北京大學
PEKING UNIVERSITY

DecentLaM: Decentralized Momentum SGD for Large-Batch Deep Training

Kun Yuan (袁　坤)

Center for Machine Learning Research @ Peking University

Nov. 22, 2022

Joint work with

Xinmeng Huang
(UPenn)

Yiming Chen
(Alibaba)

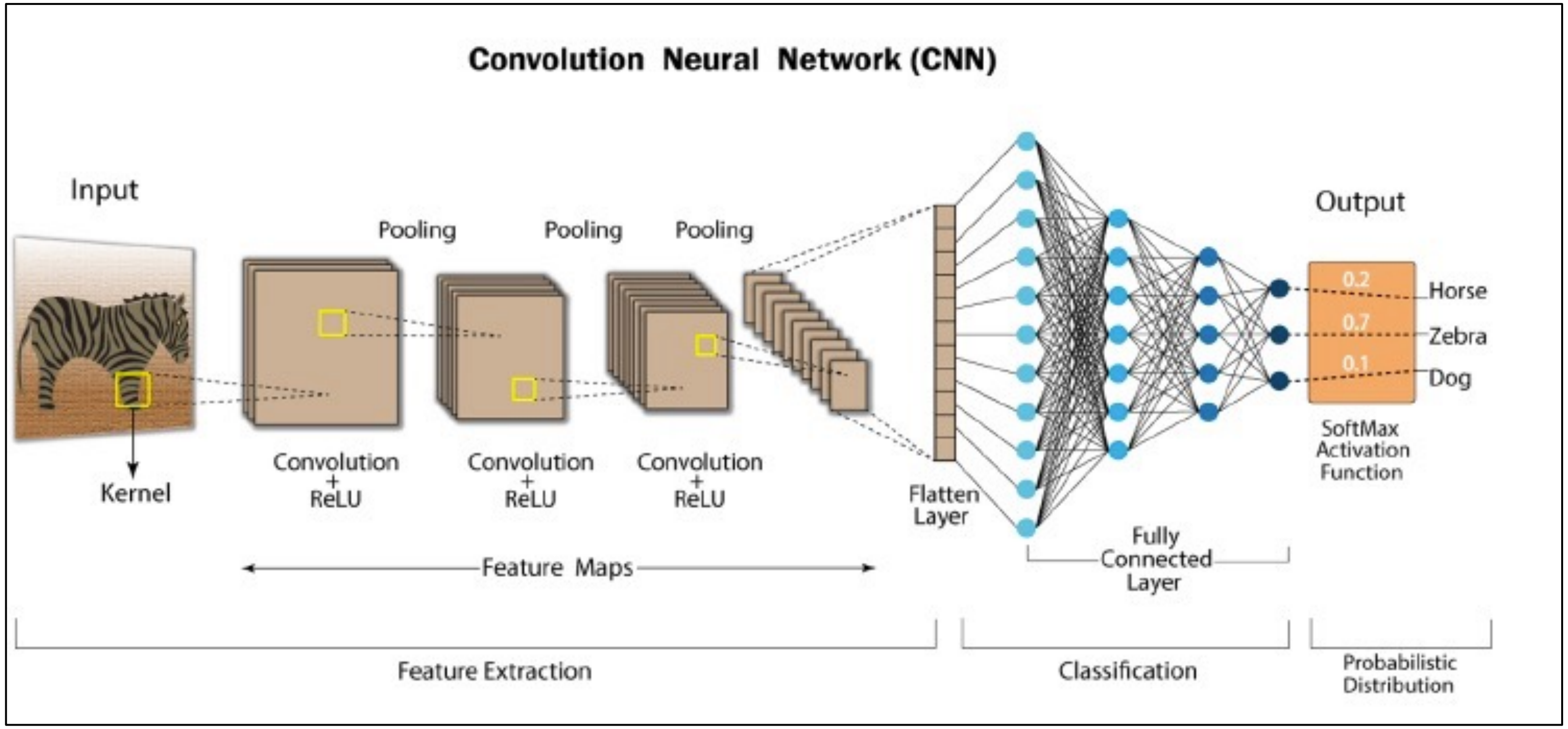
Bicheng Ying
(Google)

Wotao Yin
(Alibaba)

PART 01

Basics and Motivation

Training deep neural network is notoriously difficult



DNN training = non-convexity + **massive dataset** + huge models

Distributed learning

- Training deep neural networks typically requires **massive** datasets; efficient and scalable distributed optimization algorithms are in urgent need

- A network of n nodes (devices such as GPUs) collaborate to solve the problem:

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

  - Each component $f_i : \mathbb{R}^d \to \mathbb{R}$ is local and private to node i
  - Random variable ξ_i denotes the local data that follows distribution D_i
  - Each local distribution D_i is different; data heterogeneity exists

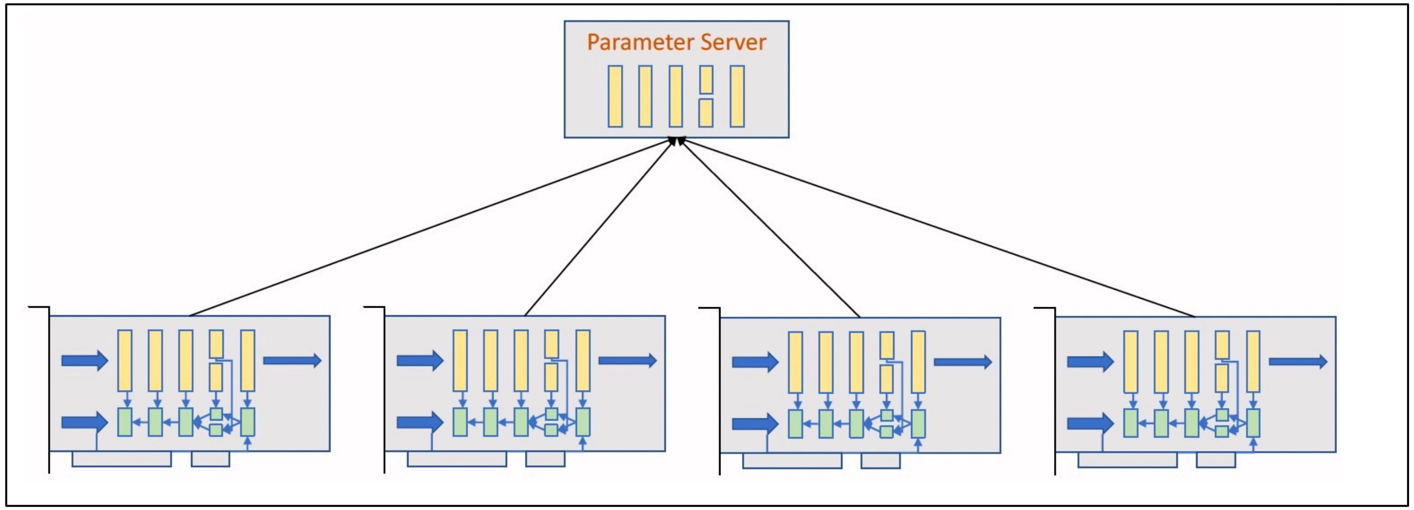
Vanilla parallel stochastic gradient descent (PSGD)

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

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

- Each node i samples data $\xi_i^{(k)}$ and computes gradient $\nabla F(x^{(k)}; \xi_i^{(k)})$
- All nodes synchronize (i.e. globally average) to update model x per iteration

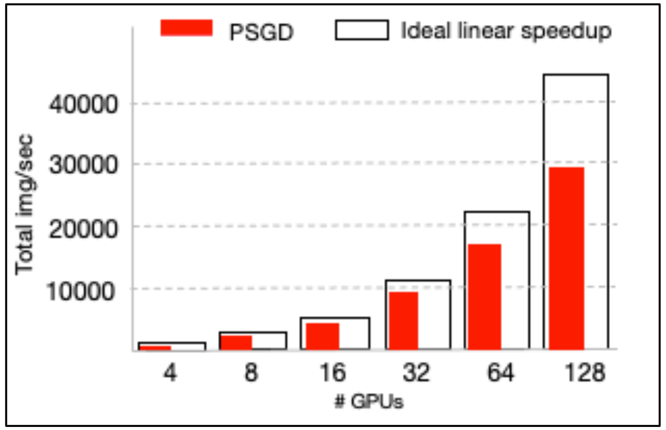
Vanilla parallel stochastic gradient descent (PSGD)



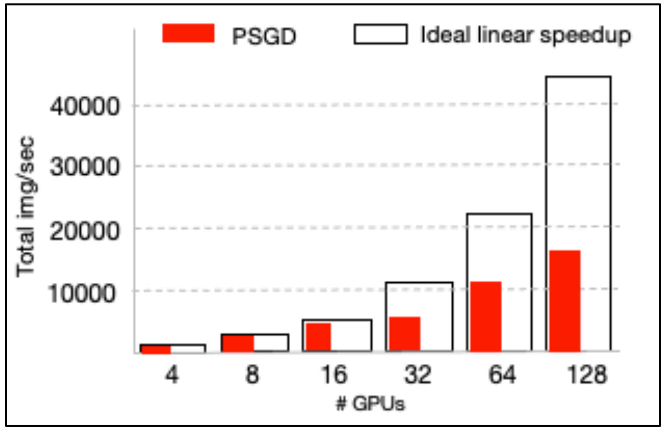
- Global average incurs $O(n)$ comm. overhead; **proportional to network size n**
- When network size n is large, PSGD suffers severe communication overhead

PSGD cannot achieve linear speedup due to comm. overhead

- PSGD cannot achieve ideal linear speedup in throughput due to comm. overhead
- Larger comm-to-compt ratio leads to worse performance in PSGD



Small comm.-to-compt. ratio



Large comm.-to-compt. ratio

- How can we accelerate PSGD? **Decentralized SGD is a promising paradigm.**

B. Ying, K. Yuan, H. Hu, Y. Chen and W. Yin, "BlueFog: Make decentralized algorithms practical for optimization and deep learning", arXiv: 2111. 04287, 2021

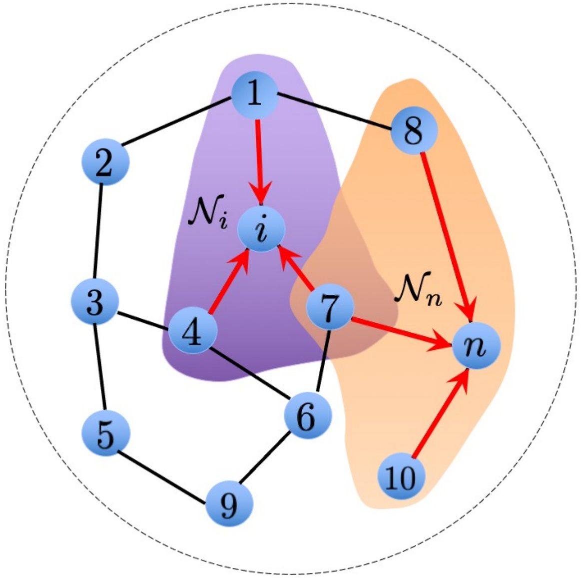
Decentralized SGD (DSGD)

北京大学 PEKING UNIVERSITY

- To break $O(n)$ comm. overhead, we replace global average with partial average

$$x_i^{(k+\frac{1}{2})} = x_i^{(k)} - \gamma \nabla F(x_i^{(k)}; \xi_i^{(k)}) \quad \text{(Local update)}$$

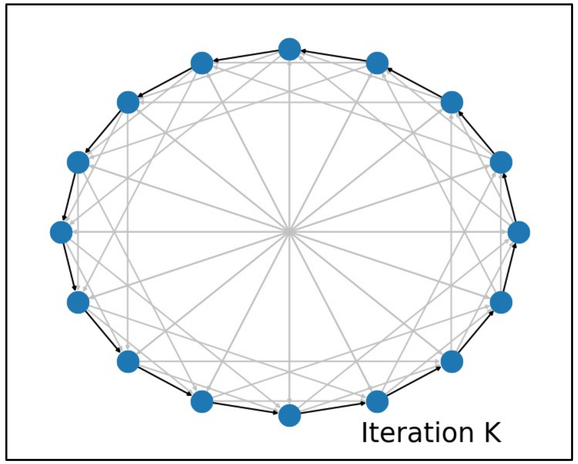
$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i} w_{ij} x_j^{(k+\frac{1}{2})} \quad \text{(Partial averaging)}$$

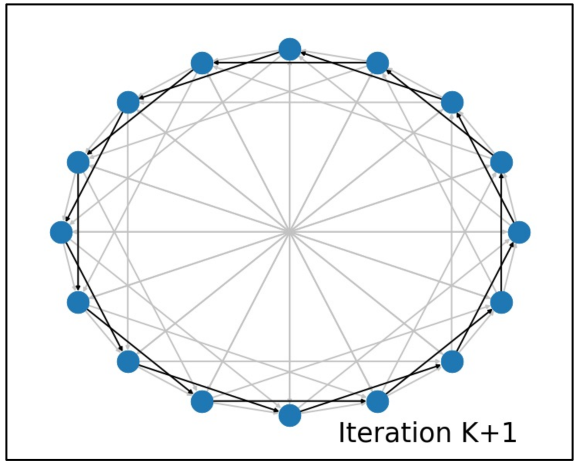


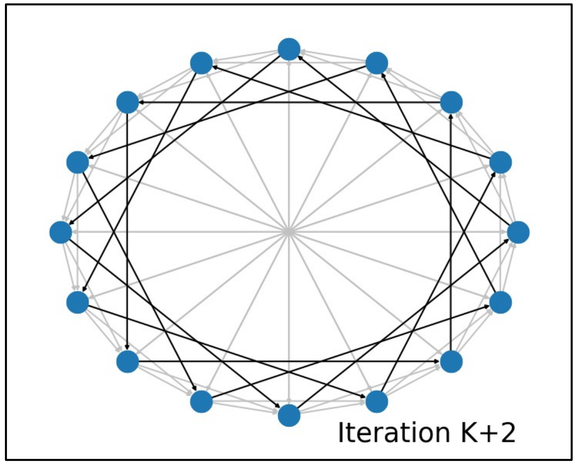
- DSGD = local SGD update + partial averaging [LS08]
- $\mathcal{N}_i$ is the set of neighbors at node i ; w_{ij} scales information from j to i
- Incurs $O(d_{\max})$ comm. overhead per iteration where $d_{\max} = \max_i \{|\mathcal{N}_i|\}$ is the graph maximum degree

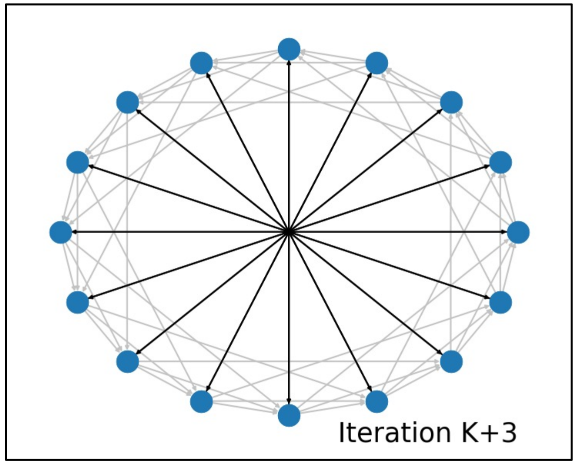
DSGD is more communication-efficient than PSGD

- Incurs $O(1)$ comm. overhead on **sparse** topologies; much less than global average $O(n)$
- Many sparse and effective topologies are proposed recently









B. Ying*, K. Yuan*, Y. Chen*, H. Hu, P. Pan, and W. Yin, "Exponential Graph is Provably Efficient for Deep Training", NeurIPS 2021

Z. Song*, W. Li*, K. Jin*, L. Shi, M. Yan, W. Yin, and K. Yuan "Communication-efficient topologies for decentralized learning with O(1) consensus rate", NeurIPS 2022

DSGD is more communication-efficient than PSGD

- A real experiment on a 256-GPUs cluster [CYZ+21]

Model	Ring-Allreduce	Partial average
ResNet-50 (25.5M)	278 ms	150 ms
Bert (300M)	1469 ms	567 ms

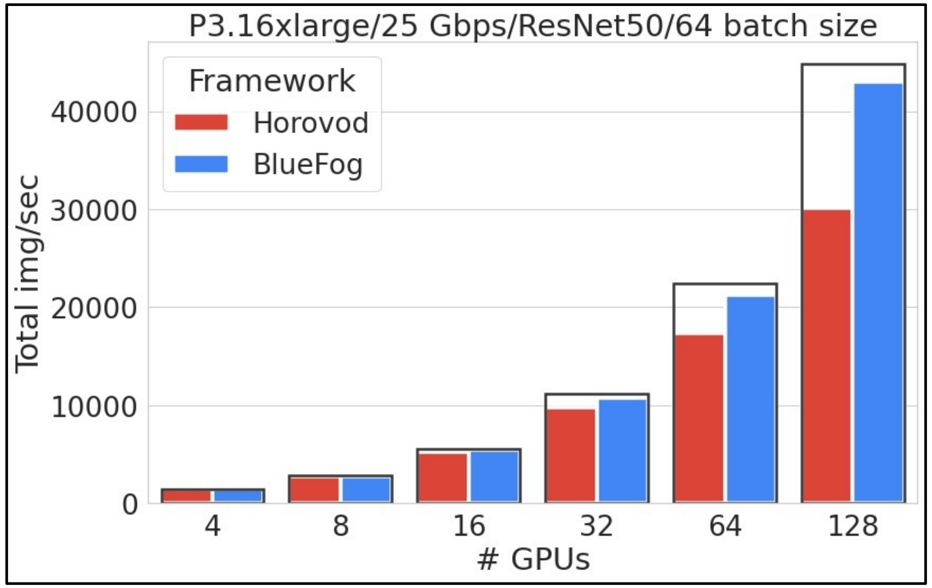
Table. Comparison of per-iter comm. time in terms of runtime with 256 GPUs

- DSGD saves more communications per iteration for larger models

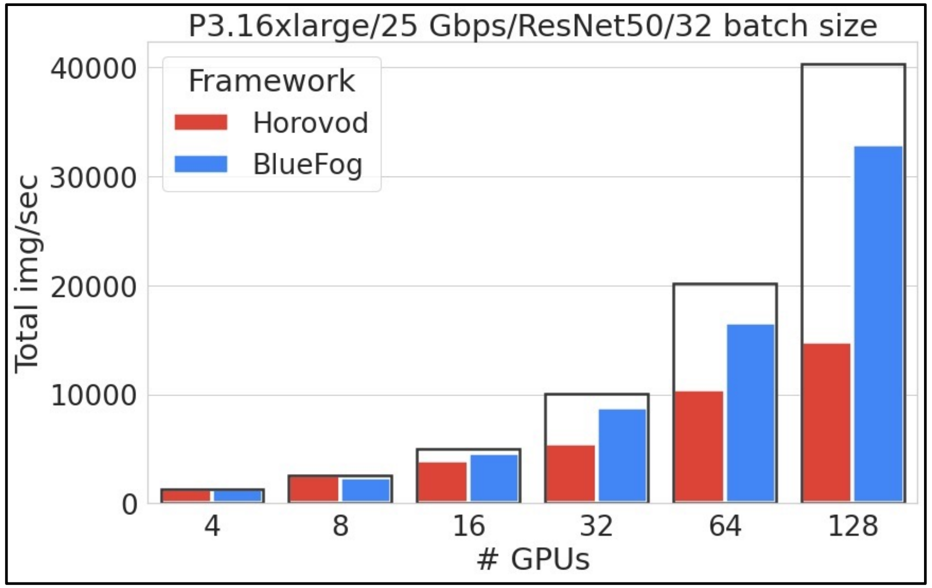
[CYZ+21] Y. Chen*, K. Yuan*, Y. Zhang, P. Pan, Y. Xu, and W. Yin, ``Accelerating Gossip SGD with Periodic Global Averaging", ICML 2021

DSGD is more communication-efficient than PSGD

- DSGD (BlueFog) has **better linear speedup** than PSGD (Horovod) due to its small comm. overhead



Small comm.-to-compt. ratio



Large comm.-to-compt. ratio

B. Ying, K. Yuan, H. Hu, Y. Chen and W. Yin, "BlueFog: Make decentralized algorithms practical for optimization and deep learning", arXiv: 2111. 04287, 2021

DSGD is more communication-efficient than PSGD

Table. Test accuracy and wall-clock training time on ImageNet [YYC+21]

nodes topology	4(4x8 GPUs) acc.	 time	8(8x8 GPUs) acc.	 time	16(16x8 GPUs) acc.	 time	32(32x8 GPUs) acc.	 time
P-SGD	76.32	11.6	76.47	6.3	76.46	3.7	76.25	2.2
D-SGD	76.34	11.1	76.52	5.7	76.47	2.8	76.27	1.5

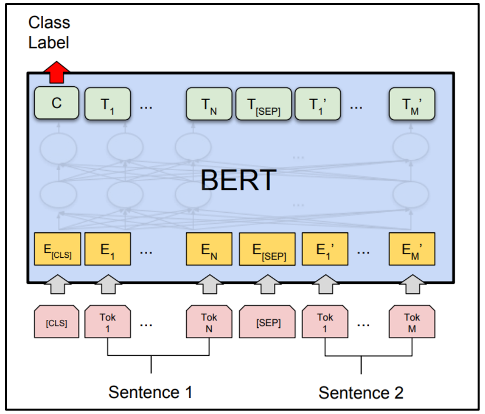


Table. Training loss and wall-clock training time on BERT [CYZ+21]

Method	Final Loss	Wall-clock Time (hrs)
P-SGD	1.75	59.02
D-SGD	1.77	30.4

[YYC+21] B. Ying*, K. Yuan*, Y. Chen*, H. Hu, P. Pan, and W. Yin, “Exponential Graph is Provably Efficient for Deep Training”, NeurIPS 2021

[CYZ+21] Y. Chen*, K. Yuan*, Y. Zhang, P. Pan, Y. Xu, and W. Yin, ``Accelerating Gossip SGD with Periodic Global Averaging”, ICML 2021

This talk focuses on decentralized momentum SGD (DmSGD)

- DSGD performs **badly** in ill-conditioned stochastic optimization; seldom used in real practice

$$x_i^{(k+\frac{1}{2})} = x_i^{(k)} - \gamma \nabla F(x_i^{(k)}; \xi_i^{(k)}) \quad \text{(Local update)}$$

$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i} w_{ij} x_j^{(k+\frac{1}{2})} \quad \text{(Partial averaging)}$$

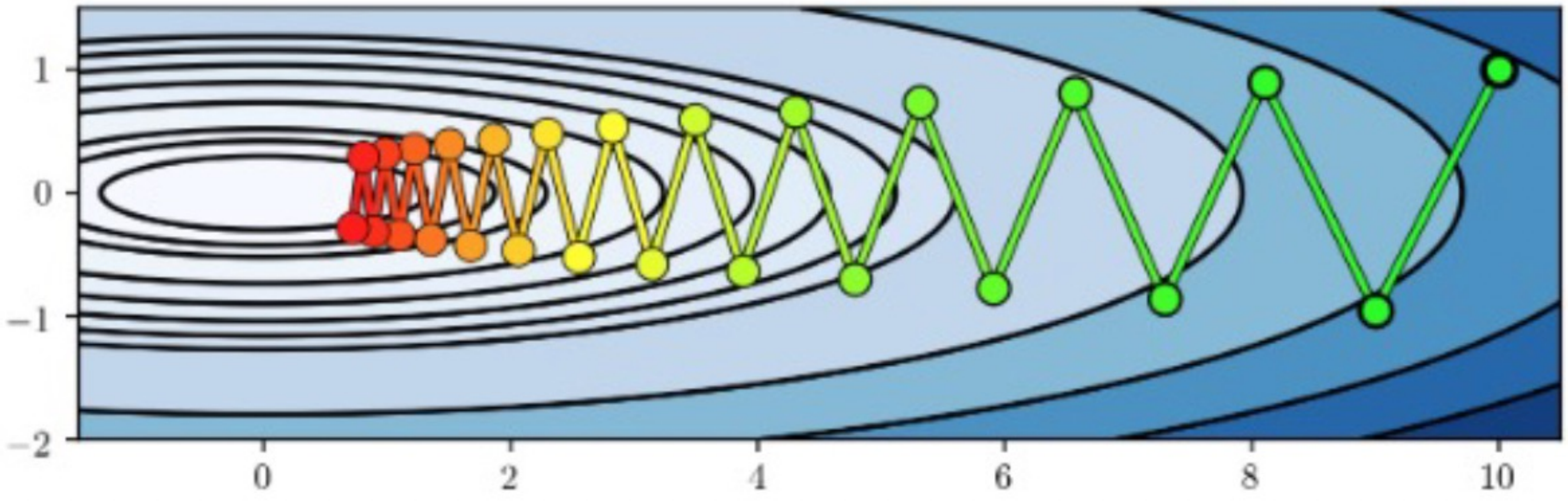


Image from "***Machine Learning Refined***"

This talk focuses on decentralized momentum SGD (DmSGD)

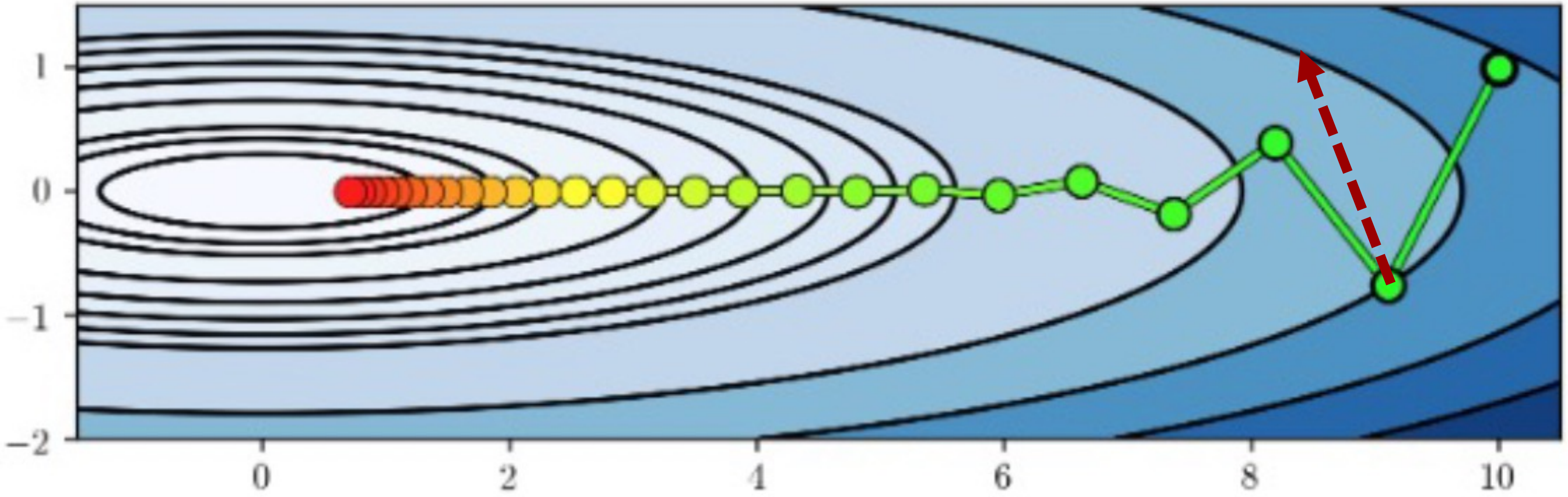
- DmSGD can alleviate the *"Zig-Zag"* and accelerate the convergence; widely used in real applications

$$m_i^{(k+1)} = \beta m_i^{(k)} + \nabla F(x_i^{(k)}; \xi_i^{(k)}) \quad \text{(Momentum update)}$$

$$x_i^{(k+\frac{1}{2})} = x_i^{(k)} - \gamma m_i^{(k+1)} \quad \text{(Local variable update)}$$

$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i} w_{ij} x_j^{(k+\frac{1}{2})} \quad \text{(Partial averaging)}$$

Reduce to DSGD when $\beta = 0$



Large-batch training is a must in large-scale deep learning

- Total batch size increases as the number of nodes (GPUs) grows
- Suppose each node takes 256 samples per iteration:

(8 nodes) $256 \times 8 = 2K$ (samples)

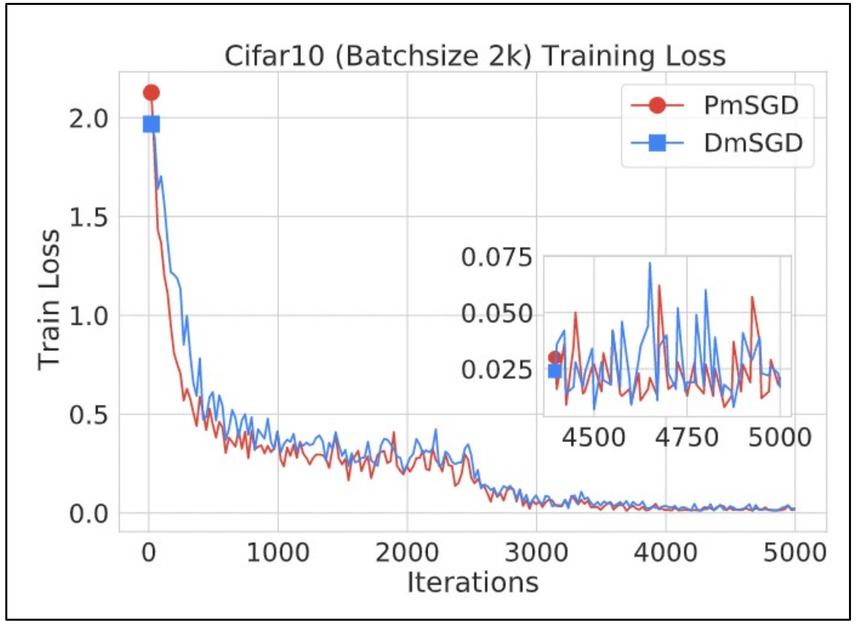
(64 nodes) $256 \times 64 = 16K$ (samples)

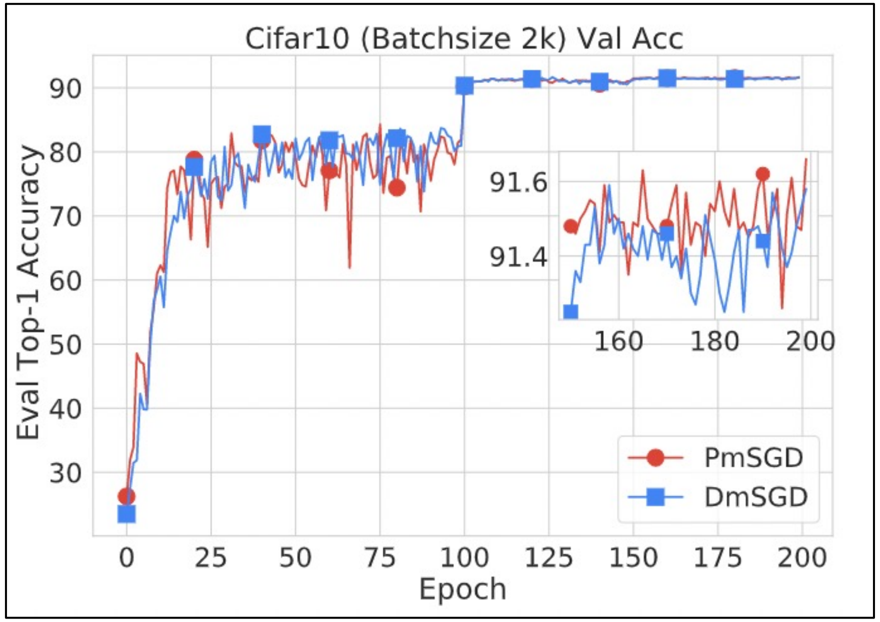
(256 nodes) $256 \times 256 = 64K$ (samples)

- Large-batch training is a **must** for large-scale deep training with massive number of GPUs

DmSGD performs well in small-batch scenario

- Experimental setting: CIFAR-10; ResNet-20
- Small-batch: 2K total batch-size per iteration
- Baseline: parallel (centralized) momentum SGD (PmSGD)

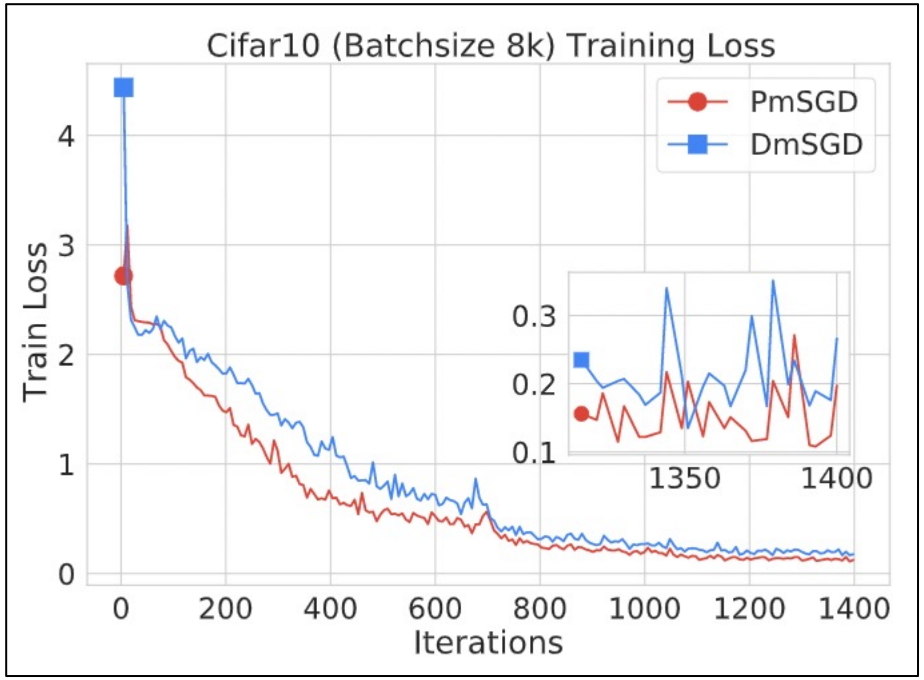


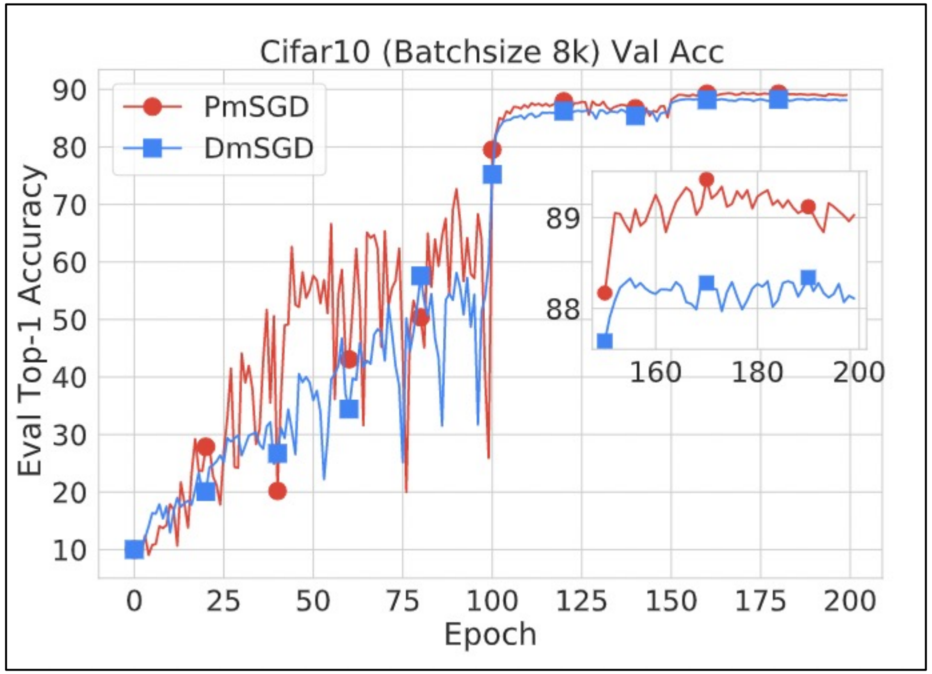


DmSGD and PmSGD have almost the **same** performance with small-batch

However, DmSGD performs badly in large-batch scenario

- Experimental setting: CIFAR-10; ResNet-20
- Large-batch: 8K total batch-size per iteration





DmSGD **drops 1% performance** compared to PmSGD with large-batch

Two critical questions:

- Why does DmSGD have severe performance degradation with large batch size?

- Can we overcome such degradation?

PART 02

Reason behind Performance Degradation

DmSGD limiting bias

- The limiting bias of DSGD/DmSGD (s.c. cost) typically suffers from two sources

$$\lim_{k\to\infty} \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\|x_i^{(k)} - x^\star\|^2 = \text{sto. bias} + \text{inconsist. bias}$$

  - Stochastic bias is caused by the gradient noise
  - Inconsistency bias is caused by data heterogeneity (i.e., different distribution $\mathcal{D}_i$)

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

DmSGD limiting bias: an illustration

- Take DSGD as an example, its limiting bias (s.c. cost) is derived as [YAYS20]

$$\lim_{k\to\infty} \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\|x_i^{(k)} - x^\star\|^2 = O\Big(\underbrace{\frac{\gamma^2\sigma^2}{n} + \frac{\gamma^2\sigma^2}{1-\rho}}_{\text{sto. bias}} + \underbrace{\frac{\gamma^2 b^2}{(1-\rho)^2}}_{\text{inconsist. bias}}\Big)$$

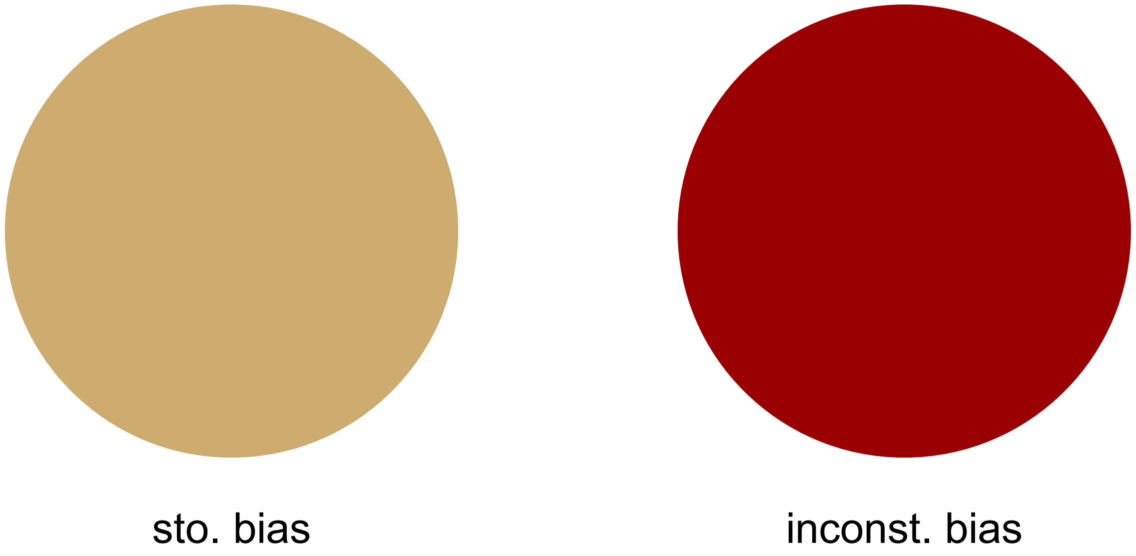
- Quantity $b^2 = \frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(x^\star)\|^2$ denotes data heterogeneity; $b^2 = 0$ when $f_i(x) = f_j(x) = f(x)$
- Quantity σ^2 denotes gradient noise; $\sigma^2 \to 0$ as batch-size grows large
- Quantity $\rho = \|W - \frac{1}{n}\mathbb{1}\mathbb{1}^T\| \in (0,1)$ characterizes the network topology connectivity

[YAYS20] K. Yuan, S. Alghunaim, B. Ying and A. Sayed, “On the influence of bias-correction on distributed stochastic optimization”, IEEE TSP, 2020

Inconsistency bias dominates large-batch setting

Proposition. Inconsistency bias dominates convergence of large-batch DmSGD.

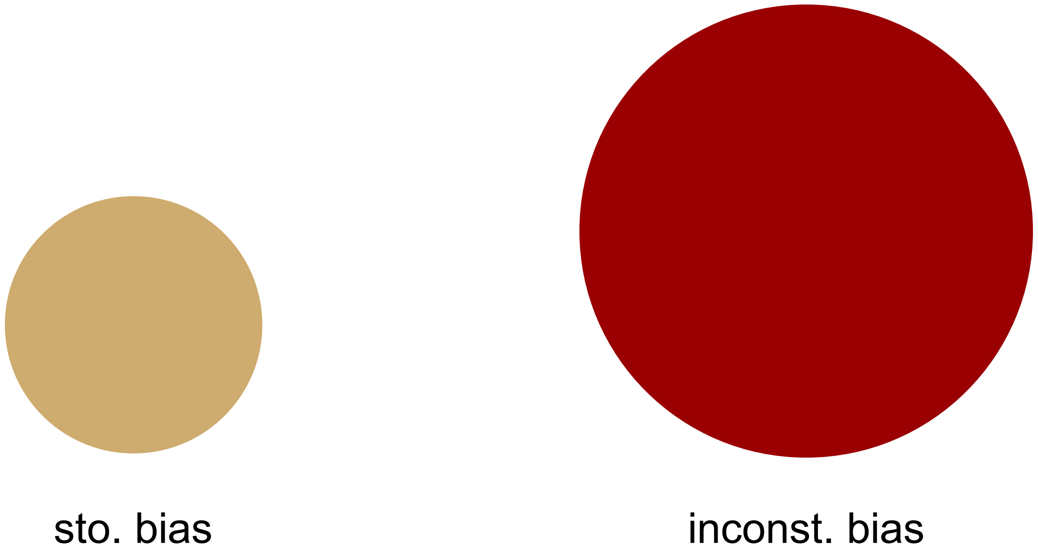
Small-batch setting



Inconsistency bias dominates large-batch setting

Proposition. Inconsistency bias dominates convergence of large-batch DmSGD.

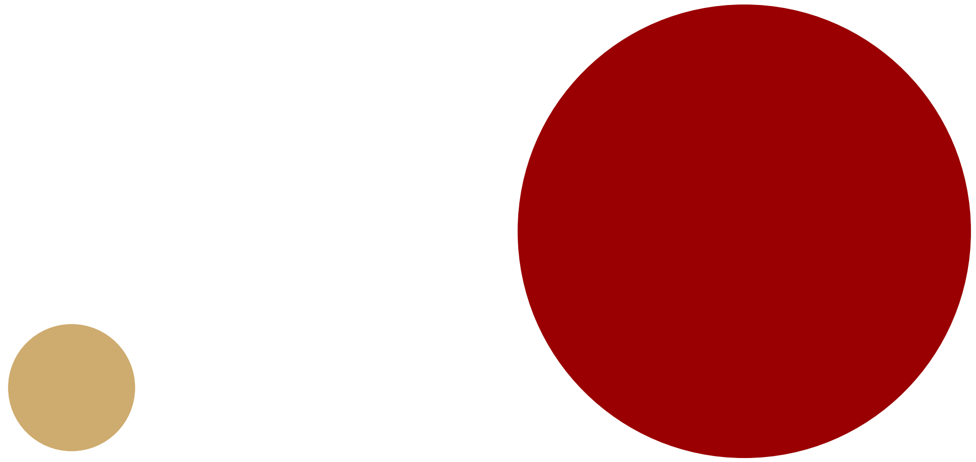
Midium-batch setting



Inconsistency bias dominates large-batch setting

Proposition. Inconsistency bias dominates convergence of large-batch DmSGD.

Large-batch setting



sto. bias

inconst. bias

DmSGD incurs severe inconsistency bias

- Therefore, it is enough to examine the inconsistency bias in large-batch setting

- We rewrite **full-batch** DmSGD as

$$m_i^{(k+1)} = \beta m_i^{(k)} + \nabla f_i(x_i^{(k)}) \quad \text{(Momentum update)}$$

$$x_i^{(k+\frac{1}{2})} = x_i^{(k)} - \gamma m_i^{(k+1)} \quad \text{(Local variable update)}$$

$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i} w_{ij} x_j^{(k+\frac{1}{2})} \quad \text{(Partial averaging)}$$

DmSGD incurs severe inconsistency bias

北京大學 PEKING UNIVERSITY

- Therefore, it is enough to examine the inconsistency bias in large-batch setting

- We rewrite full-batch DmSGD as

$$
\begin{aligned}
x_i^{(k+1)} = & \underbrace{\sum_{j \in \mathcal{N}_i} w_{ij} \left(x_j^{(k)} - \gamma \nabla f_j(x_j^{(k)}) \right)}_{\text{DSGD}} \\
& + \underbrace{\beta \left(x_i^{(k)} - \sum_{j \in \mathcal{N}_i} w_{ij} x_j^{(k-1)} \right)}_{\text{momentum}}, \quad \forall i \in [n]. \qquad \text{(DmSGD)}
\end{aligned}
$$

DmSGD incurs severe inconsistency bias

$$x_i^{(k+1)} = \underbrace{\sum_{j\in\mathcal{N}_i} w_{ij}\Big(x_j^{(k)} - \gamma\nabla f_j(x_j^{(k)})\Big)}_{\text{DSGD}} + \underbrace{\beta\Big(x_i^{(k)} - \sum_{j\in\mathcal{N}_i} w_{ij}x_j^{(k-1)}\Big)}_{\text{momentum}}, \quad \forall i \in [n]. \qquad \text{(DmSGD)}$$

- Momentum will not vanish as $x_i^{(k)} \neq \sum_{j\in\mathcal{N}_i} w_{ij}x_j^{(k-1)}$ as $k \to \infty$
- Compared to DSGD, momentum will incur additional inconsistency bias

DmSGD incurs severe inconsistency bias

Proposition. The full-batch DmSGD (S.C. cost) has the following inconsistency bias:

$$\lim_{k\to\infty} \frac{1}{n}\sum_{i=1}^{n} \|x_i^{(k)} - x^\star\|^2 = O\Big(\frac{\gamma^2 b^2}{(1-\beta)^2(1-\rho)^2}\Big),$$

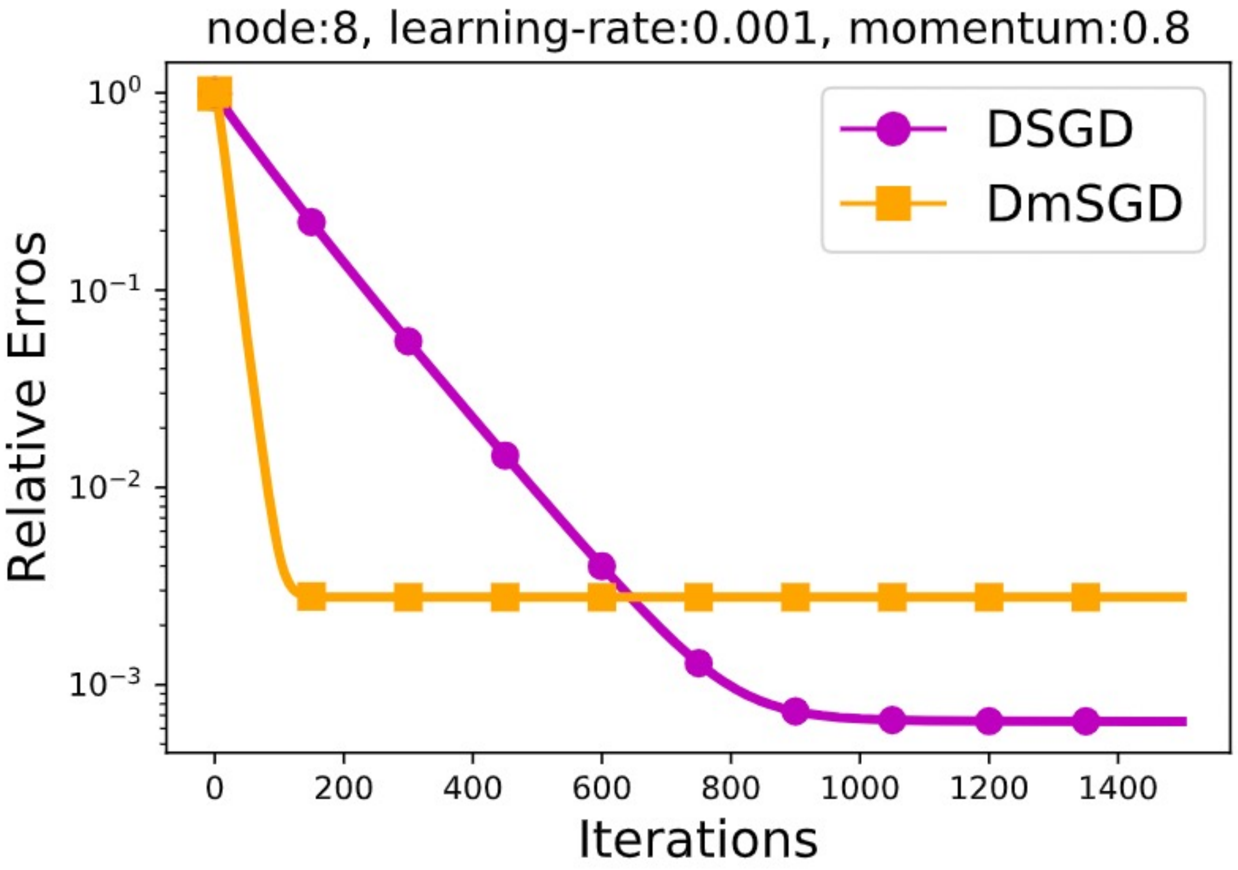
where $b^2 = (1/n)\sum_{i=1}^{n} \|\nabla f_i(x^\star)\|^2$ denotes the data inconsistency between nodes, and β is the momentum coefficient.

- Recall that full-batch DSGD has limiting bias as $O(\gamma^2 b^2/(1-\rho)^2)$
- The momentum in DmSGD **amplifies** inconsistency bias as $\beta \in (0, 1)$

DmSGD incurs severe inconsistency bias: verification

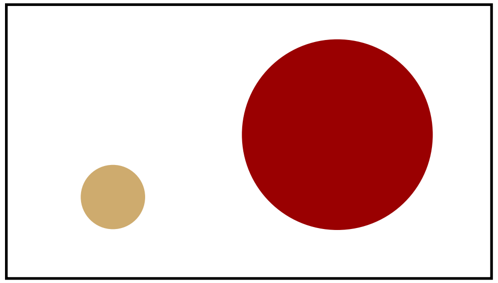
- Full-batch linear regression

- DmSGD is faster but suffers more inconsistency bias (as expected)



A brief summary

- Inconsistency bias dominates large-batch setting



- Momentum amplifies inconsistency bias in DmSGD especially when $\beta \to 1$

$$\lim_{k\to\infty} \frac{1}{n}\sum_{i=1}^{n} \|x_i^{(k)} - x^\star\|^2 = O\Big(\frac{\gamma^2 b^2}{(1-\beta)^2(1-\rho)^2}\Big)$$

- This explains why DmSGD gets poor performance in large-batch setting

PART 03

DecentLaM: Remove Momentum-Incurred Bias

DmSGD incurs severe inconsistency bias

$$x_i^{(k+1)} = \underbrace{\sum_{j \in \mathcal{N}_i} w_{ij}\Big(x_j^{(k)} - \gamma \nabla f_j(x_j^{(k)})\Big)}_{\text{DSGD}} + \underbrace{\beta\Big(x_i^{(k)} - \sum_{j \in \mathcal{N}_i} w_{ij} x_j^{(k-1)}\Big)}_{\text{momentum}}, \quad \forall i \in [n]. \qquad \text{(DmSGD)}$$

- Momentum will not vanish as $x_i^{(k)} \neq \sum_{j \in \mathcal{N}_i} w_{ij} x_j^{(k-1)}$ as $k \to \infty$
- Compared to DSGD, momentum will incur additional inconsistency bias

Remove momentum-incurred bias

- We modify the momentum term a little bit

$$x_i^{(k+1)} = \underbrace{\sum_{j \in \mathcal{N}_i} w_{ij} \Big(x_j^{(k)} - \gamma \nabla f_j (x_j^{(k)}) \Big)}_{\text{DSGD}} \qquad \text{(DecentLaM)}$$

$$+ \underbrace{\beta \Big(x_i^{(k)} - x_i^{(k-1)} \Big)}_{\text{momentum}}, \ \forall i \in [n].$$

  - $x_i^{(k)} - x_i^{(k-1)} \to 0$ as $k \to \infty$
  - Momentum-incurred bias will vanish as $k \to \infty$

Remove momentum-incurred bias

北京大学 PEKING UNIVERSITY

Proposition. Full-batch DecentLaM (S.C. cost) has an inconsistency bias as

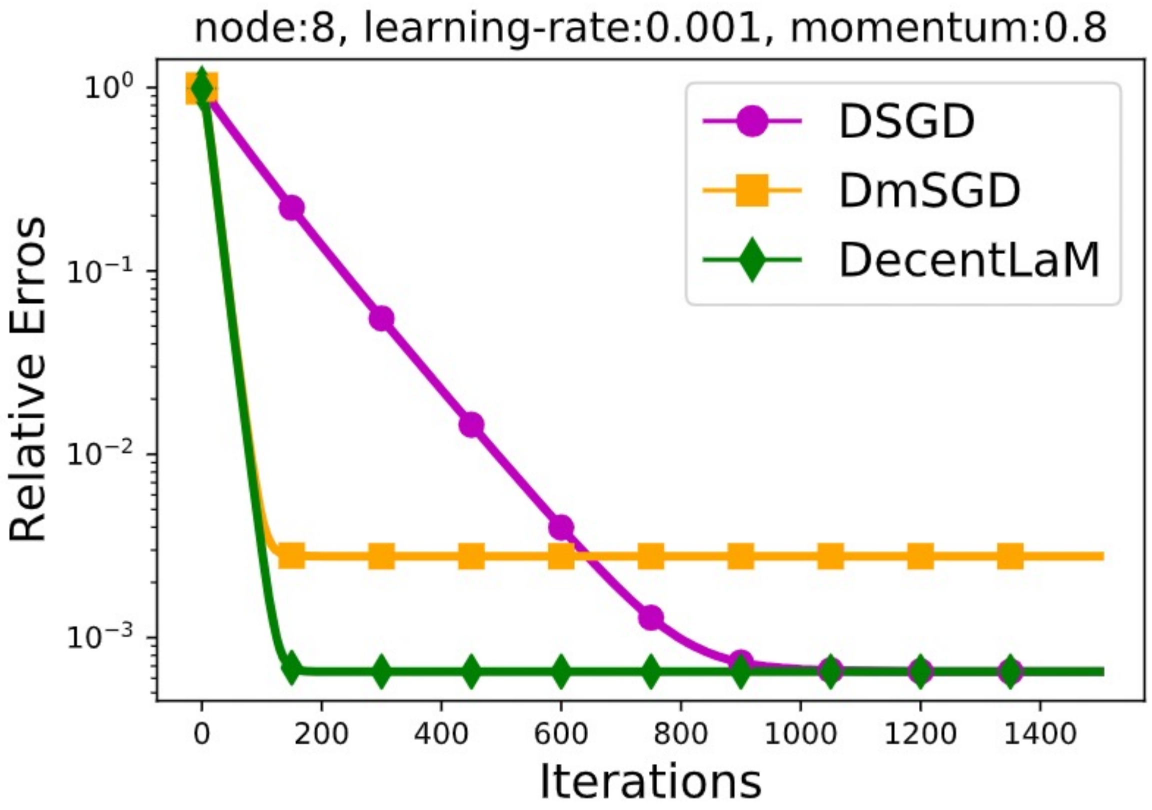
$$\lim_{k\to\infty} \frac{1}{n}\sum_{i=1}^{n} \|x_i^{(k)} - x^\star\|^2 = O\Big(\frac{\gamma^2 b^2}{(1-\rho)^2}\Big)$$

- Recall that full-batch DmSGD has limiting bias as $O(\frac{\gamma^2 b^2}{(1-\beta)^2(1-\rho)^2})$
- DecentLaM removes the momentum-incurred bias
- With smaller inconsist. bias, DecentLaM is expected to outperform DmSGD in large-batch scenario

Remove momentum-incurred bias: verification

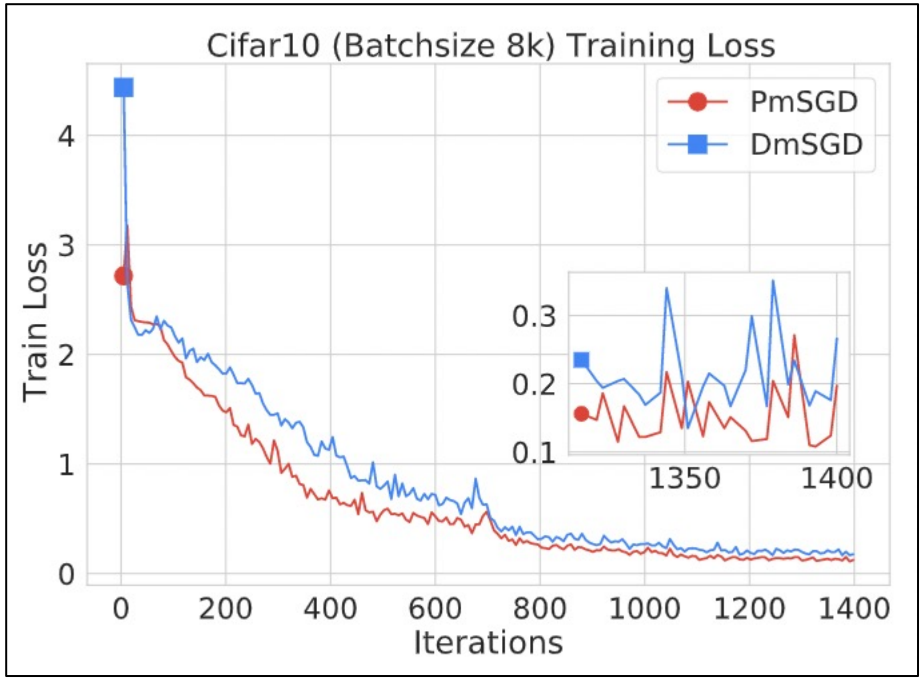
- Full-batch linear regression

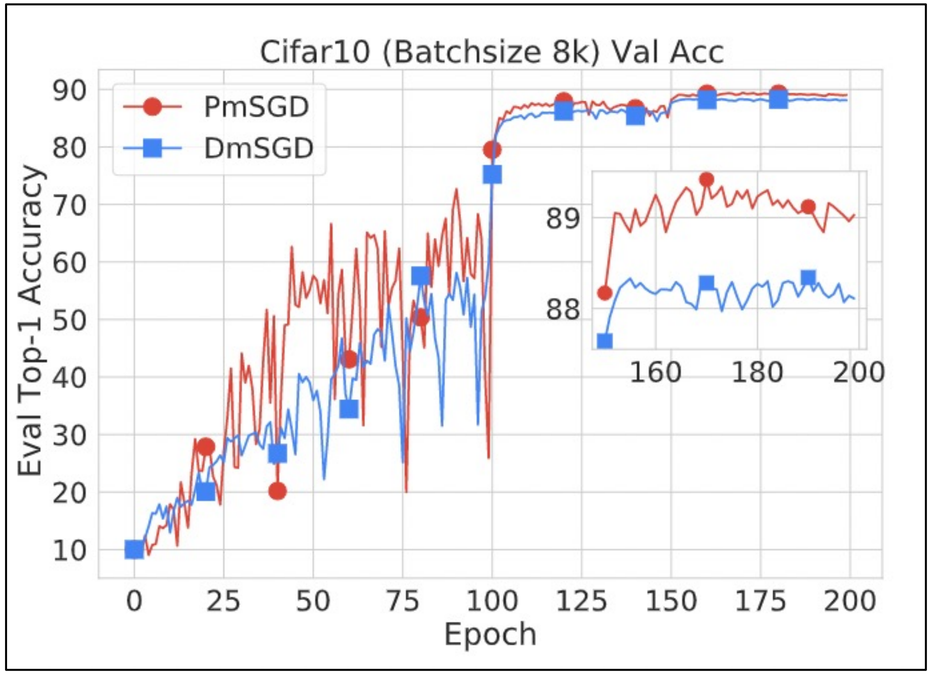
- DecentLaM is as fast as DmSGD, and as accurate as DSGD



However, DmSGD performs badly in large-batch scenario

- Experimental setting: CIFAR-10; ResNet-20
- Large-batch: 8K total batch-size per iteration

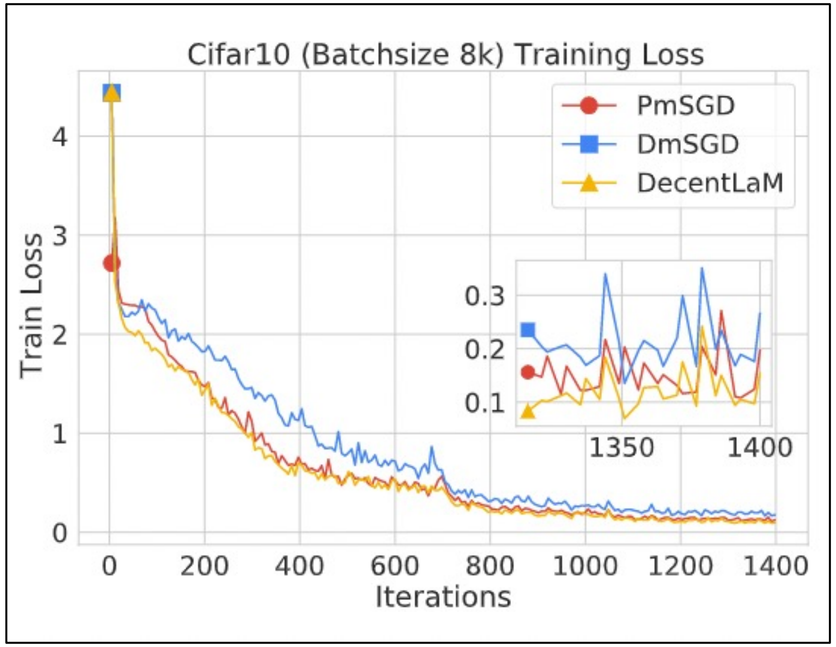


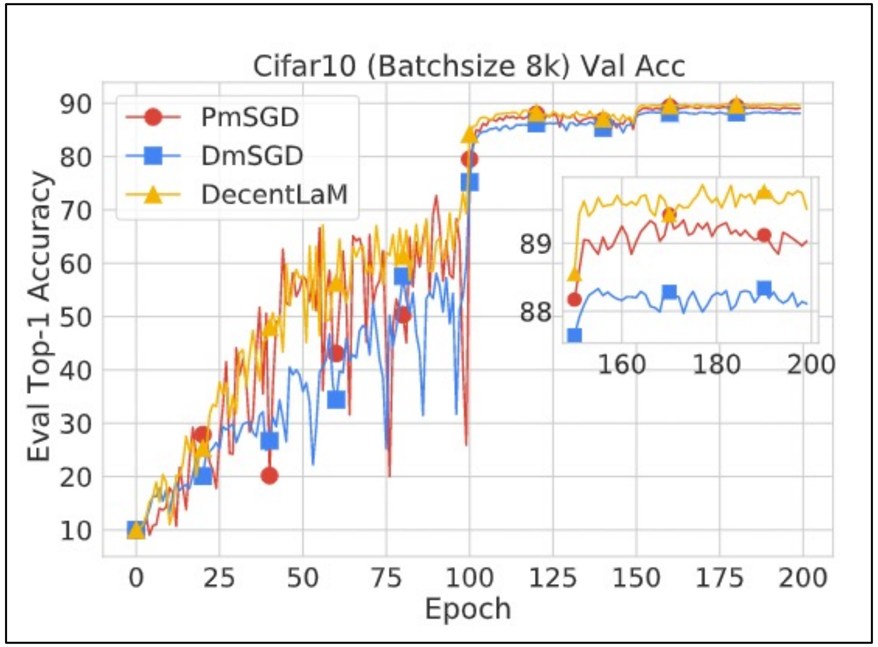


DmSGD **drops 1% performance** compared to PmSGD with large-batch

Go back to large-batch Cifar-10 experiment

- Experimental setting: CIFAR-10; ResNet-20
- Large-batch: 8K total batch-size per iteration





DecentLaM is much better than DmSGD, and is even better than PmSGD

Formal convergence in the non-convex scenario

Assumption. (A.1) Each $f_i(x)$ is L-smooth; (A.2) The gradient noise is unbiased and has bounded variance; (A.3) W is positive definite and doubly-stochastic; (A.4) Data heterogeneity is bounded: $\frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(x)-\nabla f(x)\|^2 \le b^2$ (A.5) Parameter β cannot be too close to 1

Theorem. With appropriate constant learning rate γ (see the paper), DecentLaM will converge at

$$\frac{1}{T}\sum_{k=0}^{T-1}\mathbb{E}\|\frac{1}{n}\sum_{i=1}^{n}\nabla f_i(\bar{x}^{(k)})\|^2$$

$$= O\Big(\underbrace{\frac{1-\beta}{\gamma T}}_{\text{convg. rate}} + \underbrace{\frac{\gamma\sigma^2}{n(1-\beta)} + \frac{\gamma^2\sigma^2}{1-\rho}}_{\text{sto. bias}} + \boxed{\underbrace{\frac{\gamma^2 b^2}{(1-\rho)^2}}_{\text{inconsist.bias}}}\Big)$$

removed momentum-incurred bias

DecentLaM suffers smallest inconsistency bias

北京大学 PEKING UNIVERSITY

	Strongly-convex	Non-convex
DmSGD[GH20]	N.A.	$O\left(\frac{\gamma^2 M^2}{(1-\beta)^2}\right)$
DmSGD[SDGD20]	$O\left(\frac{\gamma^{5/2} M^2}{(1-\beta)^6}\right)$	$O\left(\frac{\gamma^2 M^2}{(1-\beta)^4}\right)$
DmSGD	$O\left(\frac{\gamma^2 b^2}{(1-\beta)^2}\right)$	N.A
DA-DmSGD[YJY19]	N.A.	$O\left(\frac{\gamma^2 b^2}{(1-\beta)^2}\right)$
AWC-DmSGD[BJT+20]	$O\left(\frac{\gamma^2 M^2}{(1-\beta)^2}\right)$	$O\left(\frac{\gamma^2 M^2}{(1-\beta)^4}\right)$
QG-DmSGD[LPSJ21]	N.A	$O(\gamma^2 b^2)$
DecentLaM (Ours)	$\boldsymbol{O(\gamma^2 b^2)}$	$\boldsymbol{O(\gamma^2 b^2)}$

Note: quantity M is typically far larger than b

Experiments in deep training (image classification)

ImageNet-1K dataset

1.3M training images

50K test images

1K classes

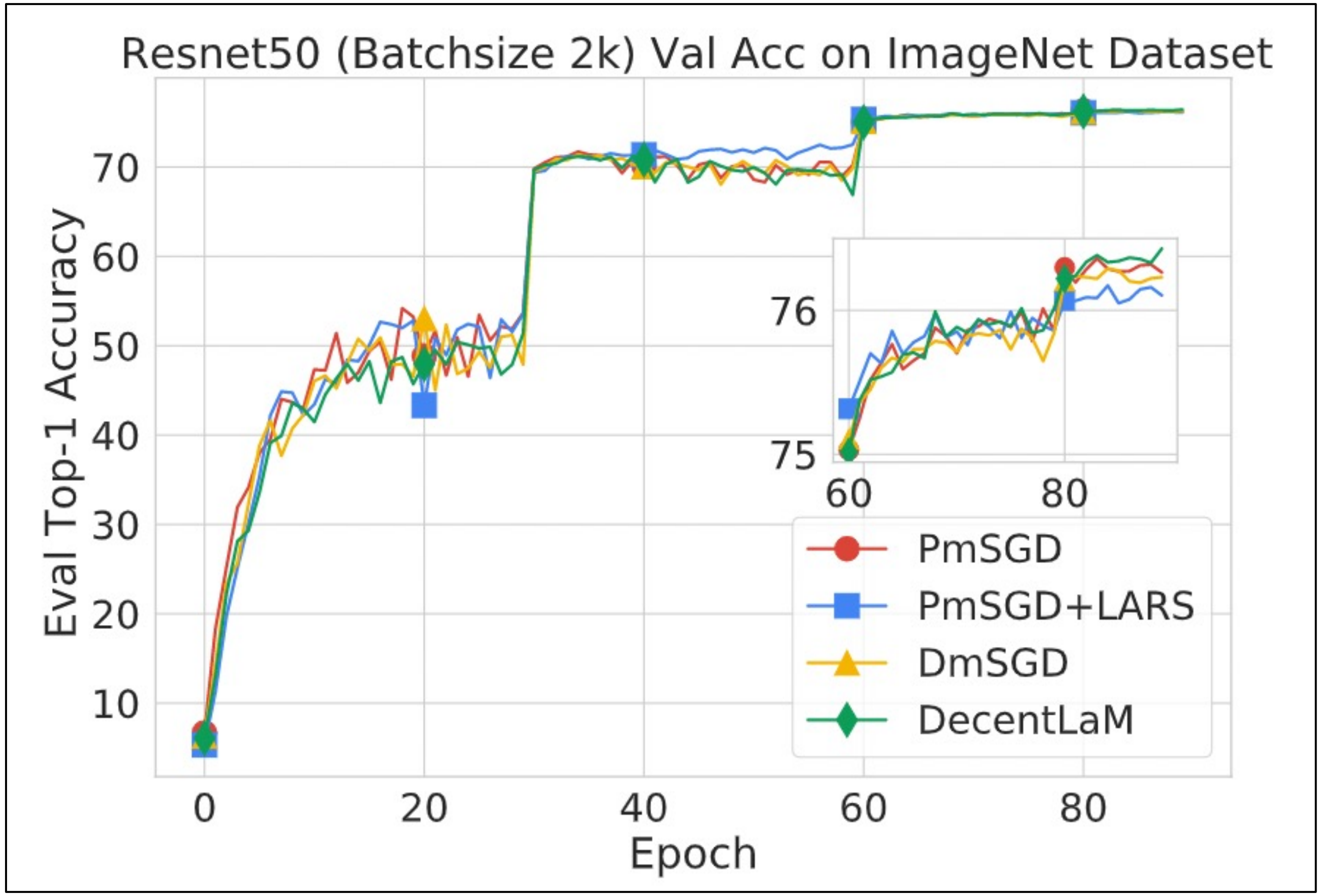
DNN model: ResNet-50 (25.5M parameters)

GPU: Up to 64 Tesla V100 GPUs

- **Batch-size:** we will test batch-sizes 2K, 16K, and 32K
- **Baseline:** PmSGD, PmSGD + LARS (layer-wise learning rate), DmSGD

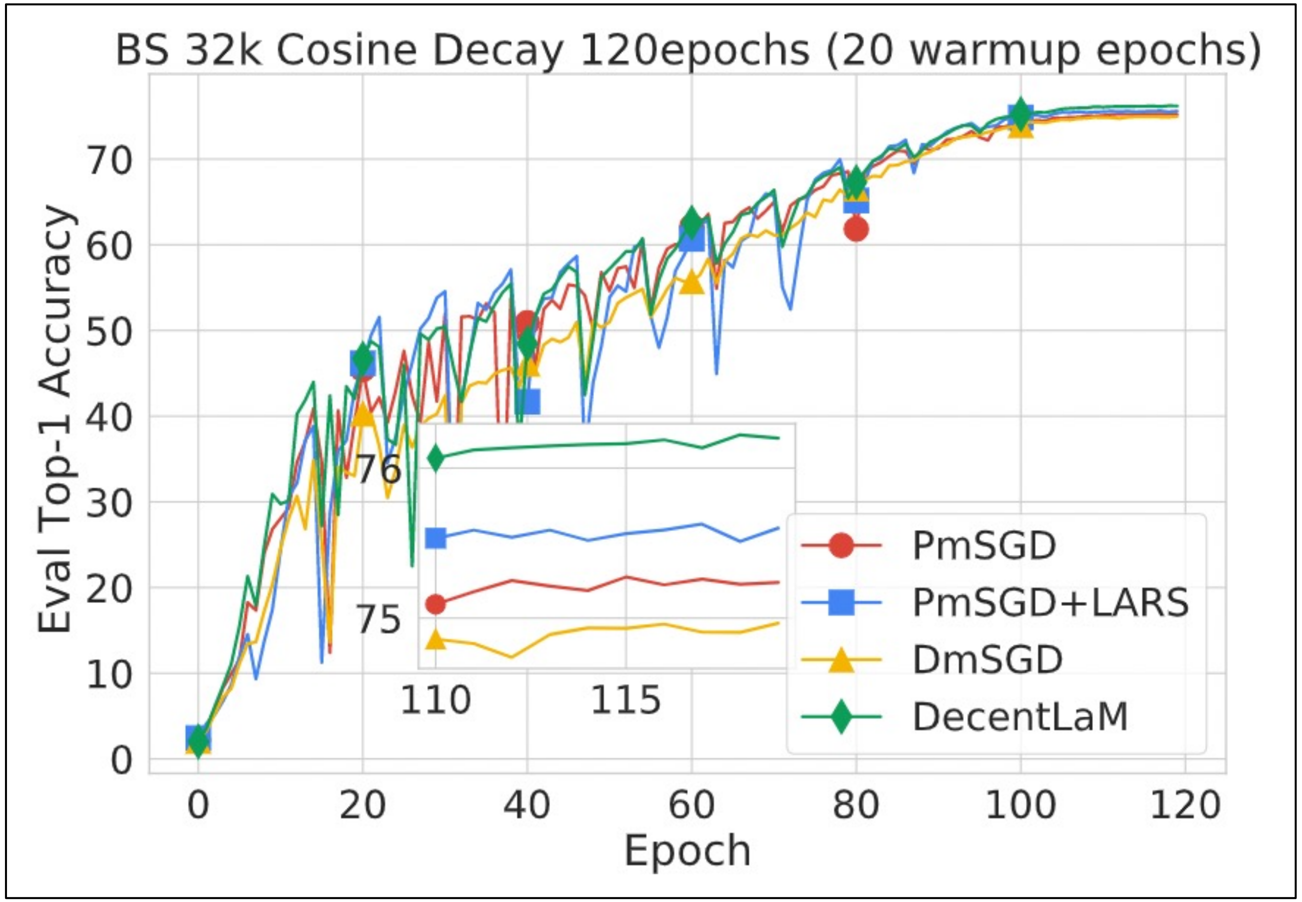
Experiments with batch-size 2K (test accuracy)

- Sto. bias dominates in **2K** batch-size
- DecentLaM performs similarly to DmSGD (as expected)



Experiments with batch-size 32K (test accuracy)

- Inconst. bias dominates in **32K** batch-size
- DecentLaM outperforms DmSGD significantly (as expected)
- DecentLaM even outperforms PmSGD with LARS



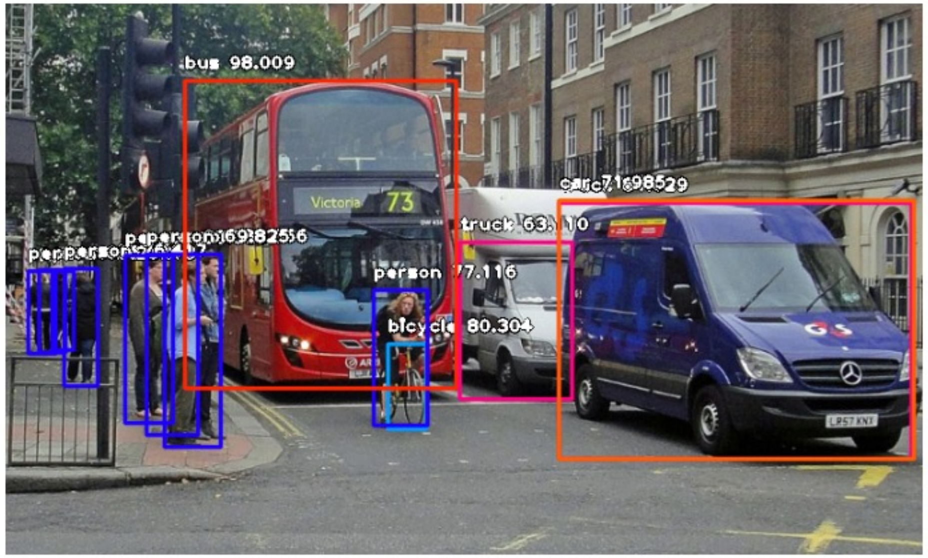
Comparison with more baselines

method	Batch Size			
	2k	8k	16k	32k
PmSGD	76.32	76.08	76.27	75.27
PmSGD+LARS	76.16	75.95	76.65	75.63
DmSGD	76.27	76.01	76.23	74.97
DA-DmSGD	76.35	76.19	76.62	75.51
AWC-DmSGD	76.29	75.96	76.31	75.37
SlowMo	76.30	75.47	75.53	75.33
QG-DmSGD	76.23	75.96	76.60	75.86
D^2-DmSGD	75.44	75.30	76.16	75.44
DecentLaM (Ours)	76.43	76.19	76.73	76.22

Outperforms all other baselines significantly for large-batch settings

Experiments in deep training (Object detection)

PASCAL/COCO dataset



Dataset	PASCAL VOC		COCO	
Model	R-Net	F-RCNN	R-Net	F-RCNN
DmSGD	79.1	80.5	36.1	36.4
DecentLaM	**79.3**	**80.7**	**36.6**	**37.1**

PART 04

BlueFog: An open-source and high-performance python library

BlueFog

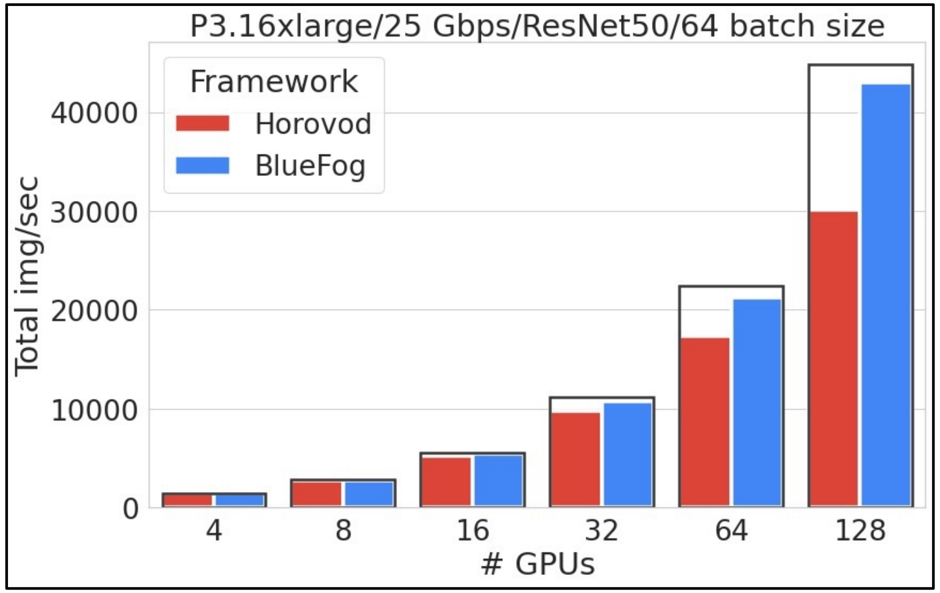
https://github.com/Bluefog-Lib/bluefog

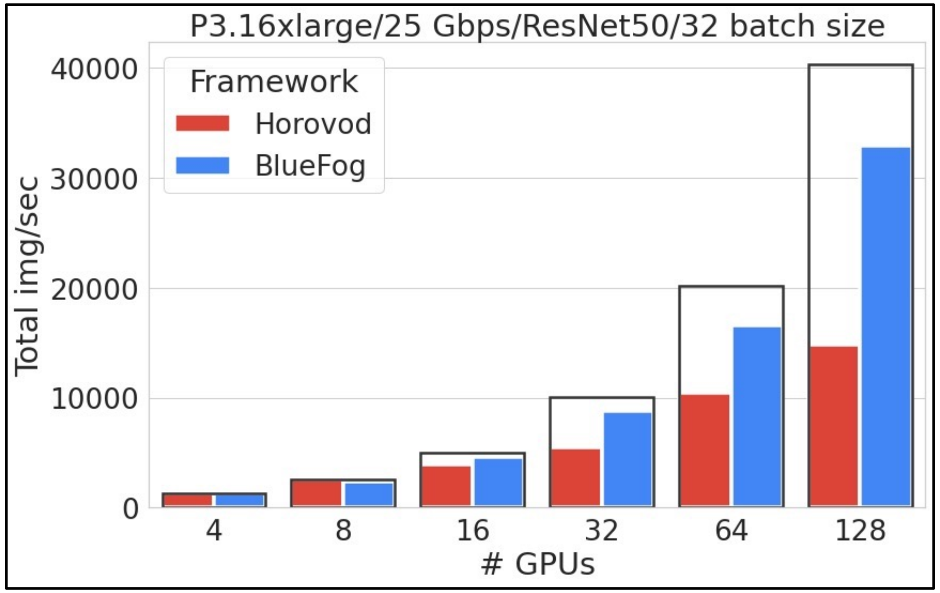
BlueFog

- An open-source library to support decentralized communication in optimization and deep learning
- High-performance
- Easy-to-use

High-performance

- BlueFog has larger throughput than Horovod (the SOTA DL system implementing PSGD) [YYH+21]





- All our research progresses are involved in BlueFog

[YYH+21] B. Ying, K. Yuan, H. Hu, Y. Chen, and W. Yin, ``BlueFog: Make Decentralized Algorithms Practical for Optimization and machine learning", arXiv:2111.04287 [GitHub site: github.com/Bluefog-Lib/bluefog]

Easy-to-use

PEKING UNIVERSITY

- Writing codes for decentralized methods is as easy as writing equations

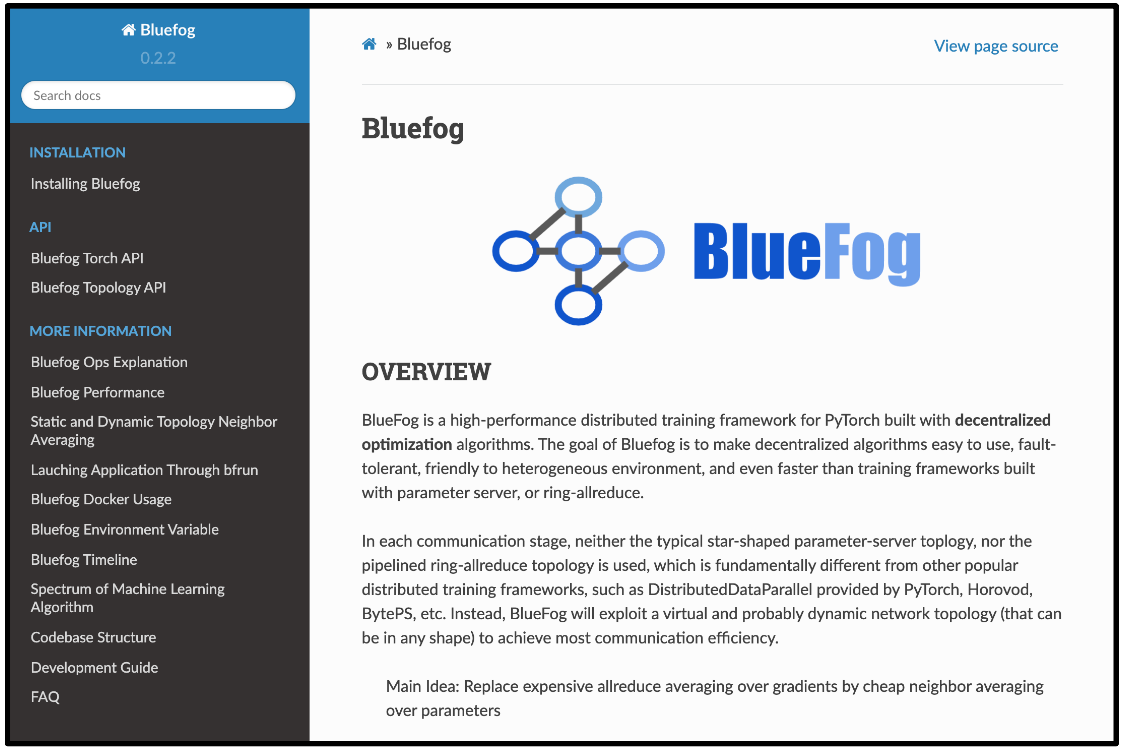
Decentralized least-square algorithms

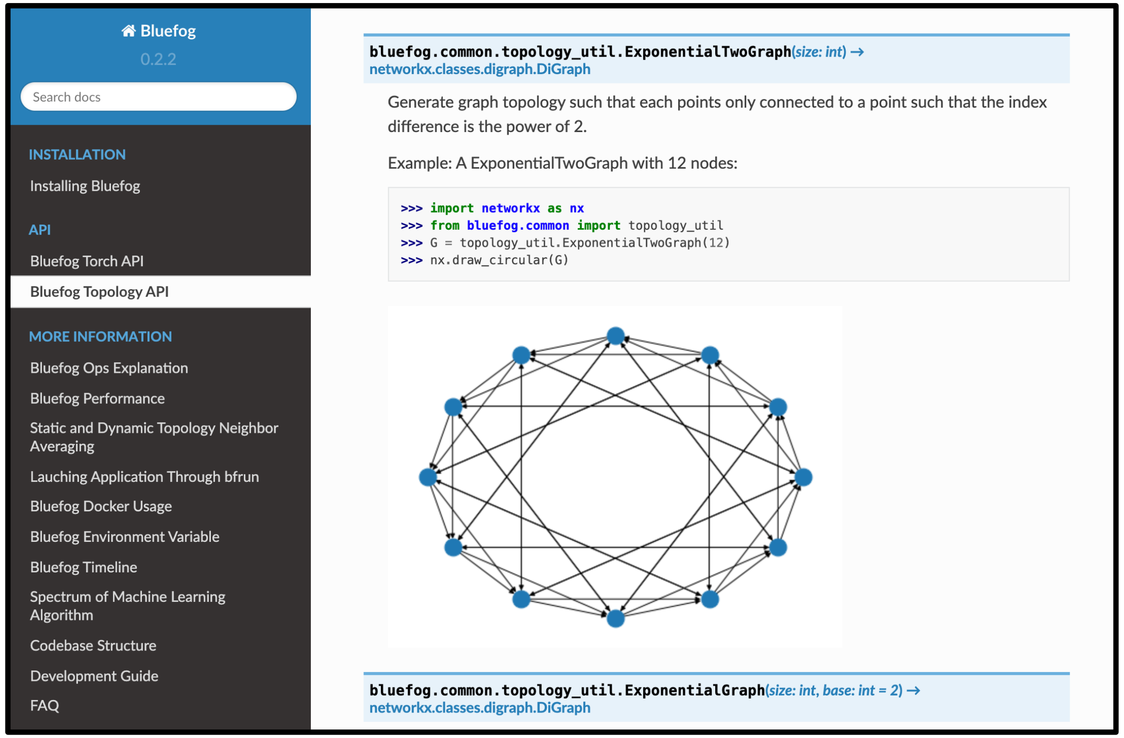
$$y_i^{(k)} = x_i^{(k)} - \gamma A_i^T(A_i x_i^{(k)} - b_i)$$
$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i} w_{ij} y_j^{(k)}$$

```python
import bluefog.torch as bf
bf.init()   # Initialize the BlueFog

# Set topology as static exponential graph.
G = bf.ExponentialTwoGraph(bf.size())
bf.set_topology(G)

# DGD implementation
for ite in range(maxite):
    grad_local = A.t().mm(A.mm(x) - b)  # compute local grad
    y = x - gamma * grad_local          # local update
    x = bf.neighbor_allreduce(y)        # partial averaging
```

Easy-to-use

Abundant documents





Easy-to-use

北京大学 PEKING UNIVERSITY

Detailed tutorials

Contents

2.1.3 Initialize BlueFog and test it

All contents in this section are displayed in Jupyter notebook, and all experimental examples are written with BlueFog and iParallel. Readers not familiar with how to run BlueFog in ipython notebook environment is encouraged to read Sec. [HelloWorld section] first. In the following codes, we will initialize BlueFog and test whether it works normally.

The output of `rc.ids` should be a list from 0 to the number of processes minus one. The number of processes is the one you set in the `ibfrun start -np {X}`.

In [1]:
```
import ipyparallel as ipp

rc = ipp.Client(profile="bluefog")
rc.ids
```

Let each agent import necessary modules and then initialize BlueFog. You should be able to see the printed information like:

> [stdout:0] Hello, I am 1 among 4 processes
>
> ...

In [2]:
```
%%px
import numpy as np
import bluefog.torch as bf
import torch
from bluefog.common import topology_util
import networkx as nx

bf.init()
print(f"Hello, I am {bf.rank()} among {bf.size()} processes")
```

Push seed to each agent so that the simulation can be reproduced.

In [3]:
```
dview = rc[:]  # A DirectView of all engines
dview.block = True

# Push the data into all workers
#   `dview.push({'seed': 2021}, block=True)`
# Or equivalently
dview["seed"] = 2021
```

After running the following code, you should be able to see the printed information like

> [stdout:0] I received seed as value: 2021

Final summary

北京大学 PEKING UNIVERSITY

- DmSGD suffers significant performance degradation in large-batch settings
- Root reason: inappropriate momentum amplifies inconsistency bias
- We propose DecentLaM to completely remove the momentum-incurred bias
- Theoretical and numerical results justify the superiority of DecentLaM to DmSGD

Thank you!

Kun Yuan homepage: https://kunyuan827.github.io/

BlueFog homepage: https://github.com/Bluefog-Lib/bluefog