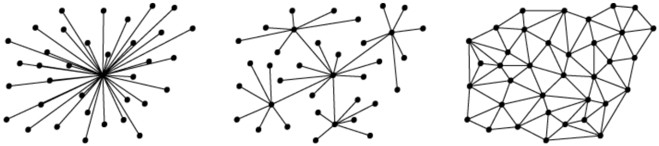
Part 2: Decentralized optimization

Decentralized optimization

Consider an undirected, connected graph $G = (V, E)$. Each node $i \in E$ has an objective function f_i.



If node i uses only its local and adjacent information to compute, we call it *local operation*.

Decentralized optimization uses local operations to achieve a global optimization objective.

Why decentralized?

- The nature of certain multi-agent systems
- Avoid long-distance communication. Reduce contention.
- Reliability, privacy considerations

Decentralized ADMM

Consider: n agents minimize their joint sum:

$$\text{minimize} \sum_{i=1}^{n} f_i(x)$$

With y_e for each undirected edge $e = \{i, j\} \in E$ we obtain the ADMM-ready form:

$$\begin{array}{ll} \underset{\substack{\{x_i\}_{i\in V} \\ \{y_e\}_{e\in E}}}{\text{minimize}} & \displaystyle\sum_{i\in V} f_i(x_i) \\ \text{subject to} & \begin{cases} x_i - y_e = 0 \\ x_j - y_e = 0 \end{cases} \qquad \forall\ e = \{i, j\} \in E. \end{array}$$

We can simplify the ADMM into

$$x_i^{k+1} = \mathbf{prox}_{(\alpha|N_i|)^{-1} f_i}(v_i^k) \qquad i \in V$$

$$\begin{cases} a_i^{k+1} = \frac{1}{|N_i|} \sum_{j \in N_i} x_j^{k+1} \\ v_i^{k+1} = v_i^k + a_i^{k+1} - \frac{1}{2} a_i^k - \frac{1}{2} x_i^k \end{cases} \qquad i \in V.$$

It uses only local computation and local communication (neighborhood allreduce).

We call this method *Decentralized ADMM*.

A broader class of methods based on local mixing

Average consensus

Goal: compute the average of decentralized vectors $a_1, \dots, a_n \in \mathbb{R}^d$.

Let x_i^k be the kth iterate of node i. Set $x_i^0 = a_i$. Common approach:

$$x_i^{k+1} = w_{ii} x_i^k + \sum_{j \in N_i} w_{ij} x_j^k, \quad i = 1, \dots, n.$$

Local operations imply: $w_{ij} \neq 0$ only for $i = j$ and $(i,j) \in E$.

Using

$$\mathbf{x} = \begin{bmatrix} - \; x_1^T \; - \\ \vdots \\ - \; x_n^T \; - \end{bmatrix} \in \mathbb{R}^{n \times d},$$

we get

$$\mathbf{x}^k = W \mathbf{x}^{k-1} = \cdots = W^k \mathbf{x}^0.$$

To obtain

$$\lim_{k\to\infty} W^k \mathbf{x}^0 = \begin{bmatrix} \vdots \\ \frac{1}{n}\sum_{i=1}^n \mathbf{x}_i^0 \\ \vdots \end{bmatrix} = \frac{1}{n}\mathbf{1}\mathbf{1}^T\mathbf{x}^0.$$

for any $\mathbf{x}^0$, we deduct

- $W\mathbf{1} = \mathbf{1}$
- $\mathbf{1}^T = \mathbf{1}^T W$
- $\lambda_{\max}(W) = 1$ and $1 > \lambda_2(W) \geq \cdots \geq \lambda_n(W) > -1$.

Consensus minimization

Average consensus

$$\underset{\mathbf{x}}{\text{minimize}} \ \sum_{i=1}^{n} \frac{1}{2}\|x_i - a_i\|^2, \quad \text{subject to } x_i = x_j, \ \ \forall \text{ nodes } i, j \in V.$$

Generalization to minimization

$$\underset{\mathbf{x}}{\text{minimize}} \ \sum_{i=1}^{n} f_i(x_i), \quad \text{subject to } x_i = x_j, \ \ \forall \text{ nodes } i, j \in V.$$

Penalty formulation

Recall property of W:

$$W\mathbf{x} = \mathbf{x} \quad \Leftrightarrow \quad (I - W)\mathbf{x} = 0 \quad \Leftrightarrow \quad x_1 = \cdots = x_n.$$

A penalty (inexact) formulation:

$$\underset{\mathbf{x}}{\text{minimize}} \sum_{i=1}^{n} h_i(x_i) + \frac{1}{2\rho}\|\mathbf{x}\|_{I-W}^2. \tag{1}$$

Applying gradient descent to (2) yields DGD[1]

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

We call it adaptation-with-combination or AWC-DGD.

[1] Nedic-Ozdaglar'09, also related to Cattivelli-Lopes-Sayed'07

Another penalty formulation

Assume $\min\{\lambda_2(W), \ldots, \lambda_n(W)\} > 0$, i.e., $W \succ 0$. Consider

$$\underset{\mathbf{x}}{\text{minimize}} \sum_{i=1}^{n} h_i(x_i) + \frac{1}{2\rho}\|\mathbf{x}\|^2_{W^{-1}-I}. \tag{2}$$

Apply variable-metric gradient descent

$$\begin{aligned}\mathbf{x}^{k+1} &= \mathbf{x}^k - \alpha W(\nabla h(\mathbf{x}^k) - \frac{1}{\alpha}(W^{-1} - I)\mathbf{x}^k) \\ &= W\left(\mathbf{x}^k - \alpha \nabla h(\mathbf{x}^k)\right)\end{aligned}$$

We call it adaptation-then-combination or ATC-DGD.

Constrained formulation

Consider

$$\underset{\mathbf{x}}{\text{minimize}} \sum_{i=1}^{n} \left(h_i(x_i) + g_i(x_i)\right), \quad \text{subject to } (I - W)\mathbf{x} = 0.$$

Write

$$h(\mathbf{x}) = \sum_{i=1}^{n} h_i(x_i) \quad \text{and} \quad g(\mathbf{x}) = \sum_{i=1}^{n} g_i(x_i).$$

Both $h(\mathbf{x})$ and $g(\mathbf{x})$ are separable. But, $(I - W)\mathbf{x} = 0$ is not.

How to derive methods based on multiplying by W?

Operator splitting

Operator splitting

1. Formulate a problem into the form

$$0 \in \varkappa(x) + (x) \quad \text{or} \quad 0 \in \varkappa(x) + (x) + (x)$$

   where $\varkappa, , : \mathbb{R}^n \to \mathbb{R}^n$ are (point- or set-valued) operators

2. Apply a splitting scheme to get $\mathbb{Q}$ such that
   - $z^* = \mathbb{Q}z^*$ recovers a solution x^*
   - computing $\mathbb{Q}$ is easy (by evaluating $\varkappa, ,$ separately)
   - $z^{k+1} \leftarrow \mathbb{Q}z^k$ converges

Forward and backward operators

	Forward op.	Backward op. (Resolvent)
definition	$(I - \gamma\varkappa)$	$J_{\gamma\varkappa} = (I + \gamma\varkappa)^{-1}$
analogy	forward Euler	backward Euler
example:	**grad descent** $\varkappa = \nabla f$ of cvx $f \in C^1$ yields $(I - \gamma\nabla f)$;	**proximal mapping** $\varkappa = \partial f$ of cvx f yields $\mathbf{prox}_{\gamma f}$;
example:	**skew-symm A**	**projection to cvx set**

Basic operator splitting schemes

$$\boxed{0 \in \varkappa x + x}$$

- **forward-backward splitting (FBS)** (Mercier'79) for
  (maximally monotone) $+$ (cocoercive)[2]

$$\mathbb{Q} = J_{\gamma\varkappa}(I - \gamma)$$

- **Douglas-Rachford splitting (DRS)** (Lion-Mercier'79) for
  (maximally monotone) $+$ (maximally monotone)

$$\mathbb{Q} = \frac{1}{2}I + \frac{1}{2}(2J_{\gamma\varkappa} - I)(2J_{\gamma} - I)$$

They generalize the **proximal point method** (PPM).

[2] $\varkappa$ is μ-*strongly monotone* or μ-*coercive* if $\langle \varkappa x - \varkappa y, x - y\rangle \geq \mu\|x - y\|^2$.
$\varkappa$ is β-*coercive* if $\langle \varkappa x - \varkappa y, x - y\rangle \geq \beta\|\varkappa x - \varkappa y\|^2$.

$$\boxed{0 \in \varkappa x + x + x}$$

- **three-operator splitting (DYS)** (Davis-Yin'15) for
  (maximally monotone) + (maximally monotone) + (cocoercive):

$$\mathbb{Q} = I - J_{\gamma\varkappa} + J_{\gamma}(2J_{\gamma\varkappa} - I + \gamma J_{\gamma\varkappa})$$

where $J_{\varkappa} := (I + \varkappa)^{-1}$.

DYS generalizes FBS and DRS.

EXTRA Method

EXTRA

EXTRA[3] is the first method that uses a fixed stepsize and converges linearly if $\sum_{i=1}^{n} f(x)$ is strongly convex.

Iteration:

$$\mathbf{x}^{k+1} \leftarrow W\mathbf{x}^k - \alpha \nabla f(\mathbf{x}^k) - \sum_{j=0}^{k-1} \frac{1}{2}(I - W)\mathbf{x}^j.$$

PG-EXTRA[4]

$$\mathbf{x}^{k+1} \leftarrow \underbrace{(I + \alpha \partial g)^{-1}}_{\mathbf{prox}_{\alpha g}} \left(W\mathbf{x}^k - \alpha \nabla f(\mathbf{x}^k) - \sum_{j=0}^{k-1} \frac{1}{2}(I - W)\mathbf{x}^j \right).$$

We can derive them from operator splitting.

[3] Shi-Ling-Wu-Yin'15

[4] Shi-Ling-Wu-Yin'15b

Saddle-point problem and splitting

Let

$$U^T U = (I - W).$$

So $U\mathbf{x} = 0$ iff $(I - W)\mathbf{x} = 0$.

With Lagrangian

$$L(\mathbf{x}; \mathbf{y}) = f(\mathbf{x}) + g(\mathbf{x}) + \mathbf{y}^T U \mathbf{x},$$

ignoring regularity conditions, we get

$$\begin{array}{ll} \text{minimize}_{\mathbf{x}} & f(\mathbf{x}) + g(\mathbf{x}) \\ \text{subject to} & (I - W)\mathbf{x} = 0. \end{array} \quad \Leftrightarrow \quad \begin{bmatrix} 0 \\ 0 \end{bmatrix} \in \begin{bmatrix} \partial_{\mathbf{x}} L(\mathbf{x}; \mathbf{y}) \\ \partial_{\mathbf{y}}(-L(\mathbf{x}; \mathbf{y})) \end{bmatrix},$$

which expands to

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \in \Bigg(\underbrace{\begin{bmatrix} \nabla f & 0 \\ 0 & 0 \end{bmatrix}} + \underbrace{\begin{bmatrix} \partial g & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & U^T \\ -U & 0 \end{bmatrix}}_{\varkappa} \Bigg) \underbrace{\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}}_{\mathbf{z}},$$

Forward-backward splitting (FBS)

Let matrix M be symmetric, positive definite.

From

$$\begin{aligned} 0 \in \varkappa(\mathbf{z}) + (\mathbf{z}) \;&\Leftrightarrow\; M\mathbf{z} - (\mathbf{z}) \in M\mathbf{z} + \varkappa(\mathbf{z}) \\ &\Leftrightarrow\; (I - M^{-1})\mathbf{z} \in (I + M^{-1}\varkappa)\mathbf{z} \\ &\Leftrightarrow\; \underbrace{(I + M^{-1}\varkappa)^{-1}}_{J_{M,\varkappa}} \underbrace{(I - M^{-1})}_{F_{M,}} \mathbf{z} = \mathbf{z} \end{aligned}$$

we derive FBS under metric M

$$\mathbf{z}^{k+1} = J_{M,\varkappa} F_{M,}(\mathbf{z}^k) \Leftrightarrow\; M\mathbf{z}^k - (\mathbf{z}^k) \in M\mathbf{z}^{k+1} + \varkappa(\mathbf{z}^{k+1})$$

Select

$$M = \begin{bmatrix} \frac{1}{\alpha} I & -U^T \\ -U & \frac{1}{\beta} I \end{bmatrix},$$

the right-hand side, $M\mathbf{z}^{k+1} + \varkappa(\mathbf{z}^{k+1})$, becomes block lower-triangular, so we can first compute $\mathbf{x}^{k+1}$, then $\mathbf{y}^{k+1}$.

Using $\mathbf{w}^k = U^T \mathbf{y}^k$ and $\beta = 1/(2\alpha)$, we obtain PG-EXTRA.

Network-independent stepsize

In EXTRA, parameter α is related to M, thus U, thus W, and thus the graph topology. α also depends on Lipschitz constants of ∇f_i.

Use a new metric

$$M = \begin{bmatrix} \frac{1}{\alpha}I & 0 \\ 0 & \frac{1}{\beta}I - \alpha U^T U \end{bmatrix}.$$

Applying Gaussian elimination to the system $M\mathbf{z} - (\mathbf{z}) \in M\mathbf{z} + \varkappa(\mathbf{z})$ yields a lower-triangular system.

Set $\beta = 1/(2\alpha)$. We obtain decentralized method NIDS[5].

NIDS converges if $\alpha < \frac{2}{L}$, independent of the graph topology.

Easy to generalize to node-specific stepsizes $\alpha_i < \frac{2}{L_i}$, where L_i is Lipschitz constant of ∇f_i.

[5] Li-Shi-Yan'19, also related to Nedic-Olshevsky-Shi-Uribe'17, Qu-Li'18.

Lots of decentralized work not covered

Nesterov-like acceleration (Qu-Li'17, Scaman et al'18), double-loop-based acceleration (Uribe-Lee-Gasnikov-Nedic'18, Li-Fang-Yin-Lin'18, ...)

Gradient tracking (Zhu-Martinez'10, Xu et al'15, Scutari-Sun'19, ...)

ADMM linear convergence (Yuan-Ling-Yin'16), on time-varying graphs (Nedic-Olshevsky-Shi'17, ...)

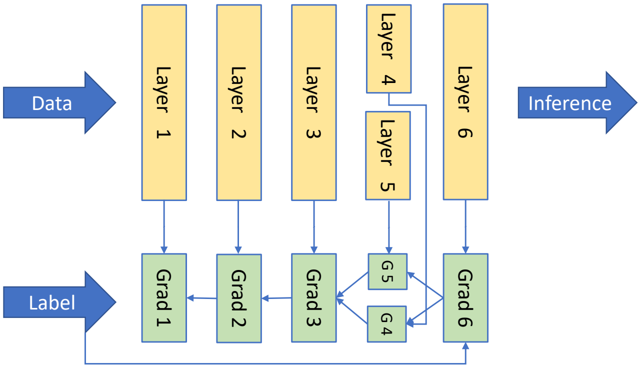
Left-stochastic W in (PushSum, Xi et al'18, Xin-Xi-Khan'19, Yuan-Ying-Zhao-Sayed'19, ...)

Asynchronous (Wu et al.18, ...)

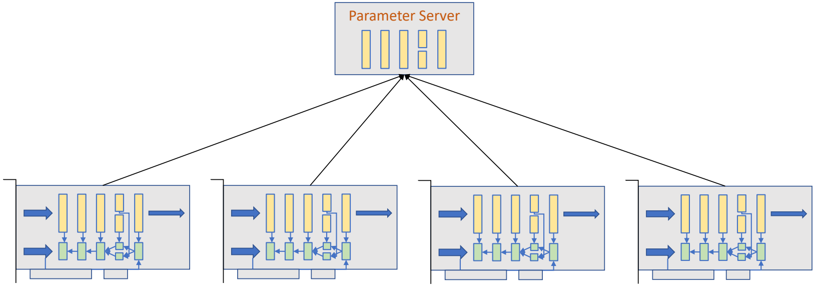
Incorporating SGD (Lian et al'17, Lian et al'18, ...)

Decentralized SGD for deep learning

DNN training



Parameter server approach [Li et.al. 2014]



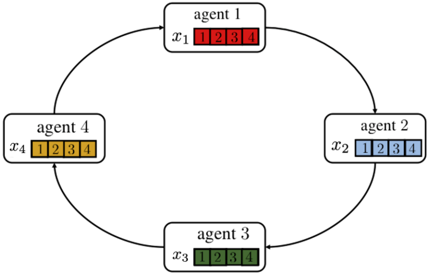
Pros: mature implementation (2015–), fault tolerance

Cons: many-to-one communication is no scalable

Ring Allreduce [Patarasuk and Yuan 2009]

Started by Distributed PaddlePaddle [Gibiansky 2017] (Baidu)

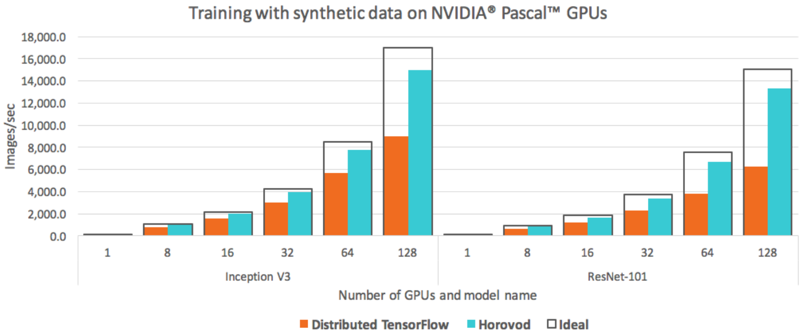
Popularized by Horovod [Sergeev and Del Balso 2018] (Linux Foundation AI)



Pros: mature implementation (2018–), bandwidth optimality

Cons: total latency grows linearly

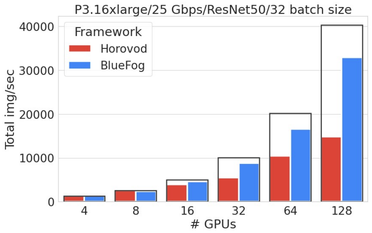
Distributed Tensorflow vs Horovod

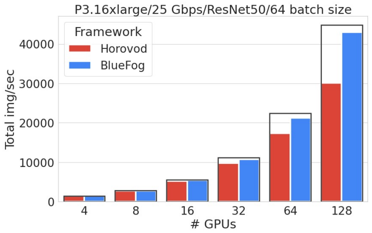


Result is from Horovod GitHub homepage.

BlueFog

- Communication framework for PyTorch
- Just a few lines of Python
- Supports MPI and NCCL
- Higher throughput than Hovovod





Fixed vs dynamic neighbor averaging

Fixed Neighbor-averaging SGD:

$$x_i^{k+1} = \sum_{j=1}^{n} W_{ij} \left(x_j^k - \alpha^k (\text{mini-batch grad at } j) \right).$$

Dynamic Neighbor-averaging SGD:

$$x_i^{k+1} = \sum_{j=1}^{n} W_{ij}^{(k)} \left(x_j^k - \alpha^k (\text{mini-batch grad at } j) \right).$$

Each round uses a different W.

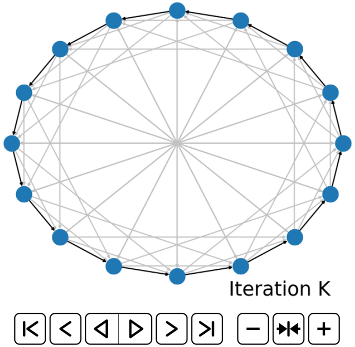
Further generalization:

1. if communication is faster, apply multiple W per mini-batch gradient
2. if communication is slower, apply multiple mini-batch gradients per W

For simplicity, assume one W per mini-batch gradient

Dynamic exp2-ring [Assran et.al. 2019]

Take $n = 16$ for example. Break a 16-node exp2-graph into four subgraphs. To each subgraph, assign a unique W with weights $1/2, 1/2$ for the active nodes.



In every subgraph, every node communicates one other node. Computing $W\mathbf{y}$ takes $O(1)$ time.

Exact averaging achieved by finite dynamic neighbor averaging

Theorem: When $n = 2^\tau$ for $\tau \in \mathbb{Z}$, dynamic exp-2 averaging satisfies

$$W^{(\tau)}W^{(\tau-1)} \cdots W^{(1)} = \frac{1}{n}\mathbf{1}\mathbf{1}^T$$

Furthermore, for any $p = 2, \ldots, \tau$,

$$W^{(p-1)} \cdots W^{(1)}W^{(\tau)} \cdots W^{(p)} = \frac{1}{n}\mathbf{1}\mathbf{1}^T.$$

This W-sequence is communication optimal among all averaging matrices.

Large-scale training for 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
- GPUs: 8×8

Method	Epochs/Hours to 76%.
Allreduce SGD	68 / 5.57
Neighbor-averaging SGD	76 / 4.23

Summary

Decentralized optimization is based on local communication like “gossiping”

Decentralized optimization relaxes strong consensus to weak consensus or multi-step strong consensus

Decentralized optimization is suitable where decentralization is natural or centralized communication is too expensive