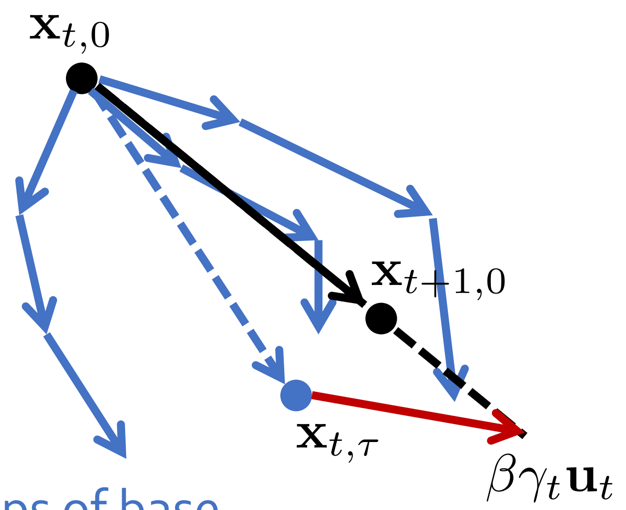
Design Choice: Buffer Strategies in SlowMo

When the base optimizer has momentum or other buffers

- For example, use Adam as base optimizer
- It has 1st-order and 2nd-order momen. buffers

After each global step, one can choose to

1. Reinitialize local buffers — Works best for Image Classification
2. Maintain local buffers — Works best for Language Modeling
3. Synchronize local buffers (additional comm. cost)



τ steps of base optimizer

Convergence Analysis: Assumptions

(A1) Lipschitz smooth: $\|\nabla F_i(\mathbf{x}) - \nabla F_i(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|$

(A2) Unbiased gradient estimation: $\mathbb{E}_{\xi^{(i)}|\mathbf{x}}[g(\mathbf{x};\xi^{(i)})] = \nabla F_i(\mathbf{x})$

(A3) Bounded variance: $\mathbb{E}_{\xi^{(i)}|\mathbf{x}}\left[\left\| g(\mathbf{x};\xi^{(i)}) - \nabla F_i(\mathbf{x}) \right\|^2\right] \leq \sigma^2$

Convergence Analysis of SlowMo

$$\mathcal{A}(\alpha, \beta, \tau, \text{optimizer})$$

The proposed algorithm can converge to a stationary point

$$\frac{1}{K}\sum_{t=0}^{T-1}\sum_{k=0}^{\tau-1}\mathbb{E}\|\nabla F(\mathbf{x}_{t,k})\|^2 \leq \mathcal{O}(\frac{1}{\sqrt{mK}}) + \mathcal{O}(\frac{1}{K}) + \underbrace{\frac{1}{K}\sum_{t=0}^{T-1}\sum_{k=0}^{\tau-1}\mathbb{E}\|\nabla F(\mathbf{x}_{t,k}) - \mathbb{E}_{t,k}[\mathbf{d}_{t,k}]\|^2}_{\text{Noise from inner optimizer}}$$

where

- K: total iterations
- m: number of worker nodes
- F: objective function

Has already been shown in previous works

$$\mathcal{O}(\frac{m}{K})$$

If base optimizer converges, then SlowMo converges in the same rate

Convergence Analysis of SlowMo

$$\mathcal{A}(\alpha, \beta, \tau, \text{optimizer})$$

The proposed algorithm can converge to a stationary point

$$\frac{1}{K}\sum_{t=0}^{T-1}\sum_{k=0}^{\tau-1}\mathbb{E}\|\nabla F(\mathbf{x}_{t,k})\|^2 \leq \mathcal{O}(\frac{1}{\sqrt{mK}}) + \mathcal{O}(\frac{1}{K}) + \underbrace{\frac{1}{K}\sum_{t=0}^{T-1}\sum_{k=0}^{\tau-1}\mathbb{E}\|\nabla F(\mathbf{x}_{t,k}) - \mathbb{E}_{t,k}[\mathbf{d}_{t,k}]\|^2}_{\text{Noise from inner optimizer}}$$

where

- K: total iterations
- m: number of worker nodes
- F: objective function

1. When total iterations is sufficiently large, the convergence rate will be dominated by $1/\sqrt{mK}$
   - *Same rate as AR-SGD*
2. Linear Speedup: more workers, less iterations
3. Changing hyper-parameters can improve constants but the rate remains the same

Subsume Previous Algorithms as Special Cases

$$\mathcal{A}(\underbrace{0.5}_{\text{Global LR}}, \underbrace{0}_{\text{Slow Momen.}}, \underbrace{\tau}_{\text{Sync. Period}}, \text{AR-SGD})$$

- Lookahead

Zhang et al., *NeurIPS 2019*, **“Lookahead Optimizer: k steps forward, 1 step back”**

$$\mathcal{A}(\alpha, \beta, \tau, \text{Local-SGD})$$

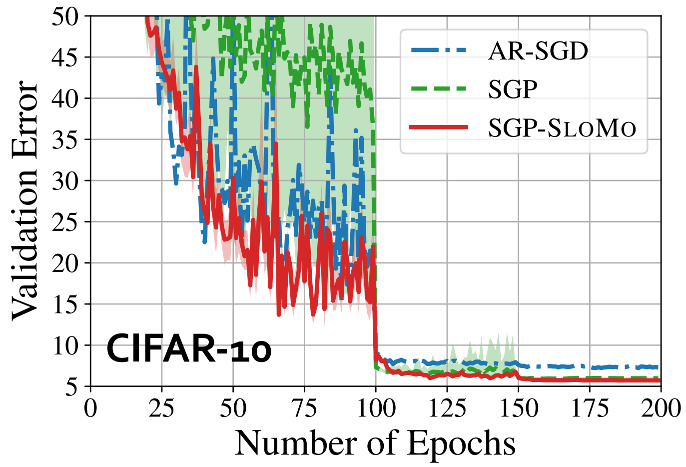
- Blockwise Model Update Filtering

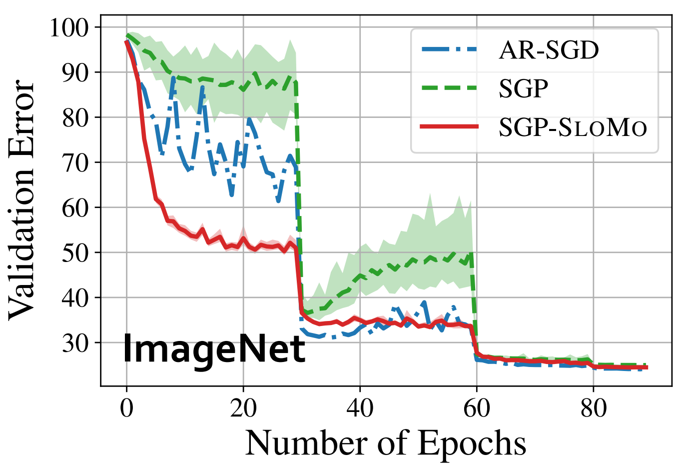
Chen & Huo., *ICASSP 2016*, **“Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering”**

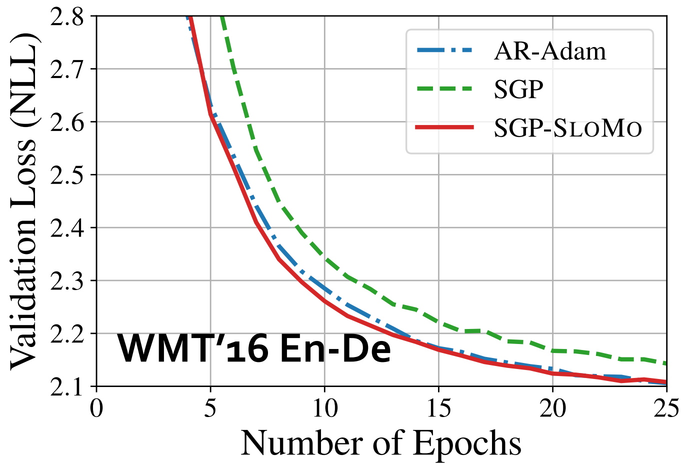
We provide the **first convergence guarantee** for these two algorithms **under non-convex setting**!

Empirical Results: Training Curves

Faster Convergence, Better Validation Accuracy







CIFAR-10

- ResNet-18
- 32 NVIDIA V100 GPUs
- Mini-batch size: 4k

$\alpha = 1, \beta = 0.7, \tau = 12$

ImageNet

- ResNet-50
- 32 NVIDIA DGX-1 servers
- Mini-batch size: 8k

$\alpha = 1, \beta = 0.7, \tau = 48$

WMT’16

- Transformer
- 8 NVIDIA DGX-1 servers
- Mini-batch size: 200k

$\alpha = 1, \beta = 0.7, \tau = 48$

Empirical Results: Validation Accuracy

Faster Convergence, Better Validation Accuracy

CIFAR-10

Base Optimizer	Original	w/ SlowMo	
Local SGD	91.73	93.20	+1.5%
OSGP	93.17	93.74	+0.6%
SGP	93.90	94.32	+0.4%
ARSGD	92.66	-	

ImageNet

Base Optimizer	Original	w/ SlowMo	
Local SGD	69.94	73.24	+3.3%
OSGP	74.96	75.54	+0.6%
SGP	75.15	75.73	+0.6%
ARSGD	76.00	-	

Empirical Results: Time / Iteration

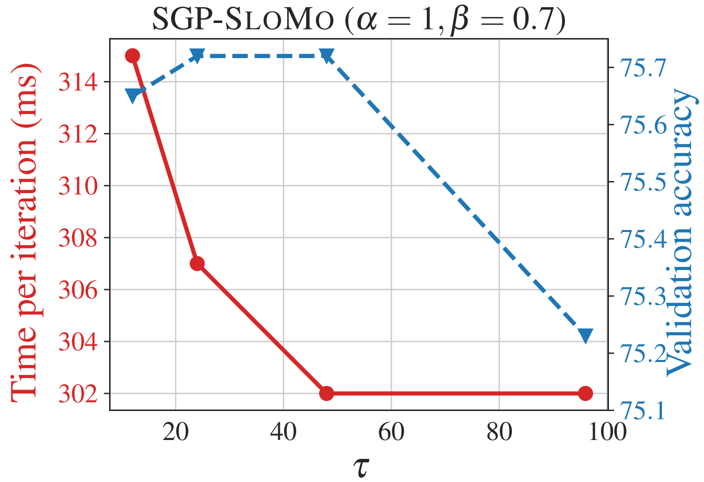
Negligible Additional Communication Cost

ImageNet

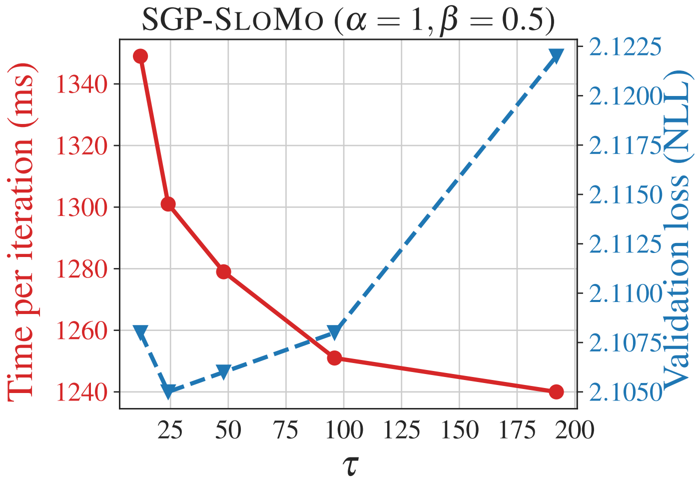
Base Optimizer	Original	w/ SlowMo
Local SGD	294 ms	282 ms
OSGP	271 ms	271 ms
SGP	304 ms	302 ms
ARSGD	402 ms	-

WMT’16 En-De

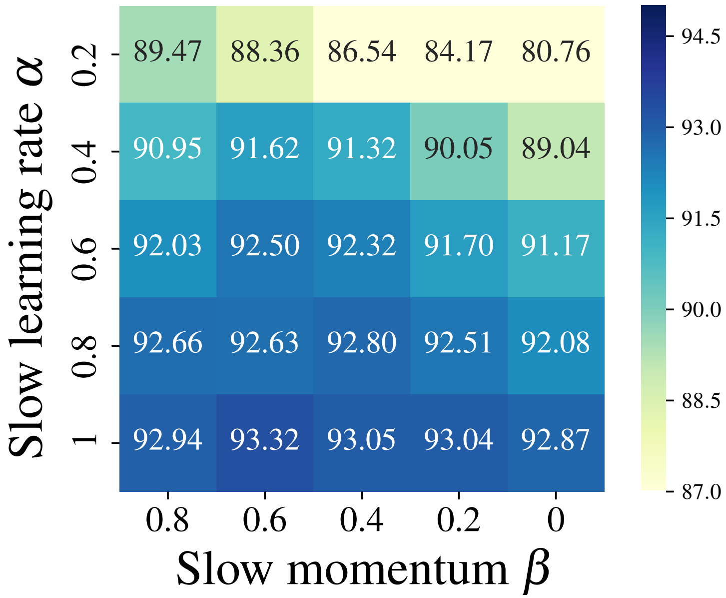
Base Optimizer	Original	w/ SlowMo
Local Adam	503 ms	505 ms
SGP	1225 ms	1279 ms
ARSGD	1648 ms	-



Effect of τ



How to set Hyper-parameters α, β?



	0.8	0.6	0.4	0.2	0
0.2	89.47	88.36	86.54	84.17	80.76
0.4	90.95	91.62	91.32	90.05	89.04
0.6	92.03	92.50	92.32	91.70	91.17
0.8	92.66	92.63	92.80	92.51	92.08
1	92.94	93.32	93.05	93.04	92.87

Parameter sweep on CIFAR-10 dataset:

- Larger global LR is better $\alpha = 1$
- There is a best value of slow momentum $\beta \in [0.4, 0.8]$

Comparison with Double-Averaging Momentum

[Yu et al. ICML 2019] proposes to average momentum buffers as well as model parameters

- Doubled/tripled communication costs

SlowMo achieves higher accuracy using less time

Algorithm	Time/iteration	Best Validation Acc.
AR-SGD	420 ms	76.00%
SGP	304 ms	75.15%
SGP-double-avg	402 ms	75.54%
SGP-SlowMO	**302 ms**	**75.73%**

Thanks for attention!

SlowMo: Improving Communication-Efficient Distributed SGD with Slow Momentum *arXiv: 1910.00643*

Code will be available soon. More questions: jianyuw1@andrew.cmu.edu,