Decentralized Optimization Algorithms for Large-Scale Deep Neural Network Training

Kun Yuan

DAMO Academy, Alibaba Group

Joint work with Yiming Chen, Pan Pan, Yinghui Xu, Wotao Yin (Alibaba), Bicheng Ying (UCLA), Hanbin Hu (UCSB), Xinmeng Huang (Upenn) , and Sulaiman A. Alghunaim (Kuwait University)

Aug 5, 2021, Zhejiang University

Contents in the lecture

Introduction to deep neural network (DNN) and various training modes (Part I)

- Single-node training
- Parallel/distributed training
- Decentralized training

Making decentralized algorithms practical for large-scale deep training (Part II)

- Exponential graphs
- Primal-dual decentralized methods
- Multiple gossip loops/Periodic global averaging

Other advanced topics and BlueFog (Part III)

- Large-batch deep training
- An open source decentralized deep training framework: BlueFog

Making decentralized methods practical: review

Which topology shall we use to organize all GPUs?

Topology	Per-iter. Comm.	Trans. Iters. (iid scenario)
Ring	$\Omega(2)$	$\Omega(n^7)$
Star	$\Omega(n)$	$\Omega(n^7)$
2D-Grid	$\Omega(4)$	$\Omega(n^5 \log_2^2(n))$
2D-Torus	$\Omega(4)$	$\Omega(n^5)$
$\frac{1}{2}$-RandGraph	$\Omega(\frac{n}{2})$	$\Omega(n^3)$
Static Exp.	$\tilde{\Omega}(1)$	$\tilde{\Omega}(n^3)$
One-peer Exp.	$\Omega(1)$	$\tilde{\Omega}(n^3)$

Making decentralized methods practical: review

How to accelerate D-SGD when non-iid data exists? Exact-Diffusion

non-iid scenario	Exact-Diffusion	D-SGD
strongly-convex	$\Omega(\frac{\rho^2 n}{1-\rho})$	$\Omega(\frac{\rho^2 n}{(1-\rho)^2})$
generally-convex	$\Omega(\frac{\rho^4 n^3}{(1-\rho)^2})$	$\Omega(\frac{\rho^4 n^3}{(1-\rho)^4})$
non-convex	N.A.	$\Omega(\frac{\rho^4 n^3}{(1-\rho)^4})$

Making decentralized methods practical: review

How to accelerate D-SGD over extremely sparse topology (i.e., $\rho \to 1$)?

Decentralized SGD with Periodic Global Averaging

scenario	DSGD-PGA	D-SGD
iid data	$\Omega(\rho^4 n^3 H^2)$	$\Omega(\frac{\rho^4 n^3}{(1-\rho)^2})$
non-iid data	$\Omega(\rho^4 n^3 H^4)$	$\Omega(\frac{\rho^4 n^3}{(1-\rho)^4})$

Part III: Other advanced topics and BlueFog

- Sec. 1 Large-batch deep training
- Sec. 2 An open source decentralized deep training framework: BlueFog

Advanced topics

- Decentralized DL with directed topology (Assran et al., 2019; Pu et al., 2020; Xin and Khan, 2018)
- Decentralized DL with time-varying topology (Koloskova et al., 2020; Nedic et al., 2017)
- Decentralized DL with severe data heterogeneity (Tang et al., 2018a; Lin et al., 2021; Xin et al., 2020; Lu et al., 2019)
- Decentralized DL with asynchrony and delays (Lian et al., 2018; Zhang and You, 2019; Wu et al., 2017)
- Decentralized DL with compression and quantization (Koloskova et al., 2019a,b; Tang et al., 2018b; Liu et al., 2020; Kovalev et al., 2021)

Unfortunately we cannot cover these topics in this lecture.

But let's discuss an important topic that is easy to be ignored:

Decentralized large-batch deep training

Motivation

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

$$\text{(8 GPUs:)} \quad 256 \times 8 = 2K \quad \text{(samples)}$$
$$\text{(64 Gpus:)} \quad 256 \times 64 = 16K \quad \text{(samples)}$$

- Large-batch training is unavoidable when more nodes participate in

Decentralized momentum SGD

- Recall the distributed optimization problem

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

- Recall the D-SGD algorithm

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

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

- The **momentum accelerated** D-SGD is more popular in real deep learning

Decentralized momentum SGD

Algorithm 1: DmSGD

Require: Initialize γ, $x_i^{(0)}$; let $m_i^{(0)} = 0, \beta \in (0, 1)$

for $k = 0, 1, 2, ..., T-1$, *every node* i **do**

Sample $\xi_i^{(k)}$ and update $g_i^{(k)} = \nabla F(x_i^{(k)}; \xi_i^{(k)})$

$m_i^{(k+1)} = \beta m_i^{(k)} + g_i^{(k)}$ ▷ momentum update

$x_i^{(k+\frac{1}{2})} = x_i^{(k)} - \gamma m_i^{(k+1)}$ ▷ local model update

$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i} w_{ij} x_j^{(k+\frac{1}{2})}$ ▷ partial average

---

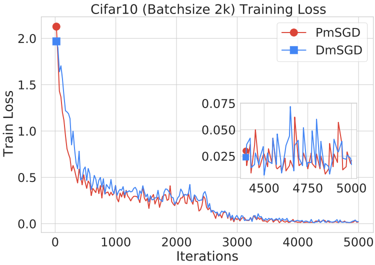
- The above DmSGD method is widely used in decentralized deep training[1]
- Reduces to D-SGD when $\beta = 0$

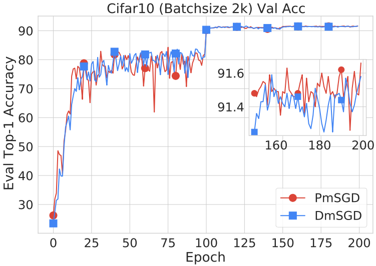
[1] [Lian et.al., 2018; Assran et.al., 2019; Gao and Huang, 2020]

Large-batch DmSGD has poor performance

Experimental setting: CIFAR-10; ResNet-20

Small-batch: 2K batch-size per iteration



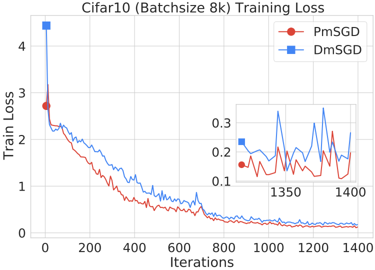


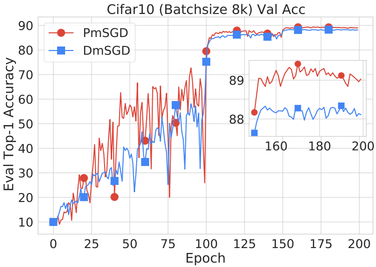
DmSGD and PmSGD have almost the **same** performance with small-batch.

Large-batch DmSGD has poor performance

Experimental setting: CIFAR-10; ResNet-20

Large-batch setting: 8K batch-size per iteration





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

Why does DmSGD have severe performance degradation than PmSGD?

Limiting bias of DmSGD

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

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

- stochastic bias is cuased by the gradient noise
- inconsistency bias is caused by the data heterogeneity (different D_i)
- As batch-size increases, sto. bias will vanish and incost. bias will dominate

Proposition

The inconsistency bias dominates the convergence of large-batch *DmSGD.*

Limiting bias of DmSGD

For example, the limiting bias of DSGD (s.c. cost) is (Yuan et al., 2020):

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

- where $b^2 = \frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(x^\star)\|^2$ denotes data heterogeneity
- when each D_i is identical, it holds that $f_i(x) = f_j(x)$ for any i and j, which implies that $\nabla f_i(x^\star) = 0$ and hence $b^2 = 0$
- in other words, $b^2 = 0$ for when each D_i is identical (i.i.d. scenario)
- $\sigma^2 \to 0$ as batch-size goes large, and hence inconsist. bias dominates

DmSGD incurs severe inconsistency bias

We rewrite full-batch DmSGD recursion as follows:

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

- No stochastic bias in the above recursion (full-batch gradient)
- momentum will not vanish as $x_i^{(k)} \neq \sum_{j \in \mathcal{N}_i} w_{ij} x_j^{(k-1)}$ as $k \to \infty$;
- momentum will incur additional inconsistency bias.

DmSGD incurs severe inconsistency bias

Proposition (Yuan et al. (2021))

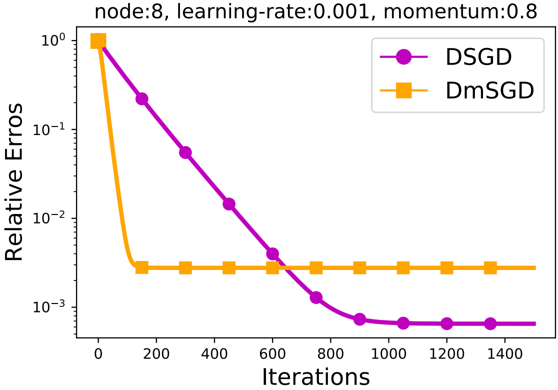
The full-batch DmSGD (S.C. cost) has the following inconsistency bias:

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

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 D-SGD has limiting bias $O(\gamma^2 b^2/(1-\rho)^2)$
- The momentum in DmSGD amplifies the inconsistency bias as $\beta\in(0,1)$
- DmSGD suffers from significant inconsist. bias when $\beta\to 1$
- Such amplified inconsist. bias results in notable performance degradation in large-batch scenario

DmSGD incurs severe inconsistency bias: verification



- A numerical verification: full-batch linear regression
- DmSGD is faster but suffers from more inconsistency bias (as expected)

Remove the momentum-incurred bias

We modify the full-batch DmSGD a little bit (Yuan et al., 2021)[2]:

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

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

- $x_i^{(k)} - x_i^{(k-1)} \to 0$ as $k \to \infty$;
- momentum-incurred bias will vanish as $k \to \infty$;
- we name the above algorithm as full-bath DecentLaM

[2] K. Yuan, Y. Chen, X. Huang, Y. Zhang, P. Pan, Y. Xu, and W. Yin, "DecentLaM: Decentralized Stochastic Momentum SGD for Large-batch Deep Training", to appear in ICCV 2021

Another useful algorithm derivation

- We let $\mathbf{x} = [x_1, \cdots, x_n]^T \in \mathbb{R}^{n\times d}$ and $f(\mathbf{x}) = \sum_{i=1}^{n} f_i(x_i)$
- We assume W is positive-definite and doubly stochastic, and $f_i(x)$ is s.c.
- We introduce $\mathbf{s} = W^{-\frac{1}{2}}\mathbf{x}$ and hence $\mathbf{x} = W^{\frac{1}{2}}\mathbf{s}$
- The full-batch DSGD algorithm can be rewritten as

$$
\begin{aligned}
& \mathbf{x}^{(k+1)} = W(\mathbf{x}^{(k)} - \gamma \nabla f(\mathbf{x}^{(k)})) \\
\Longleftrightarrow \quad & \mathbf{s}^{(k+1)} = W\mathbf{s}^{(k)} - \gamma W^{\frac{1}{2}} \nabla f(W^{\frac{1}{2}}\mathbf{s}^{(k)}) \\
& \qquad\quad = W\mathbf{s}^{(k)} - \gamma \nabla_{\mathbf{s}} f(W^{\frac{1}{2}}\mathbf{s}^{(k)}) \\
& \qquad\quad = \mathbf{s}^{(k)} - \gamma \underbrace{\left(\nabla_{\mathbf{s}} f(W^{\frac{1}{2}}\mathbf{s}^{(k)}) - \frac{1}{\gamma}(I - W)\mathbf{s}^{(k)}\right)}_{\text{gradient}}
\end{aligned}
$$

Another useful algorithm derivation

- We conclude that DSGD is essentially a standard GD for problem

$$\min_{\mathbf{s}} \quad f(W^{\frac{1}{2}}\mathbf{s}^{(k)}) + \frac{1}{2\gamma}\|\mathbf{s}\|_{I-W}^2$$

- When $\mathbf{s}^\star$ is achieved, we can derive $\mathbf{x}^\star = W^{\frac{1}{2}}\mathbf{s}^\star$

- Interpret DSGD as GD is critical; many techniques used in GD (such as momentum acceleration) can also be integrated to DSGD

- Add momentum to DSGD is equivalent to add momentum to GD:

$$\begin{aligned}
\mathbf{g}_{\mathbf{s}}^{(k)} &= \nabla_{\mathbf{s}} f(W^{\frac{1}{2}}\mathbf{s}^{(k)}) - \frac{1}{\gamma}(I-W)\mathbf{s}^{(k)} \\
\mathbf{m}_{\mathbf{s}}^{(k+1)} &= \beta \mathbf{m}_{\mathbf{s}}^{(k)} + \mathbf{g}_{\mathbf{s}}^{(k)} \\
\mathbf{s}^{(k+1)} &= \mathbf{s}^{(k)} - \gamma \mathbf{m}_{\mathbf{s}}^{(k+1)} \\
\mathbf{x}^{(k+1)} &= W^{\frac{1}{2}}\mathbf{s}^{(k+1)}
\end{aligned}$$

Another useful algorithm derivation

- Simplify the above recursions, we achieve

$$\begin{aligned}
\mathbf{g}^{(k)} &= \nabla f(\mathbf{x}^{(k)}) - \frac{1}{\gamma}(I - W)\mathbf{x}^{(k)} \\
\mathbf{m}^{(k+1)} &= \beta \mathbf{m}^{(k)} + \mathbf{g}^{(k)} \\
\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} - \gamma \mathbf{m}^{(k+1)}
\end{aligned}$$

- Combining all recursions, we achieve

$$\mathbf{x}^{(k+1)} = W(\mathbf{x}^{(k)} - \nabla f(\mathbf{x}^{(k)})) + \beta(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)})$$

DecentLaM algorithm with stochastic gradient

Algorithm 2: DecentLaM

Require: Initialize γ, $x_i^{(0)}$; let $m_i^{(0)} = 0, \beta \in (0,1)$

for $k = 0, 1, 2, ..., T-1$, *every node* i **do**

  Sample $\xi_i^{(k)}$ and update $g_i^{(k)}$ according to (10)

  $m_i^{(k+1)} = \beta m_i^{(k)} + g_i^{(k)}$ ▷ momentum update

  $x_i^{(k+1)} = x_i^{(k)} - \gamma m_i^{(k+1)}$ ▷ local model update

where $g_i^{(k)}$ is computed as follows:

$$g_i^{(k)} = \frac{1}{\gamma} x_i^{(k)} - \frac{1}{\gamma} \sum_{j \in \mathcal{N}_i} w_{ij} \left(x_j^{(k)} - \gamma \nabla F(x_j^{(k)}; \xi_j^{(k)}) \right)$$

Remove the momentum-incurred bias

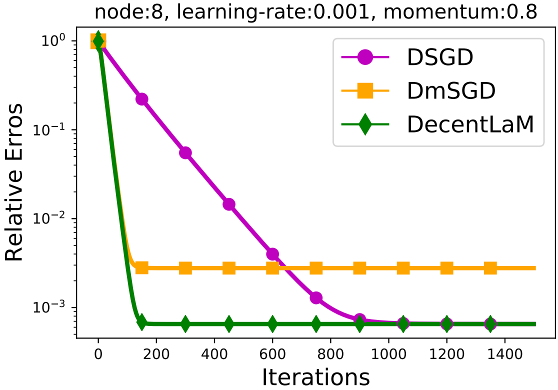
Proposition (Yuan et al. (2021))

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

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

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

Remove the momentum-incurred bias: verification

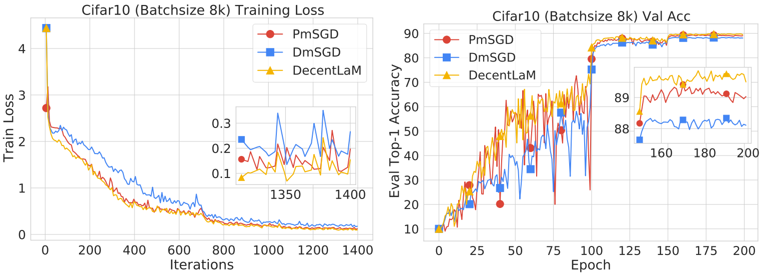


- A numerical verification: full-batch linear regression
- DecentLaM is as fast as DmSGD, and as accurate as DSGD

Go back to the large-batch Cifar-10 Experiment

Experimental setting: CIFAR-10; ResNet-20

Large-batch setting: 8K batch-size per iteration



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

Conjecture: For large-batch scenario in which the gradient noise is small, inconsistency bias can help the algorithm to escape the saddle point

Formal convergence theory of DecentLaM

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: $\frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(x) - \nabla f(x)\|^2 \le \hat{b}^2$

Theorem

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

$$\frac{1}{T}\sum_{k=0}^{T-1}\mathbb{E}\|\frac{1}{n}\sum_{i=1}^{n}\nabla f_i(\bar{x}^{(k)})\|^2$$
$$=O\Big(\underbrace{\frac{1-\beta}{\gamma T}}_{\text{convg. rate}} + \underbrace{\frac{\gamma\sigma^2}{n(1-\beta)} + \frac{\gamma^2\sigma^2}{1-\rho}}_{\text{sto. bias}} + \underbrace{\frac{\gamma^2\hat{b}^2}{(1-\rho)^2}}_{\text{inconsist.bias}}\Big)$$

Formal convergence theory of DecentLaM

- With decaying γ, DecentLaM will converge at rate $O(1/\sqrt{nT})$
- The inconsistency bias of DecentLaM is independent of momentum
- We also establish the convergence rate of DecentLaM with strongly convex cost, see (Yuan et al., 2021)
- W is not necessarily positive-definite in experiments; but it has to be symmetric

Inconsistency bias comparison between various methods

	Strongly-convex	**Non-convex**
DmSGD[3]	N.A.	$O\left(\frac{\gamma^2 M^2}{(1-\beta)^2}\right)$
DmSGD[4]	$O\left(\frac{\gamma^{5/2} M^2}{(1-\beta)^6}\right)$	$O\left(\frac{\gamma^2 M^2}{(1-\beta)^4}\right)$
DmSGD[5]	$O\left(\frac{\gamma^2 b^2}{(1-\beta)^2}\right)$	N.A
DA-DmSGD[6]	N.A.	$O\left(\frac{\gamma^2 \hat{b}^2}{(1-\beta)^2}\right)$
AWC-DmSGD[7]	$O\left(\frac{\gamma^2 M^2}{(1-\beta)^2}\right)$	$O\left(\frac{\gamma^2 M^2}{(1-\beta)^4}\right)$
SlowMo[8]	N.A	N.A
QG-DmSGD[9]	N.A	$O(\gamma^2 \hat{b}^2)$
DecentLaM (Ours)	$\boldsymbol{O(\gamma^2 b^2)}$	$\boldsymbol{O(\gamma^2 \hat{b}^2)}$

[3] (Gao and Huang, 2020)
[4] (Singh et al., 2020)
[5] Derived in this work
[6] (Yu et al., 2019)
[7] (Balu et al., 2020)
[8] (Wang et al., 2019)
[9] (Lin et al., 2021), a concurrent work

Comparison with decentralized primal-dual methods

	Strongly-convex	Non-convex
D2/E-D[10]	0	0
Gradient Tracking[11]	0	0
DecentLaM (Ours)	$O(\gamma^2 b^2)$	$O(\gamma^2 \hat{b}^2)$

- Theoretically, primal-dual methods can completely remove inconsistency bias, which is better than DecentLaM
- Empirically, they are worse than primal methods in validation accuracy
- Conjecture I: no effective acceleration exists for P.-D.
- Conjecture II: some inconsistency bias is beneficial for generalization
- It is still an open question to make decentralized P.-D. useful in DL

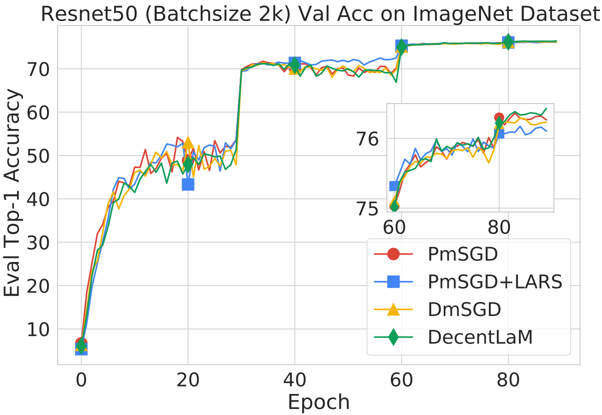
[10] (Yuan et al., 2019; Li et al., 2019; Tang et al., 2018a; Yuan et al., 2020)
[11] (Xu et al., 2015; Di Lorenzo and Scutari, 2016; Nedic et al., 2017; Qu and Li, 2018)

Experiments in Deep Training: Image classification

Image Classification:

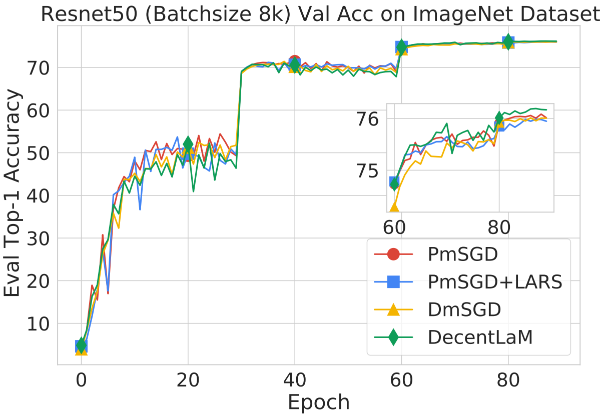
- Model: ResNet-50 ($\sim$25.5M parameters)
- Dataset: ImageNet-1K (1000 classes)
- Size: 1,281,167 training images and 50,000 validation images
- Hardware: 8 GPU $\times$ 8 machines
- We will test the proposed algorithm with batch-size 2K, 8K, 16K, and 32K
- Batchsize $\geq$ 8K is regarded as large batch-size
- Baselines: PmSGD, PmSGD $+$ LARS (layer-wise learning rate), DmSGD

Experiments with batchsize 2K (test accuracy)



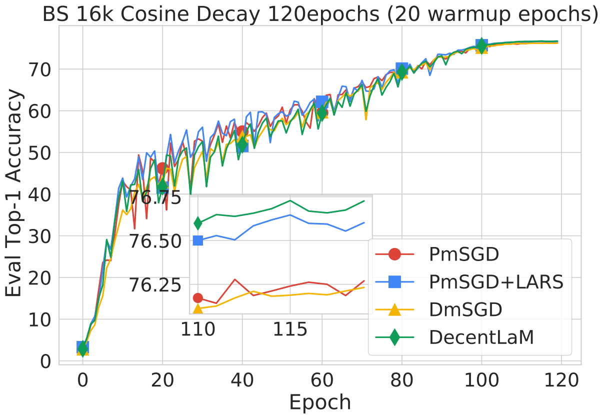
- DecentLaM has similar performance to DmSGD (sto. bias dominates)
- Decentralized methods are no worse than PmSGD

Experiments with batchsize 8K (test accuracy)



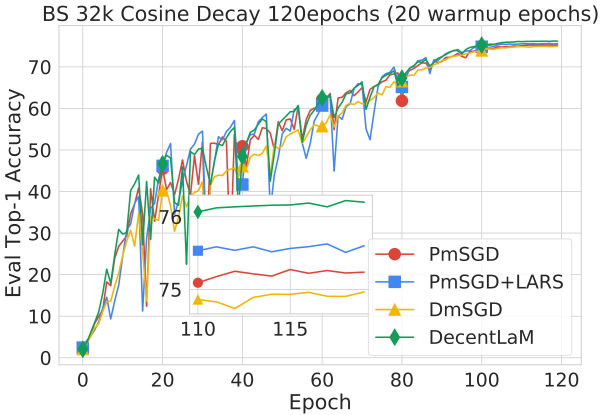
- DecentLaM outperforms DmSGD marginally (sto. bias diminishes)
- DecentLaM also outperforms PmSGD

Experiments with batchsize 16K (test accuracy)



- DecentLaM outperforms DmSGD significantly (incosist. bias diminishes)
- DecentLaM outperforms PmSGD significantly; even better than LARS

Experiments with batchsize 32K (test accuracy)



- DecentLaM outperforms DmSGD significantly (incosist. bias diminishes)
- DecentLaM outperforms PmSGD significantly; even better than 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

Table: Top-1 validation accuracy when training ResNet-50 with different batch sizes.

- DecentLaM can outperform PmSGD (esp. with large-batch)
- D^2-DmSGD is worse than QG-DmSGD and DecentLaM[12]

[12] Similar result was also reported in [Lin et.al., 2021]

Comparison across different DL models

method	ResNet-18	ResNet-34	ResNet-50	MobileNet-v2	EfficientNet
PmSGD	68.3	72.9	76.3	69.5	78.1
DmSGD	68.7	72.4	76.2	72.1	77.5
DecentLaM	70.5	73.4	76.7	72.2	78.3

Table: Top-1 validation accuracy when training ImageNet with $16K$ batchsize.

- DecentLaM outperforms DmSGD with large-batch (as expected)
- DecentLaM also outperforms PmSGD with better generalization error; a surprising result that cannot be explained by current optimization theory
- Conjecture: certain amount of inconsist. bias is beneficial

Running time saving

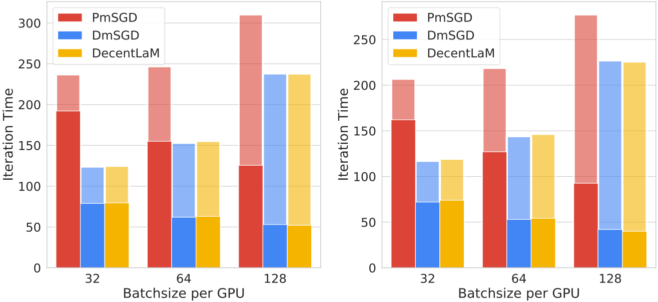


Figure: Runtime comparison on ResNet-50 with different batch sizes and network bandwidth (Left: 10Gbps; Right: 25Gbps). Each column indicates the averaged iteration runtime of 500 iterations. The thick part highlights the comm. overhead.

Experiments in object detection

Dataset	PASCAL VOC		COCO	
Model	R-Net	F-RCNN	R-Net	F-RCNN
PmSGD	79.0	80.3	36.2	36.5
PmSGD+LARS	78.5	79.8	35.7	36.2
DmSGD	79.1	80.5	36.1	36.4
DA-DmSGD	79.0	80.5	36.4	37.0
DecentLaM	79.3	80.7	36.6	37.1

Table: Comparision with different models on PASCAL VOC and COCO datasets. R-Net and F-RCNN refer to RetinaNet and Faster-RCNN respectively.

More results are available in the paper.

Summary

- DmSGD has significant accuracy degradation with large batch-size
- Momentum in DmSGD incurs significant inconsistency bias
- We propose DecentLaM to correct the momentum-incurred bias
- DecentLaM promises both fast and high-quality large-batch training

Part III: Other advanced topics and BlueFog

- Sec. 1 Large-batch deep training
- Sec. 2 An open source decentralized deep training framework: BlueFog

BlueFog: Making Decentralized Algorithms Practical for Optimizaiton and Deep Learning

A library available at `https://github.com/Bluefog-Lib/bluefog`

Aug 5, 2021, Zhejiang University

Main features

- BlueFog is open-source; supports parallel/decentralized methods
- Supports any dynamic and static network topology
- Supports efficient implementation of neighbor-allreduce (partial averaging)
- Suppose both CPU and GPU training through integration with PyTorch
- Wrap up torch optimizers; several codes to run decentralized deep training
- Detailed tutorials with Jupyter notebook on how to use it:

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

DNN example

BlueFog has a high-level API that wraps around any torch optimizer.

Example:

```
import torch
import bluefog.torch as bf
bf.init()
...
optimizer = optim.SGD(model.parameters(), lr=lr*bf.size())
optimizer = bf.DistributedNeighborAllreduceOptimzer( \
optimizer, model=model)
...
# Torch training code
```

BlueFog also provides optimizers: Distributed Allreduce, Distributed Hierarchical Neighbor Allreduce, etc.

SPMD (single program, multiple data)

One code for all nodes; different nodes have different data and unique ranks.

```
# hello_world.py
import bluefog.torch as bf
bf.init()
print("I am rank {} in size {}".format(bf.rank(), bf.size()))
```

```
> bfrun -np 2 python hello_world.py
```

```
I am rank 1 in size 2
I am rank 0 in size 2
```

Partial averaging

Example: compute the average of ranks of the nodes

```
import torch
import bluefog.torch as bf
bf.init()

x = torch.Tensor([bf.rank()])

for _ in range(100):
  x = bf.neighbor_allreduce(x)
  print("rank {} has x={}".format(bf.rank(), x))
```

Defaults:

- `bf.init()` creates a static exponential graph
- neighbor-averaging weights are set to $\frac{1}{\text{neighbors}+1}$ for every incoming neighbors and the node itself

```
> bfrun -np 10 python neighbor_avg.py

rank 0 has x=tensor([4.5000])
rank 3 has x=tensor([4.5000])
rank 9 has x=tensor([4.5000])
rank 1 has x=tensor([4.5000])
rank 7 has x=tensor([4.5000])
rank 4 has x=tensor([4.5000])
rank 2 has x=tensor([4.5000])
rank 6 has x=tensor([4.5000])
rank 5 has x=tensor([4.5000])
rank 6 has x=tensor([4.5000])
```

Partial averaging using dynamic subgraphs

Example: Default one-peer exponential averaging

```
dynamic_neighbors = topology_util.GetDynamicSendRecvRanks(
bf.load_topology(), bf.rank())

for _ in range(maxite):
  to_neighbors, from_neighbors = next(dynamic_neighbors)

  avg_weight = 1/(len(from_neighbors) + 1)

  xi = bf.neighbor_allreduce(xi, name='x',
  self_weight=avg_weight,
  neighbor_weights={r: avg_weight for r in from_neighbors},
  send_neighbors=to_neighbors)
```

You can replace `GetDynamicSendRecvRanks()` with your own.

Decentralized gradient descent (Nedic and Ozdaglar, 2009)

To approximate solve

$$\underset{\mathbf{x}}{\text{minimize}} \;\; \alpha \sum_{i=1}^{n} f_i(x_i) \qquad \text{subject to } x_1 = \cdots = x_n,$$

we can apply *decentralized gradient descent*:

$$\mathbf{x}^{k+1} = W\mathbf{x}^k - \alpha \nabla f(\mathbf{x}^k).$$

Implementation using static exp2:

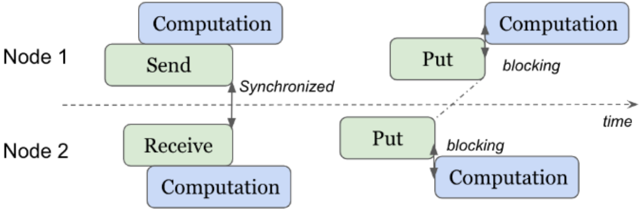
```
# DGD recursion
for k in range(maxite):
  xi = bf.neighbor_allreduce(xi) - alpha*ComputeGrad(fi,xi)
```

Blocking and asynchrony

Each node has two threads: communication thread and computation thread

- **non-blocking:** allow concurrent threads to save time
- blocking: computation starts after communication completes

Synchronization is similar concept but applies to operations across different nodes. All collective communications are synchronous.



Left: nonblocking but synchronized; **Right:** blocking, may or may not sync'd

By default, BlueFog is blocking and synchronized, but it also supports non-blocking and asynchronous operations

To save time, we ask neighbor allreduce $W\mathbf{x}^k$ not to block computation $\nabla f(\mathbf{x}^k)$, so they can run concurrently.

```
1 for k in range(maxite):
2   handle = bf.neighbor_allreduce_nonblocking(xi)
3   gradi = ComputeGrad(fi, xi)
4   avg_x = bf.wait(handle)
5   xi = avg_x - alpha*gradi
```

Since Line 5 must wait for the result of $W\mathbf{x}^k$.

EXTRA (Shi et al., 2015)

EXTRA was the first method that solves

$$\underset{x}{\text{minimize}} \sum_{i=1}^{n} f_i(x_i) \qquad \text{subject to } x_1 = \cdots = x_n$$

with a constant α. One form of this method is

$$\begin{cases} \mathbf{x}^1 = W\mathbf{x}^0 - \alpha \nabla f(\mathbf{x}^0), \\ \mathbf{x}^{k+1} = W(2\mathbf{x}^k - \mathbf{x}^{k-1}) - \alpha(\nabla f(\mathbf{x}^k) - \nabla f(\mathbf{x}^{k-1})), \quad k = 1, 2, \cdots \end{cases}$$

The code structure is similar to DGD. Non-blocking communication can accelerate the code.

Gradient-Tracking

DIGing Nedic et al. (2017) is a tracking-based method. For static W, DIGing is a special case of EXTRA. However, DIGing works for dynamic W.

$$\begin{cases} \mathbf{x}^{k+1} = W^{(k)}\mathbf{x}^k - \alpha \mathbf{y}^k \\ \mathbf{y}^{k+1} = W^{(k)}\mathbf{y}^k + \nabla f(\mathbf{x}^{k+1}) - \nabla f(\mathbf{x}^k) \end{cases}$$

$(\mathbf{y}^k)_k$ a tracking sequence converging to $\lim_k \frac{1}{n}\sum_{i=1}^n \nabla f_i(\mathbf{x}^k)$ if it exists.

```
xi = np.zeros((d,1))
yi = fi_grad_prev = ComputeGrad(fi, xi)
for k in range(maxite):
  self_weight, recv_weights = ComputeWeights(k, bf.rank())
  xi = bf.neighbor_allreduce(xi, self_weight, recv_weights) \
  - alpha*yi
  gi = ComputeGrad(fi, xi)
  yi = bf.neighbor_allreduce(gi, self_weight, recv_weights) \
  + gi - gi_prev
  gi_prev = gi.copy()
```

Summary

- Decentralized computing can accelerate large-scale deep training
- Exponential graphs are provably efficient for decentralized deep training
- Periodic global averaging can further accelerate decentralized deep training
- We develop a GitHub ropo to help implement decentralized training easily

References I

M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic gradient push for distributed deep learning," in *International Conference on Machine Learning (ICML)*, 2019, pp. 344–353.

S. Pu, W. Shi, J. Xu, and A. Nedić, "Push–pull gradient methods for distributed optimization in networks," *IEEE Transactions on Automatic Control*, vol. 66, no. 1, pp. 1–16, 2020.

R. Xin and U. A. Khan, "A linear algorithm for optimization over directed graphs with geometric convergence," *IEEE Control Systems Letters*, vol. 2, no. 3, pp. 315–320, 2018.

A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. U. Stich, "A unified theory of decentralized sgd with changing topology and local updates," in *International Conference on Machine Learning (ICML)*, 2020, pp. 1–12.

A. Nedic, A. Olshevsky, and W. Shi, "Achieving geometric convergence for distributed optimization over time-varying graphs," *SIAM Journal on Optimization*, vol. 27, no. 4, pp. 2597–2633, 2017.

References II

H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, “d^2: Decentralized training over decentralized data,” in *International Conference on Machine Learning*, 2018, pp. 4848–4856.

T. Lin, S. P. Karimireddy, S. U. Stich, and M. Jaggi, “Quasi-global momentum: Accelerating decentralized deep learning on heterogeneous data,” *arXiv preprint arXiv:2102.04761*, 2021.

R. Xin, U. A. Khan, and S. Kar, “An improved convergence analysis for decentralized online stochastic non-convex optimization,” *arXiv preprint arXiv:2008.04195*, 2020.

S. Lu, X. Zhang, H. Sun, and M. Hong, “Gnsd: A gradient-tracking based nonconvex stochastic algorithm for decentralized optimization,” in *2019 IEEE Data Science Workshop (DSW)*. IEEE, 2019, pp. 315–321.

X. Lian, W. Zhang, C. Zhang, and J. Liu, “Asynchronous decentralized parallel stochastic gradient descent,” in *International Conference on Machine Learning*, 2018, pp. 3043–3052.

References III

J. Zhang and K. You, "Asyspa: An exact asynchronous algorithm for convex optimization over digraphs," *IEEE Transactions on Automatic Control*, vol. 65, no. 6, pp. 2494–2509, 2019.

T. Wu, K. Yuan, Q. Ling, W. Yin, and A. H. Sayed, "Decentralized consensus optimization with asynchrony and delays," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 2, pp. 293–307, 2017.

A. Koloskova, S. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *International Conference on Machine Learning*, 2019, pp. 3478–3487.

A. Koloskova, T. Lin, S. U. Stich, and M. Jaggi, "Decentralized deep learning with arbitrary communication compression," in *International Conference on Learning Representations*, 2019.

H. Tang, S. Gan, C. Zhang, T. Zhang, and J. Liu, "Communication compression for decentralized training," *arXiv preprint arXiv:1803.06443*, 2018.

References IV

X. Liu, Y. Li, R. Wang, J. Tang, and M. Yan, "Linear convergent decentralized optimization with compression," in *International Conference on Learning Representations*, 2020.

D. Kovalev, A. Koloskova, M. Jaggi, P. Richtarik, and S. Stich, "A linearly convergent algorithm for decentralized optimization: Sending less bits for free!" in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 4087–4095.

K. Yuan, S. A. Alghunaim, B. Ying, and A. H. Sayed, "On the influence of bias-correction on distributed stochastic optimization," *IEEE Transactions on Signal Processing*, 2020.

K. Yuan, Y. Chen, X. Huang, Y. Zhang, P. Pan, Y. Xu, and W. Yin, "Decentlam: Decentralized momentum sgd for large-batch deep training," *arXiv preprint arXiv:2104.11981*, 2021.

H. Gao and H. Huang, "Periodic stochastic gradient descent with momentum for decentralized training," *arXiv preprint arXiv:2008.10435*, 2020.

References V

N. Singh, D. Data, J. George, and S. Diggavi, "Squarm-sgd: Communication-efficient momentum sgd for decentralized optimization," *arXiv preprint arXiv:2005.07041*, 2020.

H. Yu, R. Jin, and S. Yang, "On the linear speedup analysis of communication efficient momentum sgd for distributed non-convex optimization," in *International Conference on Machine Learning*. PMLR, 2019, pp. 7184–7193.

A. Balu, Z. Jiang, S. Y. Tan, C. Hedge, Y. M. Lee, and S. Sarkar, "Decentralized deep learning using momentum-accelerated consensus," *arXiv preprint arXiv:2010.11166*, 2020.

J. Wang, V. Tantia, N. Ballas, and M. Rabbat, "Slowmo: Improving communication-efficient distributed sgd with slow momentum," *arXiv preprint arXiv:1910.00643*, 2019.

K. Yuan, B. Ying, X. Zhao, and A. H. Sayed, "Exact dffusion for distributed optimization and learning – Part I: Algorithm development," *IEEE Transactions on Signal Processing*, vol. 67, no. 3, pp. 708 – 723, 2019.

References VI

Z. Li, W. Shi, and M. Yan, "A decentralized proximal-gradient method with network independent step-sizes and separated convergence rates," *IEEE Transactions on Signal Processing*, July 2019, early acces. Also available on arXiv:1704.07807.

J. Xu, S. Zhu, Y. C. Soh, and L. Xie, "Augmented distributed gradient methods for multi-agent optimization under uncoordinated constant stepsizes," in *IEEE Conference on Decision and Control (CDC)*, Osaka, Japan, 2015, pp. 2055–2060.

P. Di Lorenzo and G. Scutari, "Next: In-network nonconvex optimization," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 2, no. 2, pp. 120–136, 2016.

G. Qu and N. Li, "Harnessing smoothness to accelerate distributed optimization," *IEEE Transactions on Control of Network Systems*, vol. 5, no. 3, pp. 1245–1260, 2018.

A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

References VII

W. Shi, Q. Ling, G. Wu, and W. Yin, "EXTRA: An exact first-order algorithm for decentralized consensus optimization," *SIAM Journal on Optimization*, vol. 25, no. 2, pp. 944–966, 2015.