北京大学
PEKING UNIVERSITY

# A Mathematics-Inspired Learning-to-Optimize Framework for Decentralized Optimization

**Kun Yuan (袁 坤)**

**Center for Machine Learning Research @ Peking University**

Oct. 21, 2024

# Joint work with

**Yutong He**
**(Peking U.)**

**Qiulin Shang**
**(Peking U.)**

**Xinmeng Huang**
**(UPenn)**

**Jialin Liu**
**(CFU)**

PART 01

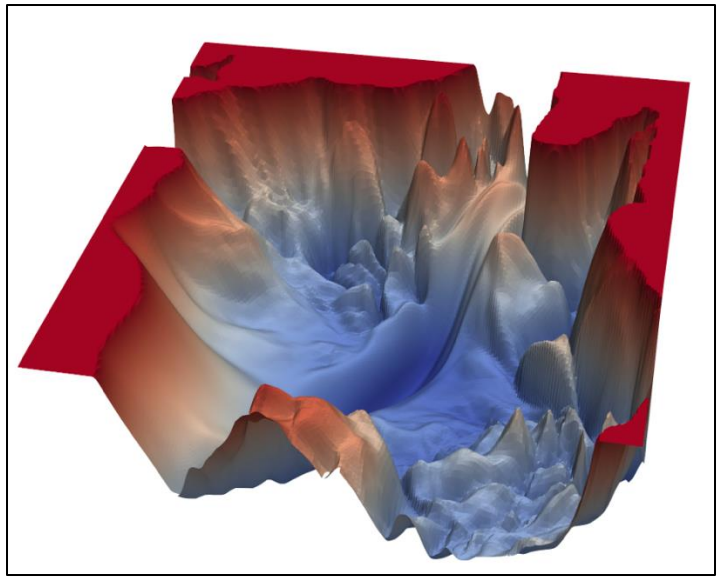
# Basics and Motivation

# DNN model is notoriously difficult to train

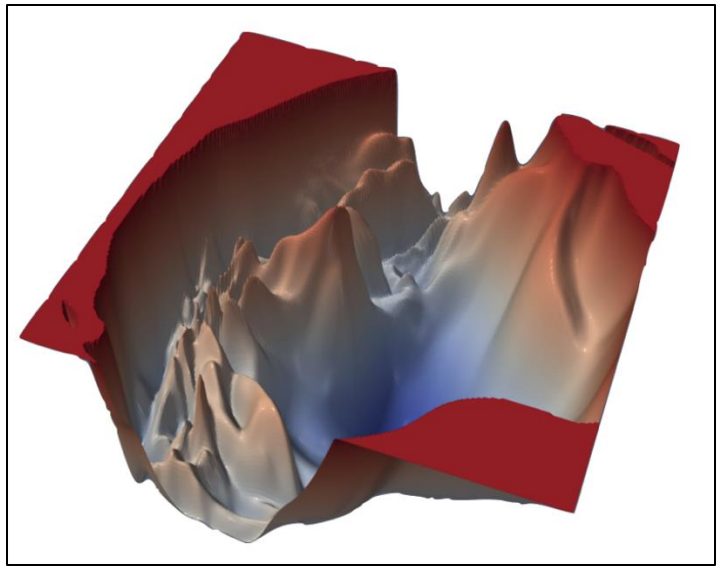
北京大学 PEKING UNIVERSITY

- Highly **non-convex** cost functions; cannot find global minima; trapped into local minimum

VGG-56



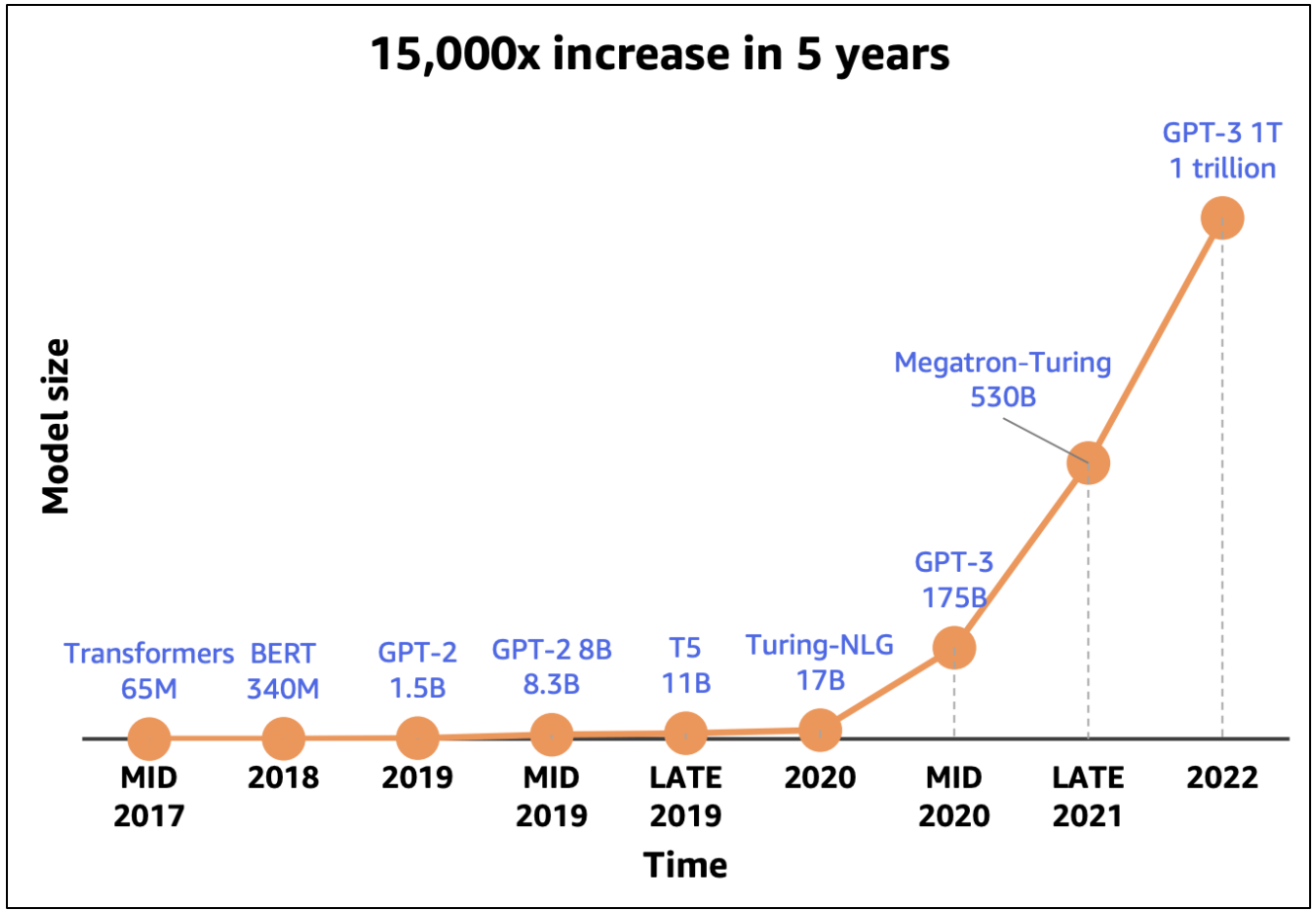
VGG-110



[Visualizing the loss landscape of neural nets]

# DNN model is notoriously difficult to train

- The model size is **large**, i.e., $x \in \mathbb{R}^d$ is of extremely high dimensions




[Train and deploy large language models on Amazon SageMaker]

# DNN model is notoriously difficult to train

北京大学 PEKING UNIVERSITY

- The size of the dataset is **huge**

| Dataset | Sampling prop. | Epochs | Disk size |
|---|---|---|---|
| CommonCrawl | 67.0% | 1.10 | 3.3 TB |
| C4 | 15.0% | 1.06 | 783 GB |
| Github | 4.5% | 0.64 | 328 GB |
| Wikipedia | 4.5% | 2.45 | 83 GB |
| Books | 4.5% | 2.23 | 85 GB |
| ArXiv | 2.5% | 1.06 | 92 GB |
| StackExchange | 2.0% | 1.03 | 78 GB |

Table 1: **Pre-training data.** Data mixtures used for pre-training, for each subset we list the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion.

Crawled data from websites; in both high quality and low quality

High-quality data

[LLaMA: Open and Efficient Foundation Language Models]

# DNN model is notoriously difficult to train

DNN training = Non-convex training + Huge dimensions + Huge dataset

**Efficient** and **scalable** distributed optimization approaches are in urgent need

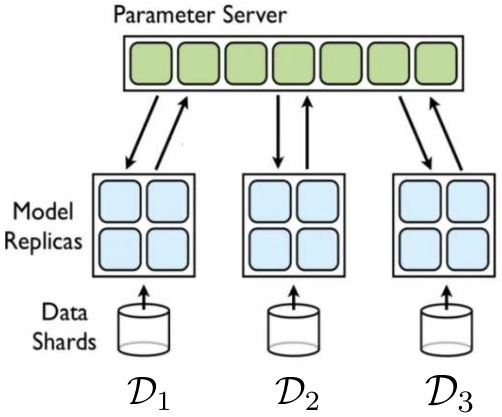
# Distributed learning

- This talk focuses on data-parallel approaches to process **massive** datasets;

- 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 $\xi_i$ 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)

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

**PSGD**

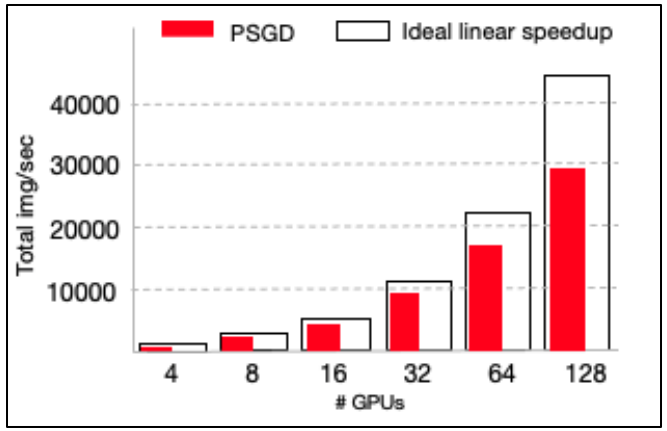
$$g_i^{(k)} = \nabla F(x^{(k)};\xi_i^{(k)}) \qquad \text{(Local compt.)}$$

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

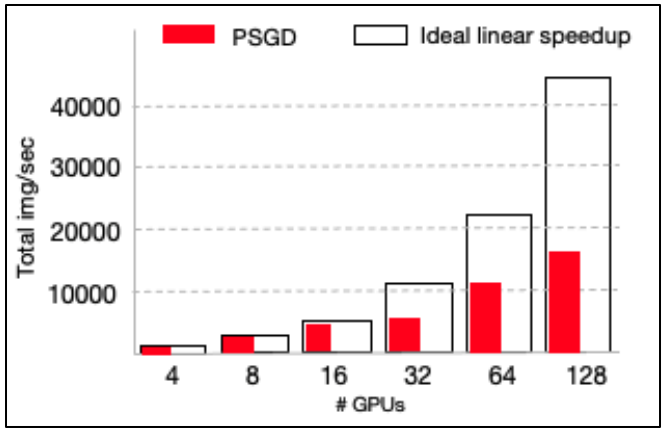
- Each node $i$ samples data $\xi_i^{(k)}$ and computes gradient $\nabla F(x^{(k)};\xi_i^{(k)})$
- All nodes synchronize (i.e. globally average) to update model $x$ per iteration
- The global average takes significant communication overhead (esp. when n is large)

# PSGD cannot achieve ideal scalability due to comm. overhead

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




Small comm.-to-compt. ratio




Large comm.-to-compt. ratio

Ring-Allreduce is used in each experiment

- How can we reduce the communication overhead in PSGD?

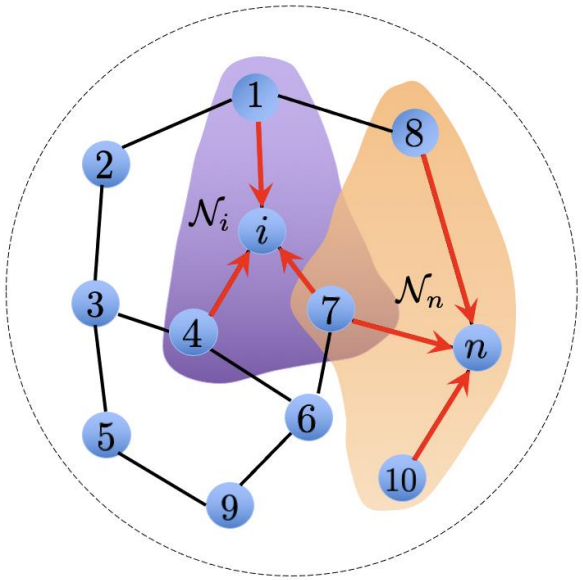
B. Ying, K. Yuan, H. Hu, Y. Chen and W. Yin, “BlueFog: Make decentralized algorithms practical for optimization and deep learning”, arXiv: 2111. 04287, 2021

# Decentralized SGD (DSGD)

- To reduce 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 [1, 2]
- $\mathcal{N}_i$ 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

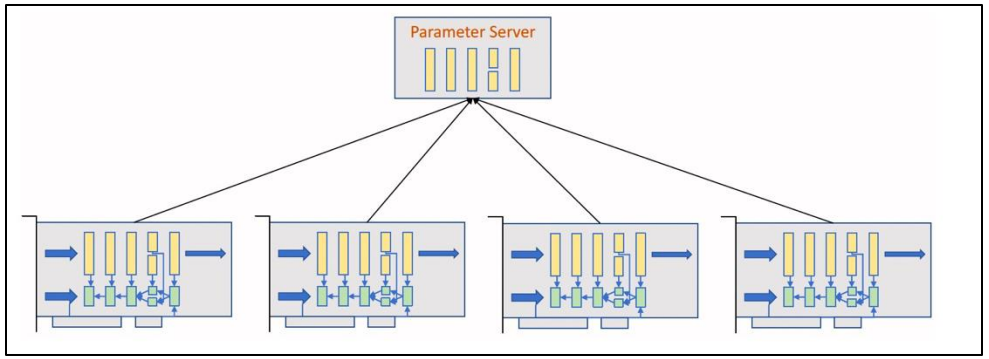
---


[1] C. G. Lopes and A. H. Sayed, "Diffusion least-mean-squares over adaptive networks", ICASSP, 2007

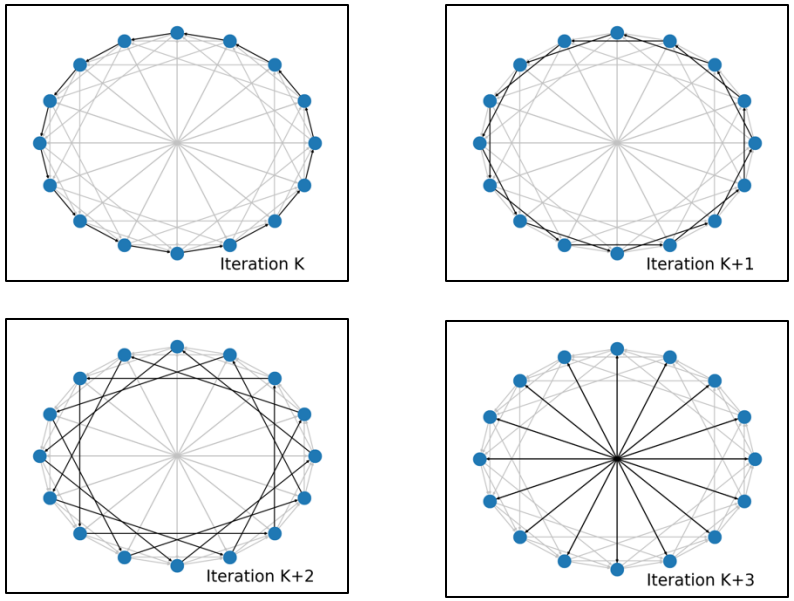
[2] A. Nedich and A. Ozdaglar, " Distributed subgradient methods for multi-agent optimization", IEEE TAC, 2009

# DSGD is more communication-efficient than PSGD

- Incurs $O(1)$ comm. overhead on **sparse** topologies; much less than global average $O(n)$



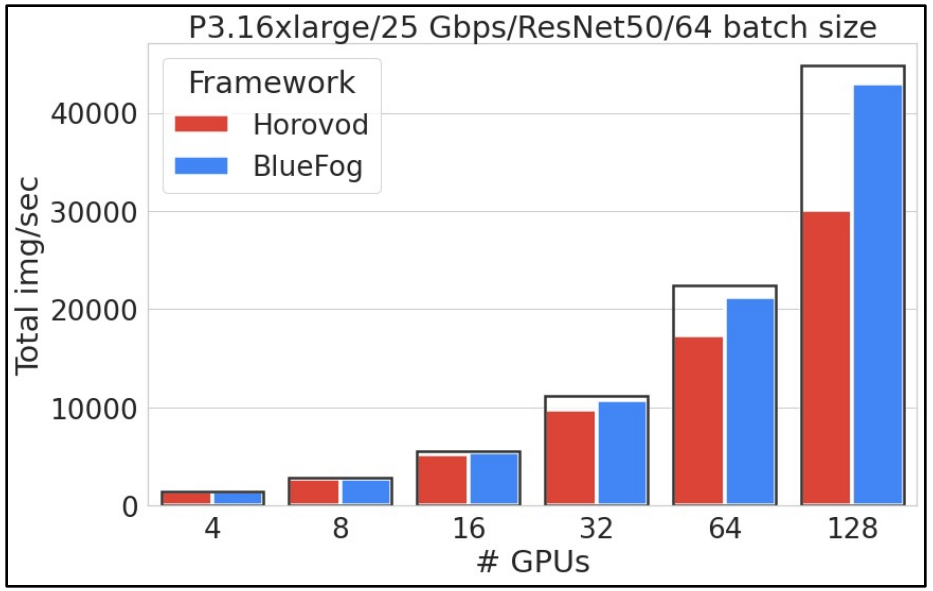

Global averaging has $O(n)$ comm. overhead



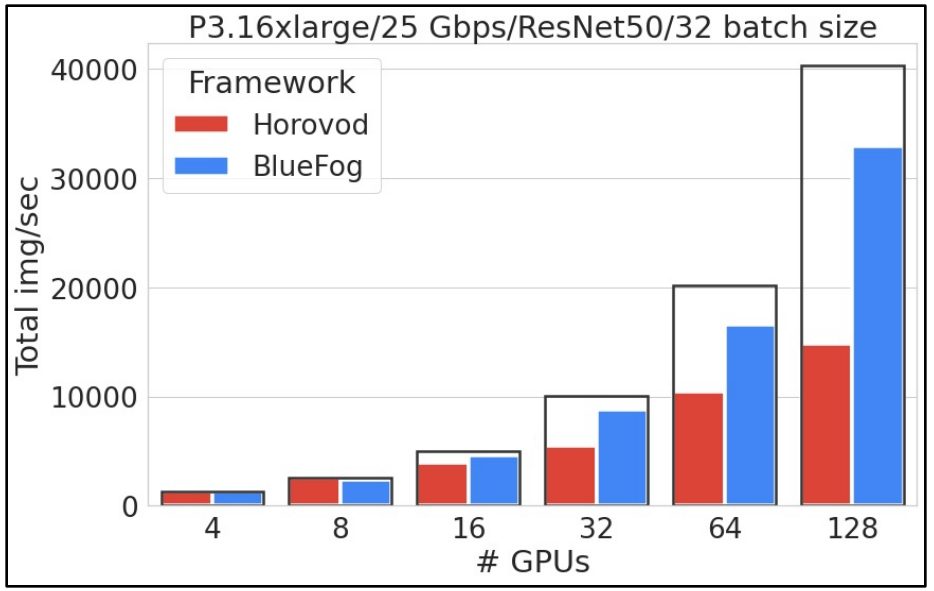

Partial averaging has $O(1)$ comm. overhead

# DSGD is more communication-efficient than PSGD

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



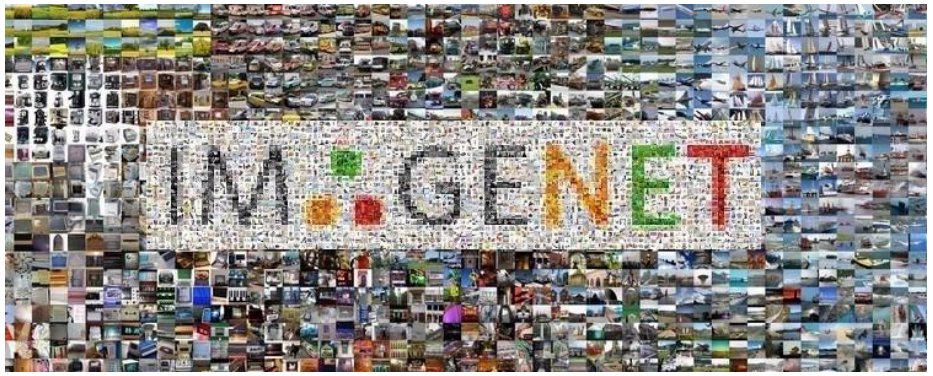

Small comm.-to-compt. ratio




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 saves more wall-clock time without severely hurting performance

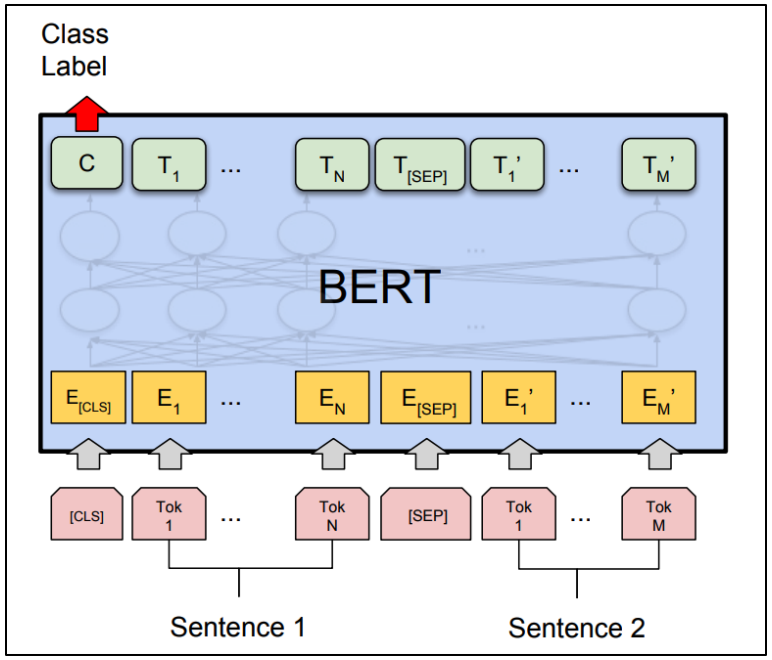



| nodes<br>topology | 4(4x8 GPUs)<br>acc. | <br>time | 8(8x8 GPUs)<br>acc. | <br>time | 16(16x8 GPUs)<br>acc. | <br>time | 32(32x8 GPUs)<br>acc. | <br>time |
|---|---|---|---|---|---|---|---|---|
| P-SGD | 76.32 | 11.6 | 76.47 | 6.3 | 76.46 | 3.7 | 76.25 | 2.2 |
| Decentralized | 76.34 | 11.1 | 76.52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

DSGD shows very impressive linear speedup performance and saves more time than PSGD!

B. Ying, K. Yuan, Y. Chen, H. Hu, and W. Yin, “Exponential Graph is Provably Efficient for Decentralized Deep Training”, NeurIPS 2021

# DSGD saves more wall-clock time without severely hurting performance




Model: BERT-Large (330M parameters)

Dataset: Wikipedia (2500M words) and BookCorpus (800M words)

Hardware: 64 GPUs

Table. Comparison in loss and training time [1]

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

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

# Many well-known decentralized algorithms have been proposed

北京大学 PEKING UNIVERSITY

- Since 2009, many well-known decentralized algorithms have been proposed in literature

Base optimizer

DGD | Diffusion | Dual averaging | D-ADMM | EXTRA

Exact Diffusion | Gradient Tracking | Network Newton | ESOM

Practical settings

Non-smooth setting | Non-convex setting | Directed network | Stochastic setting | Asynchronous setting

- The combination of base optimizer and practical settings gives rise to massive algorithms
- All these algorithms are **hand-crafted**

# Hand-crafted algorithms


Function space → Expert design → Hand-crafted optimizers

- ☺ Works well for a **wide range** of target problems
- ☹ Excessively focused on worst-case problems, overlooking the problem-specific structures
- ☹ Hyperparameters to **tune**
- ☹ Hand-crafted algorithms can be **sub-optimal**

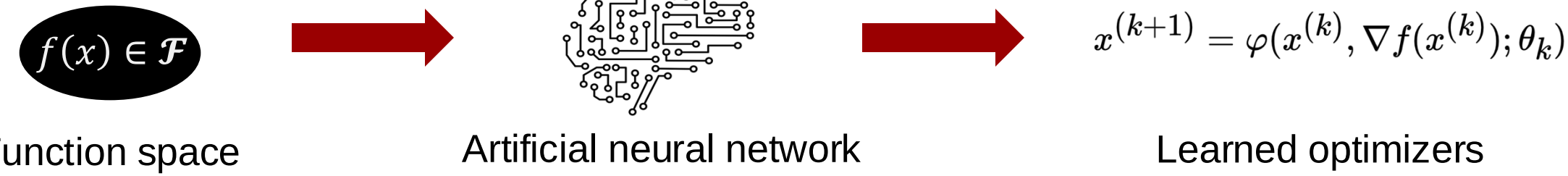
**Can we develop algorithms automatically?**

北京大学
PEKING UNIVERSITY

PART 02

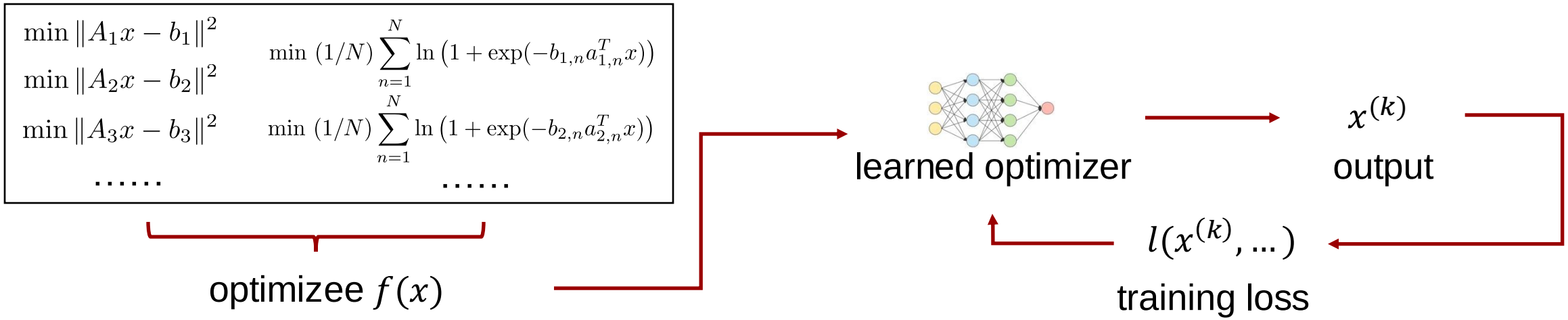
# Learn-to-Optimize

# Learning to optimize




- ☹ Works well on a **specialized** domain; not on all problems
- ☺ Trained on real problems, learned to utilize problem-specific structures
- ☺ **Free** from hyperparameter tuning
- ☺ **Faster** than hand-crafted optimizers

# How to learn an optimizer?




- Training data: target problem functions to optimize, called **optimizees**.
- Learned optimizer: neural network models that map $x^{k-1}$ to $x^k$
- Loss function: performance of the learned optimizer on training optimizees
- Two major approaches: **algorithm unrolling** and **generic L2O**

# Algorithm unrolling: principle

Rule-based loop
(from hand-crafted algorithms)

unrolling

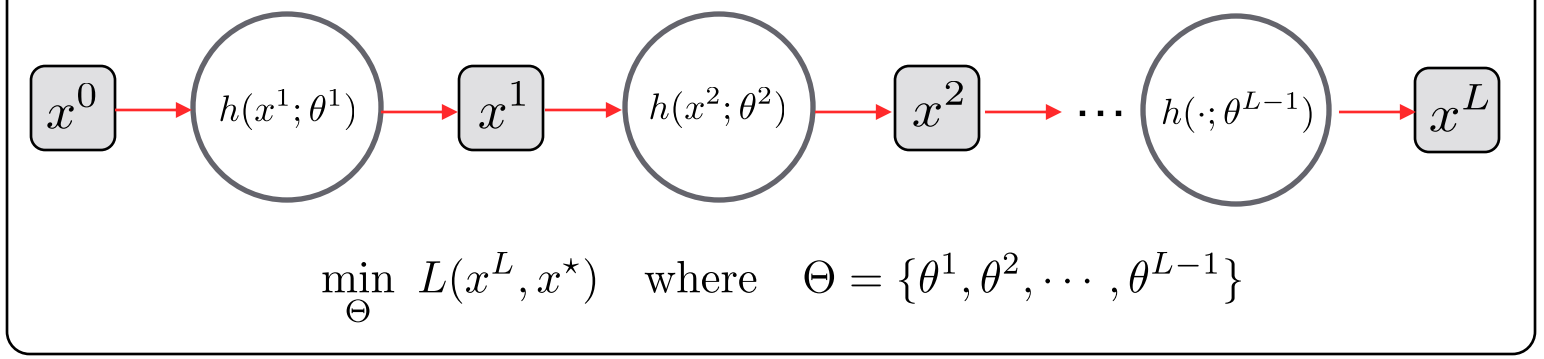
Learnable layers

$$x^{\ell+1} = h(x^{\ell})$$

parameterization

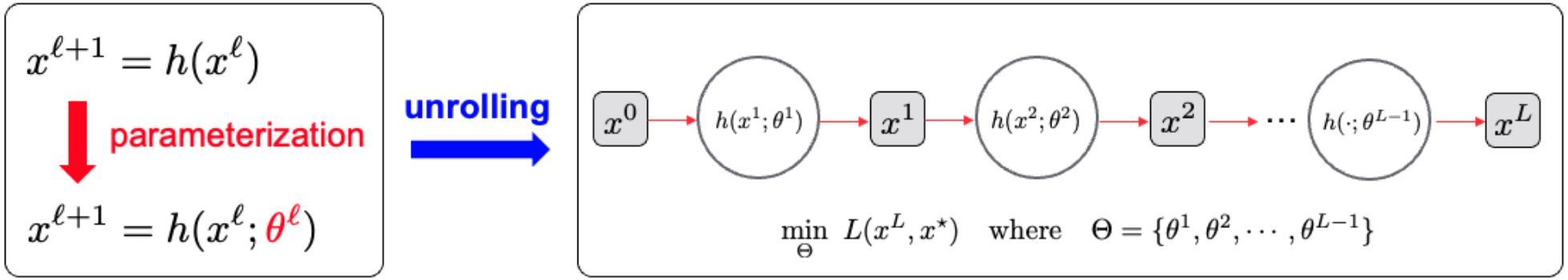
$$x^{\ell+1} = h(x^{\ell}; \theta^{\ell})$$

unrolling




K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In Proceedings of the 27th international conference on international conference on machine learning, pages 399–406, 2010.

# Algorithm unrolling: Pros and Cons




- ☺ Perform **well** on in-distribution tasks
- ☺ **Compact** search space; easy to train
- ☹ Based on **hand-crafted** algorithms; not learning new algorithms but only improve them
- ☹ **Hardly** generalize to longer iterations due to **storage issues**

# Generic L2O

- Generic L2O typically learns a recurrent neural network (RNN) as the iterative update rule without a hand-crafted base algorithm.

Specific rule: $\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t)$

**parameterization**

Abstract rule: $\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi)$

RNN parameterization

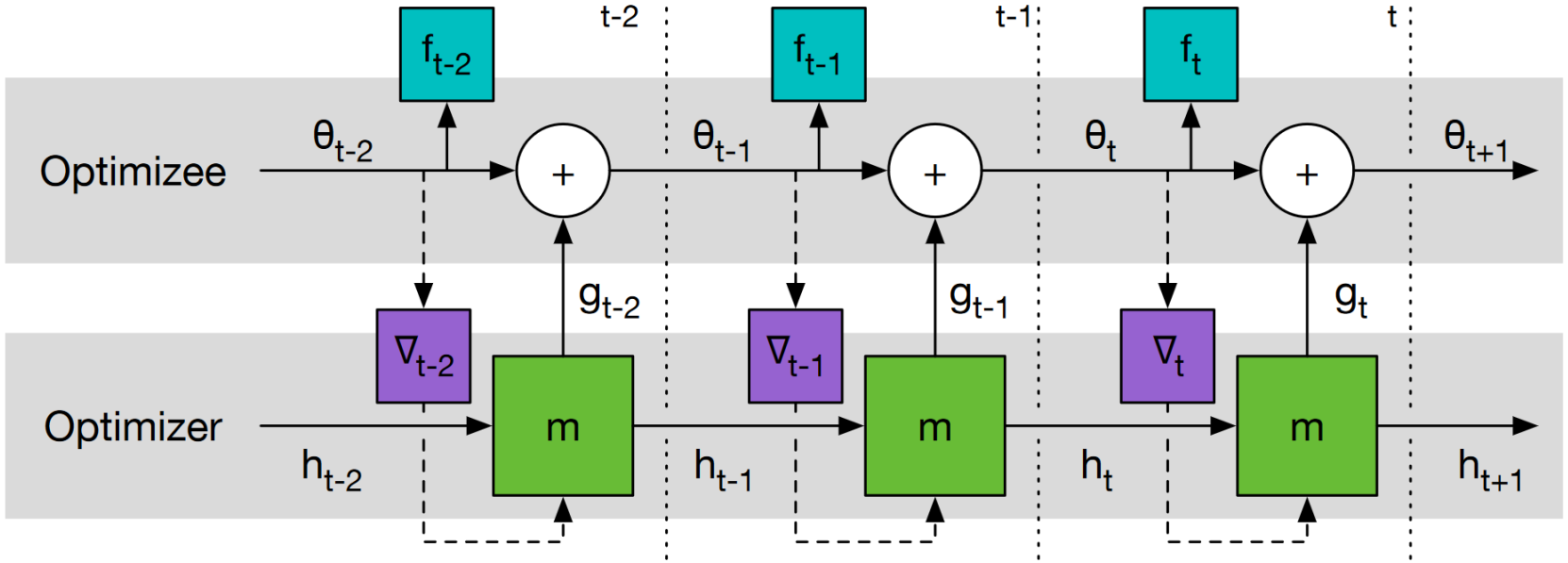
M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. Advances in neural information processing systems, 29, 2016.

# Learning to learn with recurrent neural networks

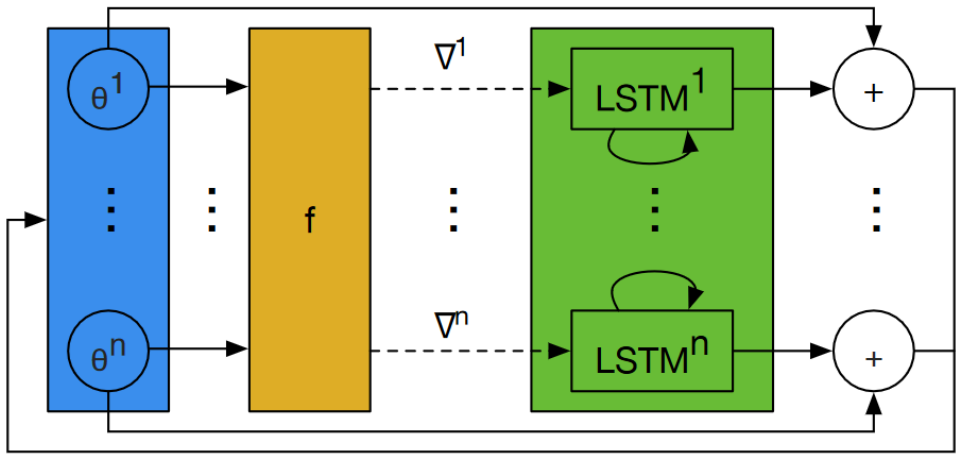
$$\theta_{t+1} = \theta_t + g_t \,,$$

$$\begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} = m(\nabla_t, h_t, \phi) \,.$$

Very general; do not specialize to any concrete algorithms such as momentum GD, adaptive GD, etc.



# Coordinate-wise LSTM optimizer




- LSTMs for different coordinates have **shared parameters** but **different hidden state**.
- Coordinate-wise LSTM optimizer has **fewer trainable parameters**, and is **more flexible** to problems with different dimensions.

☺ RNN shares the same parameters; **Memory efficient**; Generalize to **longer** iterations

☺ Do **not depend on** a hand-crafted algorithm

☹ **Vast search space**; difficult to learn a good update rule

☹ **Cannot handle constraints**

# A brief summary on existing L2O approaches

北京大学 PEKING UNIVERSITY

## Unrolling

- ☺ Perform **well** on in-distribution tasks
- ☺ **Compact** search space; easy to train
- ☹ Based on **hand-crafted** algorithms
- ☹ Memory **expensive**

## Generic L2O

- ☺ **Memory efficient**; fit for long iterations
- ☺ **No need** for hand-crafted algorithms
- ☹ **Cannot handle constraints**
- ☹ **Vast** search space; **difficult** to train

Both approaches show **weak generalization** on **out-of-distribution** problems!

# Our target

Can we develop an effective L2O algorithm for decentralized optimization that is

- **Memory efficient:** the learned algorithm can run arbitrarily large iterations
- **Easy to train:** the search space is compact
- **Enforcing consensus constraint:** ensure convergence to the consensus solution among nodes
- **Strong generalization:** can generalize well to out-of-distribution problems

北京大学
PEKING UNIVERSITY

PART 03

# A math-inspired L2O framework for decentralized optimization

# A trivial decentralized L2O framework

Consider a distributed optimization problem:

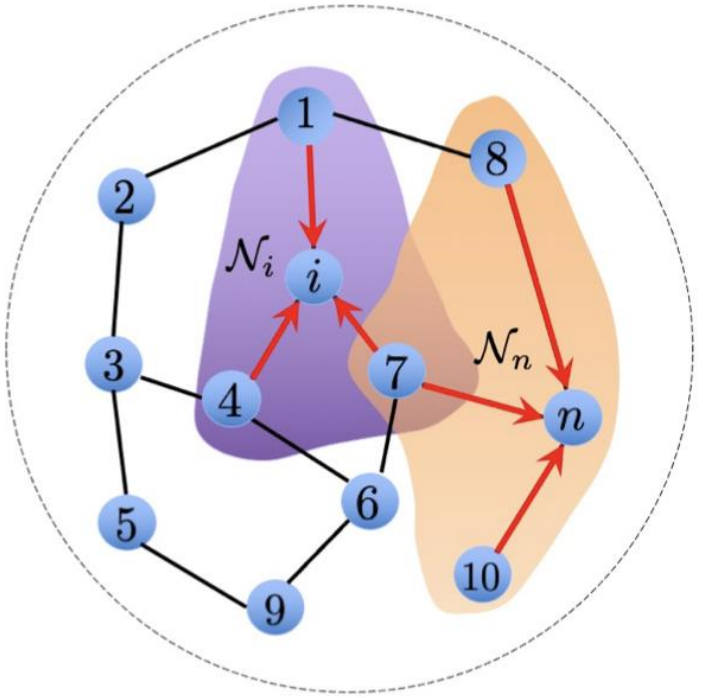
$$\min_{x\in\mathbb{R}^d} f(x)=\frac{1}{n}\sum_{i=1}^{n} f_i(x) \quad \text{where} \quad f_i(x)=\mathbb{E}_{\xi_i\sim D_i}F(x;\xi_i)$$

Recall the generic L2O approach:

$$x^{k+1}=x^k+g^k\left(\nabla f\left(x^k\right);\theta\right), \quad k=0,1,\cdots,K-1$$

If we directly follow the generic L2O approach, we arrive at

$$x_i^{k+1}=x_i^k+g_i^k\left(\left\{x_j^k,\nabla f_j\left(x_j^k\right)\right\}_{j\in\mathcal{N}(i)\cup\{i\}};\theta_i\right), \quad \forall i\in\mathcal{V},$$




**takes gradients and variables from neighbors**

**each node $i$ learns an RNN parameterized by $\theta_i$**

# Limitations

$$x_i^{k+1} = x_i^k + g_i^k\left(\left\{x_j^k, \nabla f_j\left(x_j^k\right)\right\}_{j\in\mathcal{N}(i)\cup\{i\}}; \theta_i\right), \quad \forall i \in \mathcal{V},$$

- The search space consisting of all combinations between the gradients and variables from neighbors is extremely **large**; very **difficult** to train
- Each node updates its own variables independently; **no mechanism to ensure** convergence to the optimal consensus solution $x_1 = \cdots = x_n = x^\star$
- **Weak** generalization is observed in experiments

**Can we find more structures in the update rule $g_i(\cdot)$**

# Base update rules

We reformulate the unconstrained optimization problem into the following constrained form:

$$\min_{\boldsymbol{X}\in\mathbb{R}^{n\times d}} \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x}_i) + r(\boldsymbol{x}_i), \quad \text{s.t.} \quad \boldsymbol{x}_i = \boldsymbol{x}_j, \forall\{i,j\} \in \mathcal{E}.$$

The corresponding Lagrangian function is:

$$\mathcal{L}\left(X, \{\nu_{i,j}\}_{\{i,j\}\in\mathcal{E}}\right) = \frac{1}{n}\sum_{i=1}^{n}(f_i(x_i) + r(x_i)) + \sum_{\{i,j\}\in\mathcal{E}} \langle \nu_{i,j}, x_i - x_j\rangle,$$

Solving the above function using the primal-dual algorithm yields:

$$\begin{aligned} x_i^{k+1} &= x_i^k - \tfrac{\gamma}{n}\left(\nabla f_i\left(x_i^k\right) + g_i^{k+1} + y_i^k\right), \quad g_i^{k+1} \in \partial r\left(x_i^{k+1}\right) \\ y_i^{k+1} &= y_i^k + 2\gamma \textstyle\sum_{j\in\mathcal{N}(i)}\left(x_i^{k+1} - x_j^{k+1}\right) \end{aligned}$$

# Base update rules

$$x_i^{k+1} = x_i^k - \frac{\gamma}{n}\left(\nabla f_i\left(x_i^k\right) + g_i^{k+1} + y_i^k\right), \quad g_i^{k+1} \in \partial r\left(x_i^{k+1}\right) \tag{1}$$

$$y_i^{k+1} = y_i^k + 2\gamma \sum_{j \in \mathcal{N}(i)} \left(x_i^{k+1} - x_j^{k+1}\right) \tag{2}$$

- Update (2) implies **consensus**. With $x_i^k \to x_i^*$ and $y_i^k \to y_i^*$ we have

$$x_i^* = \left(\frac{1}{|\mathcal{N}(i)|}\right) \sum_{j \in \mathcal{N}(i)} x_j^*, \quad \forall i \in \{1, \cdots, n\} \longrightarrow \boldsymbol{x}_1^\star = \cdots = \boldsymbol{x}_n^\star.$$

- Update (1) implies **optimality**. Since $x_i^k \to x_i^*$ , $y_i^k \to y_i^*$ and $\sum_{i=1}^n y_i^\star = 0$ we have

$$\nabla f_i(\boldsymbol{x}^\star) + \partial r(\boldsymbol{x}^\star) + \boldsymbol{y}_i^\star \ni 0 \longrightarrow \frac{1}{n}\sum_{i=1}^n \nabla f_i(\boldsymbol{x}^\star) + \partial r(\boldsymbol{x}^\star) \ni 0,$$

# Base update rules

$$x_i^{k+1} = x_i^k - \frac{\gamma}{n}\left(\nabla f_i\left(x_i^k\right) + g_i^{k+1} + y_i^k\right), \quad g_i^{k+1} \in \partial r\left(x_i^{k+1}\right)$$
$$y_i^{k+1} = y_i^k + 2\gamma \sum_{j\in\mathcal{N}(i)}\left(x_i^{k+1} - x_j^{k+1}\right)$$

- Inspired by the above primal-dual algorithm, we propose a new L2O update rule

$$\boldsymbol{x}_i^{k+1} = \boldsymbol{x}_i^k - \boldsymbol{m}_i^k(\nabla f_i(\boldsymbol{x}_i^k), \boldsymbol{g}_i^{k+1}, \boldsymbol{y}_i^k; \boldsymbol{\theta}_{i,1}), \quad \boldsymbol{g}_i^{k+1} \in \partial r(\boldsymbol{x}_i^{k+1}),$$
$$\boldsymbol{y}_i^{k+1} = \boldsymbol{y}_i^k + \boldsymbol{s}_i^k(\{\boldsymbol{x}_i^{k+1} - \boldsymbol{x}_j^{k+1}\}_{j\in\mathcal{N}(i)}; \boldsymbol{\theta}_{i,2}),$$

- Compared with the following trivial update rule

$$x_i^{k+1} = x_i^k + g_i^k\left(\left\{x_j^k, \nabla f_j\left(x_j^k\right)\right\}_{j\in\mathcal{N}(i)\cup\{i\}}; \theta_i\right), \quad \forall i \in \mathcal{V},$$

We introduce the **dual variable y** to enforce consensus constraints

# Base update rules

$$\boldsymbol{x}_i^{k+1} = \boldsymbol{x}_i^k - \boldsymbol{m}_i^k\left(\nabla f_i\left(\boldsymbol{x}_i^k\right), \boldsymbol{g}_i^{k+1}, \boldsymbol{y}_i^k; \boldsymbol{\theta}_{i,1}\right), \quad \boldsymbol{g}_i^{k+1} \in \partial r\left(\boldsymbol{x}_i^{k+1}\right)$$
$$\boldsymbol{y}_i^{k+1} = \boldsymbol{y}_i^k + \boldsymbol{s}_i^k\left(\left\{\boldsymbol{x}_i^{k+1} - \boldsymbol{x}_j^{k+1}\right\}_{j\in\mathcal{N}(i)}; \boldsymbol{\theta}_{i,2}\right)$$

- However, the above update rules does not fully utilize neighboring info to update $x$
- We thus make a small adjustment to ensure that neighboring info benefits both $x$ and $y$

$$\boldsymbol{z}_i^{k+1} = \boldsymbol{x}_i^k - \boldsymbol{m}_i^k\left(\nabla f_i\left(\boldsymbol{x}_i^k\right), \boldsymbol{g}_i^{k+1}, \boldsymbol{y}_i^k; \boldsymbol{\theta}_{i,1}\right), \quad \boldsymbol{g}_i^{k+1} \in \partial r\left(\boldsymbol{z}_i^{k+1}\right)$$
$$\boldsymbol{y}_i^{k+1} = \boldsymbol{y}_i^k + \boldsymbol{s}_i^k\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j\in\mathcal{N}(i)}; \boldsymbol{\theta}_{i,2}\right)$$
$$\boldsymbol{x}_i^{k+1} = \boldsymbol{z}_i^{k+1} - \boldsymbol{u}_i^k\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j\in\mathcal{N}(i)}; \boldsymbol{\theta}_{i,3}\right)$$

**Base update rule**

# Mathematics-inspired update rules

- Does the base update rule converge to the consensus and optimal solution?

  **No theoretical guarantees. There exists bad $m_i, s_i$ and $u_i$ to deviate the convergence**

- What conditions does a good base update rule have to satisfy?

- Can these conditions provide guidance for the development of $m_i, s_i$ and $u_i$?

# Mathematics-inspired update rules

$$
\begin{aligned}
\boldsymbol{z}_i^{k+1} &= \boldsymbol{x}_i^k - \boldsymbol{m}_i^k\left(\nabla f_i\left(\boldsymbol{x}_i^k\right), \boldsymbol{g}_i^{k+1}, \boldsymbol{y}_i^k; \boldsymbol{\theta}_{i,1}\right), \quad \boldsymbol{g}_i^{k+1} \in \partial r\left(\boldsymbol{z}_i^{k+1}\right) \\
\boldsymbol{y}_i^{k+1} &= \boldsymbol{y}_i^k + \boldsymbol{s}_i^k\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j \in \mathcal{N}(i)}; \boldsymbol{\theta}_{i,2}\right) \\
\boldsymbol{x}_i^{k+1} &= \boldsymbol{z}_i^{k+1} - \boldsymbol{u}_i^k\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j \in \mathcal{N}(i)}; \boldsymbol{\theta}_{i,3}\right)
\end{aligned}
$$

If $(x_i^k, y_i^k, z_i^k)$ stays at the optimal solution $(x^\star, y_i^\star, z_i^\star)$, the next iterate $(x_i^{k+1}, y_i^{k+1}, z_i^{k+1})$ should be fixed

Condition 1 (FIXED POINT):For any $\boldsymbol{x}^\star \in \arg\min_{\boldsymbol{x} \in \mathbb{R}^d} f(\boldsymbol{x}) + r(\boldsymbol{x}), \boldsymbol{g}_i^\star \in \partial r(\boldsymbol{x}^\star)$ and $\boldsymbol{y}_i^\star = -\nabla f_i(\boldsymbol{x}^\star) - \boldsymbol{g}_i^\star$, it holds for any $i \in \mathcal{V}$ that

$$
\lim_{k\to\infty} \boldsymbol{m}_i^k(\nabla f_i(\boldsymbol{x}^\star), \boldsymbol{g}_i^\star, \boldsymbol{y}_i^\star) = \lim_{k\to\infty} \boldsymbol{s}_i^k\left(\{\mathbf{0}_d\}_{j \in \mathcal{N}(i)}\right) = \lim_{k\to\infty} \boldsymbol{u}_i^k\left(\{\mathbf{0}_d\}_{j \in \mathcal{N}(i)}\right) = \mathbf{0}_d
$$

# Mathematics-inspired update rules

$$
\begin{aligned}
\boldsymbol{z}_i^{k+1} &= \boldsymbol{x}_i^k - \boldsymbol{m}_i^k\left(\nabla f_i\left(\boldsymbol{x}_i^k\right), \boldsymbol{g}_i^{k+1}, \boldsymbol{y}_i^k; \boldsymbol{\theta}_{i,1}\right), \quad \boldsymbol{g}_i^{k+1} \in \partial r\left(\boldsymbol{z}_i^{k+1}\right) \\
\boldsymbol{y}_i^{k+1} &= \boldsymbol{y}_i^k + \boldsymbol{s}_i^k\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j \in \mathcal{N}(i)}; \boldsymbol{\theta}_{i,2}\right) \\
\boldsymbol{x}_i^{k+1} &= \boldsymbol{z}_i^{k+1} - \boldsymbol{u}_i^k\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j \in \mathcal{N}(i)}; \boldsymbol{\theta}_{i,3}\right)
\end{aligned}
$$

Any fixed points should converge to the optimal primal and dual solution $(x^\star, y_i^\star, z_i^\star)$

Condition 2 (GLOBAL CONVERGENCE): For any sequences generated by the base update rules, *there exists* $\boldsymbol{x}^\star \in \arg\min_{\boldsymbol{x} \in \mathbb{R}^d} f(\boldsymbol{x}) + r(\boldsymbol{x})$, $\boldsymbol{y}_i^\star \in -\nabla f_i(\boldsymbol{x}^\star) - \partial r(\boldsymbol{x}^\star)$ such that

$$
\lim_{k \to \infty} \boldsymbol{x}_i^k = \lim_{k \to \infty} \boldsymbol{z}_i^k = \boldsymbol{x}^\star, \quad \lim_{k \to \infty} \boldsymbol{y}_i^k = \boldsymbol{y}_i^\star, \quad \forall\, i \in \mathcal{V}
$$

# Mathematics-inspired update rules

Conditions 1 and 2 provide guidance to develop explicit structures for base update rule

Theorem 1 (MATHEMATICS-INSPIRED STRUCTURE): Given $f_i \in \mathcal{F}_L(\mathbb{R}^d)$, $r \in \mathcal{F}(\mathbb{R}^d)$ and base update rules $\{\boldsymbol{m}_i^k,\ \boldsymbol{s}_i^k,\ \boldsymbol{u}_i^k\}_{k=0}^{\infty}$ with $\boldsymbol{m}_i^k \in \mathcal{D}_C(\mathbb{R}^{3d})$, $\boldsymbol{s}_i^k,\ \boldsymbol{u}_i^k \in \mathcal{D}_C\left(\mathbb{R}^{|\mathcal{N}(i)|d}\right)$, if Conditions 1 and 2 hold, there exist $\boldsymbol{P}_i^k,\ \boldsymbol{P}_{i,j,1}^k,\ \boldsymbol{P}_{i,j,2}^k \in \mathbb{R}^{d\times d}$ and $\boldsymbol{b}_{i,1}^k,\ \boldsymbol{b}_{i,2}^k,\ \boldsymbol{b}_{i,3}^k \in \mathbb{R}^d$ satisfying

$$\boldsymbol{m}_i^k\left(\nabla f_i\left(\boldsymbol{x}_i^k\right), \boldsymbol{g}_i^{k+1}, \boldsymbol{y}_i^k\right) = \boldsymbol{P}_i^k\left(\nabla f_i\left(\boldsymbol{x}_i^k\right) + \boldsymbol{g}_i^{k+1} + \boldsymbol{y}_i^k\right) + \boldsymbol{b}_{i,1}^k$$

$$\boldsymbol{s}_i^k\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j\in\mathcal{N}(i)}\right) = \sum_{j\in\mathcal{N}(i)} \boldsymbol{P}_{i,j,1}^k\left(\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right) + \boldsymbol{b}_{i,2}^k$$

$$\boldsymbol{u}_i^k\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j\in\mathcal{N}(i)}\right) = \sum_{j\in\mathcal{N}(i)} \boldsymbol{P}_{i,j,2}^k\left(\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right) + \boldsymbol{b}_{i,3}^k$$

We name it as **structured update rule**. $P_i^k$, $P_{i,j,1}^k$, $P_{i,j,2}^k$ and $b_{i,1}^k$, $b_{i,2}^k$ and $b_{i,3}^k$ are parameters to learn.

# MiLoDo update rules

- To improve computational efficiency, we make the *P* matrices diagonal.

- Inspired by Theorem 1, which shows that the bias terms $b_{i,1}^k, b_{i,2}^k, b_{i,3}^k$ vanish asymptotically, we eliminate these terms for simplicity.

- Specifically, we set:

$$\boldsymbol{P}_i^k = \mathrm{Diag}\left(\boldsymbol{p}_i^k\right), \boldsymbol{P}_{i,j,1}^k = \mathrm{Diag}\left(\boldsymbol{p}_{i,j,1}^k\right), \boldsymbol{P}_{i,j,2}^k = \mathrm{Diag}\left(\boldsymbol{p}_{i,j,2}^k\right), \boldsymbol{b}_{i,1}^k = \boldsymbol{b}_{i,2}^k = \boldsymbol{b}_{i,3}^k = \mathbf{0}_d,$$

$$\text{where } \boldsymbol{p}_i^k, \boldsymbol{p}_{i,j,1}^k, \boldsymbol{p}_{i,j,2}^k \in \mathbb{R}_+^d, \text{ and } \boldsymbol{p}_{i,j,1}^k = \boldsymbol{p}_{j,i,1}^k, \forall \{i,j\} \in \boldsymbol{\mathcal{E}}$$

# MiLoDo update rules

Finally, we reach the MiLoDo update rules

$$
\begin{aligned}
\boldsymbol{z}_i^{k+1} &= \operatorname{prox}_{r,\operatorname{Diag}(\boldsymbol{p}_i^k)}\left(\boldsymbol{x}_i^k - \boldsymbol{p}_i^k \odot \left(\nabla f_i\left(\boldsymbol{x}_i^k\right) + \boldsymbol{y}_i^k\right)\right) \\
\boldsymbol{y}_i^{k+1} &= \boldsymbol{y}_i^k + \sum_{j\in\mathcal{N}(i)} \boldsymbol{p}_{i,j,1}^k \odot \left(\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right) \\
\boldsymbol{x}_i^{k+1} &= \boldsymbol{z}_i^{k+1} - \sum_{j\in\mathcal{N}(i)} \boldsymbol{p}_{i,j,2}^k \odot \left(\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right)
\end{aligned}
$$

MiLoDo: **M**athematics-**i**nspired **L**earning-to-**o**ptimize framework for **D**ecentralized **o**ptimization

# Theoretical guarantees for MiLoDo

北京大学 PEKING UNIVERSITY

**Any fixed points of MiLoDo update rules are the primal and dual optimal solutions.**

**Theorem 2** (EXACT CONVERGENCE)**:** Assume $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is strongly connected, $\{f_i\}_{i\in\mathcal{V}} \subset \mathcal{F}_L(\mathbb{R}^d)$, $r \in \mathcal{F}(\mathbb{R}^d)$ and there exists $0 < m < M < \infty$ such that $[\boldsymbol{p}_i^k]_l \geq m$, $m \leq [\boldsymbol{p}_{i,j,1}^k]_l \leq M$, $|[\boldsymbol{p}_{i,j,2}^k]_l| \leq M$ for all $k \geq 0$ and $1 \leq l \leq d$. Here $[\boldsymbol{x}]_l$ denotes the $l$-th coordinate of the vector $\boldsymbol{x}$. If a sequence generated by (22)-(24) with initialization $\boldsymbol{y}_i^0 = \mathbf{0}_d$ converges to $\{\boldsymbol{x}_i^\star, \boldsymbol{y}_i^\star, \boldsymbol{z}_i^\star\}_{i=1}^n$, then this limit must be the primal and dual optimal solutions to problem (2). In other words, there exists $\boldsymbol{x}^\star \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x}) + r(\boldsymbol{x})$ such that $\boldsymbol{x}_i^\star = \boldsymbol{z}_i^\star = \boldsymbol{x}^\star$ and $\boldsymbol{y}_i^\star \in -\nabla f_i(\boldsymbol{x}^\star) - \partial r(\boldsymbol{x}^\star)$ holds.

# MiLoDo Evolution

北京大学 PEKING UNIVERSITY

## ① Generic L2O

$$x_i^{k+1} = x_i^k + g_i^k\left(\left\{x_j^k, \nabla f_j\left(x_j^k\right)\right\}_{j\in\mathcal{N}(i)\cup\{i\}}; \theta_i\right)$$

Inspired by primal-dual algorithm

## ② Base rule

$$\begin{aligned}
\boldsymbol{z}_i^{k+1} &= \boldsymbol{x}_i^k - \boldsymbol{m}_i^k\left(\nabla f_i\left(\boldsymbol{x}_i^k\right), \boldsymbol{g}_i^{k+1}, \boldsymbol{y}_i^k; \boldsymbol{\theta}_{i,1}\right), \\
\boldsymbol{y}_i^{k+1} &= \boldsymbol{y}_i^k + \boldsymbol{s}_i^k\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j\in\mathcal{N}(i)}; \boldsymbol{\theta}_{i,2}\right) \\
\boldsymbol{x}_i^{k+1} &= \boldsymbol{z}_i^{k+1} - \boldsymbol{u}_i^k\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j\in\mathcal{N}(i)}; \boldsymbol{\theta}_{i,3}\right)
\end{aligned}$$

Fixed-point cond.

Global converge

## ③ Math-inspired rule

$$\begin{aligned}
\boldsymbol{m}_i^k\left(\nabla f_i\left(\boldsymbol{x}_i^k\right), \boldsymbol{g}_i^{k+1}, \boldsymbol{y}_i^k\right) &= \boldsymbol{P}_i^k\left(\nabla f_i\left(\boldsymbol{x}_i^k\right) + \boldsymbol{g}_i^{k+1} + \boldsymbol{y}_i^k\right) + \boldsymbol{b}_{i,1}^k \\
\boldsymbol{s}_i^k\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j\in\mathcal{N}(i)}\right) &= \sum_{j\in\mathcal{N}(i)} \boldsymbol{P}_{i,j,1}^k\left(\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right) + \boldsymbol{b}_{i,2}^k \\
\boldsymbol{u}_i^k\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j\in\mathcal{N}(i)}\right) &= \sum_{j\in\mathcal{N}(i)} \boldsymbol{P}_{i,j,2}^k\left(\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right) + \boldsymbol{b}_{i,3}^k
\end{aligned}$$

Simplification

## ④ MiLoDo

$$\begin{aligned}
\boldsymbol{z}_i^{k+1} &= \mathrm{prox}_{r,\mathrm{Diag}\left(\boldsymbol{p}_i^k\right)}\left(\boldsymbol{x}_i^k - \boldsymbol{p}_i^k \odot \left(\nabla f_i\left(\boldsymbol{x}_i^k\right) + \boldsymbol{y}_i^k\right)\right) \\
\boldsymbol{y}_i^{k+1} &= \boldsymbol{y}_i^k + \sum_{j\in\mathcal{N}(i)} \boldsymbol{p}_{i,j,1}^k \odot \left(\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right) \\
\boldsymbol{x}_i^{k+1} &= \boldsymbol{z}_i^{k+1} - \sum_{j\in\mathcal{N}(i)} \boldsymbol{p}_{i,j,2}^k \odot \left(\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right)
\end{aligned}$$

Introduce math structures to L2O framework to narrow down search space and promote generalization

# Generic L2O vs MiLoDo

## Generic L2O

$$x_i^{k+1} = x_i^k + g_i^k\left(\left\{x_j^k, \nabla f_j\left(x_j^k\right)\right\}_{j\in\mathcal{N}(i)\cup\{i\}}; \theta_i\right)$$

- **Large** search space; **difficult** to train
- Consensus and optimality are **NOT** ensured
- **Weak** out-of-distribution generalization

## MiLoDo

$$\begin{aligned}
\boldsymbol{z}_i^{k+1} &= \mathrm{prox}_{r,\mathrm{Diag}(\boldsymbol{p}_i^k)}\left(\boldsymbol{x}_i^k - \boldsymbol{p}_i^k \odot \left(\nabla f_i(\boldsymbol{x}_i^k) + \boldsymbol{y}_i^k\right)\right), \\
\boldsymbol{y}_i^{k+1} &= \boldsymbol{y}_i^k + \sum_{j\in\mathcal{N}(i)} \boldsymbol{p}_{i,j,1}^k \odot (\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}), \\
\boldsymbol{x}_i^{k+1} &= \boldsymbol{z}_i^{k+1} - \sum_{j\in\mathcal{N}(i)} \boldsymbol{p}_{i,j,2}^k \odot (\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}).
\end{aligned}$$

- **Compact** search space; **easy** to train
- Consensus and optimality are **ensured**
- **Strong** out-of-distribution generalization

PART 04

# Parameterization and Training

# MiLoDo update rules

$$\boldsymbol{z}_i^{k+1} = \mathrm{prox}_{r,\mathrm{Diag}(\boldsymbol{p}_i^k)}\left(\boldsymbol{x}_i^k - \boldsymbol{p}_i^k \odot (\nabla f_i(\boldsymbol{x}_i^k) + \boldsymbol{y}_i^k)\right),$$

$$\boldsymbol{y}_i^{k+1} = \boldsymbol{y}_i^k + \sum_{j\in\mathcal{N}(i)} \boldsymbol{p}_{i,j,1}^k \odot (\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}),$$

$$\boldsymbol{x}_i^{k+1} = \boldsymbol{z}_i^{k+1} - \sum_{j\in\mathcal{N}(i)} \boldsymbol{p}_{i,j,2}^k \odot (\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}).$$

How to learn $p_i^k, p_{i,j,1}^k, p_{i,j,2}^k$ with deep neural network?

# LSTM parameterization

$$\boldsymbol{p}_i^k, \boldsymbol{h}_{M,i}^{k+1} = \phi_{M,i}\left(\nabla f\left(\boldsymbol{x}_i^k\right), \boldsymbol{y}_i^k, \boldsymbol{h}_{M,i}^k; \boldsymbol{\theta}_{M,i}\right),$$

$$\left\{\tilde{\boldsymbol{p}}_{i,j,1}^k\right\}_{j\in\mathcal{N}(i)}, \boldsymbol{h}_{S,i}^{k+1} = \phi_{S,i}\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j\in\mathcal{N}(i)}, \boldsymbol{h}_{S,i}^k; \boldsymbol{\theta}_{S,i}\right),$$

$$\left\{\boldsymbol{p}_{i,j,2}^k\right\}_{j\in\mathcal{N}(i)}, \boldsymbol{h}_{U,i}^{k+1} = \phi_{U,i}\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j\in\mathcal{N}(i)}, \boldsymbol{h}_{U,i}^k; \boldsymbol{\theta}_{U,i}\right),$$

- We parameterize $\boldsymbol{p}_i^k, \boldsymbol{p}_{i,j,1}^k, \boldsymbol{p}_{i,j,2}^k$ using three local LSTM networks $\phi_{M,i}, \phi_{S,i}, \phi_{U,i}$.
- LSTM hidden states $\boldsymbol{h}_{M,i}^k, \boldsymbol{h}_{S,i}^k, \boldsymbol{h}_{U,i}^k$ are randomly initialized, and parameters $\boldsymbol{\theta}_{M,i}, \boldsymbol{\theta}_{S,i}, \boldsymbol{\theta}_{U,i}$ are learnable.

# MiLoDo operation

$$\nabla f_i(\mathbf{x}_i^k), \mathbf{y}_i^k, \{\mathbf{z}_i^{k+1} - \mathbf{z}_j^{k+1}\}_{j\in\mathcal{N}(i)}$$

## Optimization

$$\begin{aligned}
\boldsymbol{z}_i^{k+1} &= \mathrm{prox}_{r,\mathrm{Diag}(\boldsymbol{p}_i^k)}\left(\boldsymbol{x}_i^k - \boldsymbol{p}_i^k \odot (\nabla f_i(\boldsymbol{x}_i^k) + \boldsymbol{y}_i^k)\right), \\
\boldsymbol{y}_i^{k+1} &= \boldsymbol{y}_i^k + \sum_{j\in\mathcal{N}(i)} \boldsymbol{p}_{i,j,1}^k \odot (\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}), \\
\boldsymbol{x}_i^{k+1} &= \boldsymbol{z}_i^{k+1} - \sum_{j\in\mathcal{N}(i)} \boldsymbol{p}_{i,j,2}^k \odot (\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}).
\end{aligned}$$

(solve the target optimization problem)

## Learning

$$\begin{aligned}
\boldsymbol{p}_i^k, \boldsymbol{h}_{M,i}^{k+1} &= \phi_{M,i}\left(\nabla f\left(\boldsymbol{x}_i^k\right), \boldsymbol{y}_i^k, \boldsymbol{h}_{M,i}^k; \boldsymbol{\theta}_{M,i}\right), \\
\left\{\tilde{\boldsymbol{p}}_{i,j,1}^k\right\}_{j\in\mathcal{N}(i)}, \boldsymbol{h}_{S,i}^{k+1} &= \phi_{S,i}\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j\in\mathcal{N}(i)}, \boldsymbol{h}_{S,i}^k; \boldsymbol{\theta}_{S,i}\right), \\
\left\{\boldsymbol{p}_{i,j,2}^k\right\}_{j\in\mathcal{N}(i)}, \boldsymbol{h}_{U,i}^{k+1} &= \phi_{U,i}\left(\left\{\boldsymbol{z}_i^{k+1} - \boldsymbol{z}_j^{k+1}\right\}_{j\in\mathcal{N}(i)}, \boldsymbol{h}_{U,i}^k; \boldsymbol{\theta}_{U,i}\right),
\end{aligned}$$

(learn the parameters $p_i, p_{i,j,1}, p_{i,j,2}$)

$$\mathbf{p}_i^k, \{\tilde{\mathbf{p}}_{i,j,1}^k\}_{j\in\mathcal{N}(i)}, \{\mathbf{p}_{i,j,2}^k\}_{j\in\mathcal{N}(i)}$$

# MiLoDo training

- The parameters to train is $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_{M,i}, \boldsymbol{\theta}_{S,i}, \boldsymbol{\theta}_{U,i}\}_{i=1}^{n}$

- We evaluate and refine the performance of the optimizer over the initial K steps on a batch of training problems

$$\min_{\boldsymbol{\Theta}} \mathcal{L}_K(\boldsymbol{\Theta}, \mathcal{F}_B) := \frac{1}{|\mathcal{F}_B|} \sum_{f \in \mathcal{F}_B} \left[ \frac{1}{K} \sum_{k=1}^{K} f(\bar{\boldsymbol{x}}^k) \right].$$

  where $\bar{\boldsymbol{x}}^k = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i^k$ is the global average of all local variables

- We typically set K = 100 during the training phrase

# Pre-training + fine-tuning

Inspired by LLM fine-tuning techniques, we propose the following innovative training strategy:

1. **Special Initialization:**

- Initialize MiLoDo parameters with traditional decentralized optimization algorithms instead of random values. (e.g., EXTRA, Exact Diffusion)

2. **Mixed Dataset Pre-Training:**

- **Pre-train** MiLoDo on a mixed dataset of problems with various dimensions to learn general optimization strategies, enhancing the generalization ability.

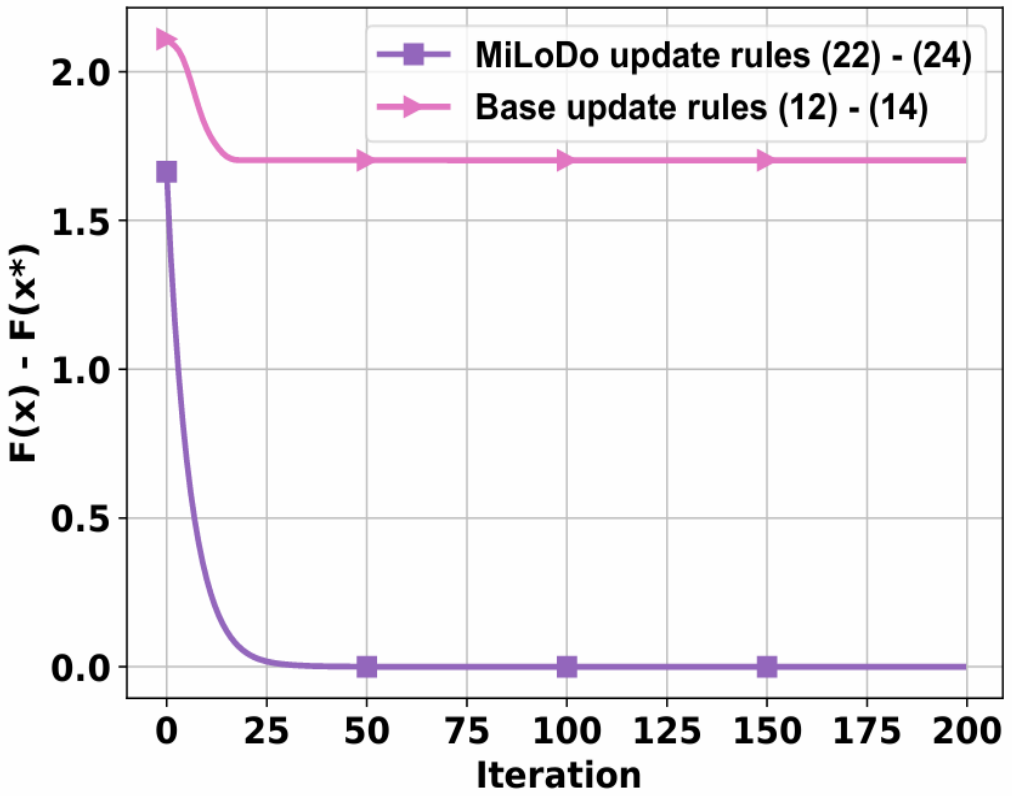
- Then, **fine-tune** on specific tasks to improve adaptability and performance.

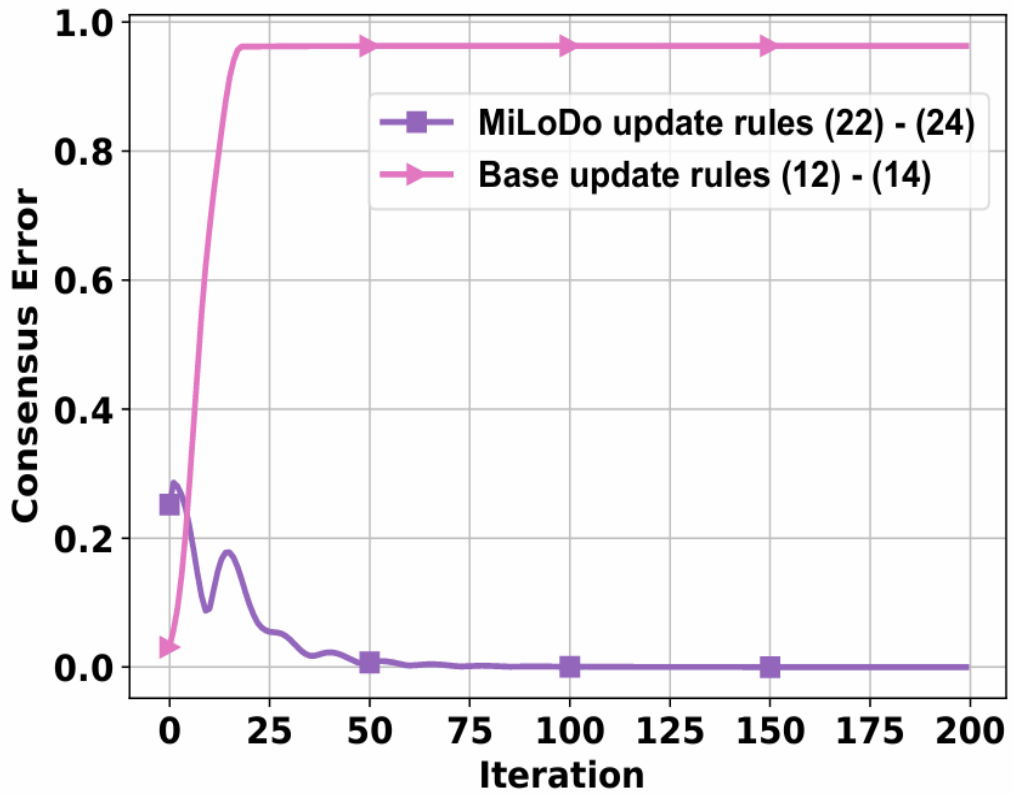
PART 05

# Experimental results

# MiLoDo vs Generic L2O

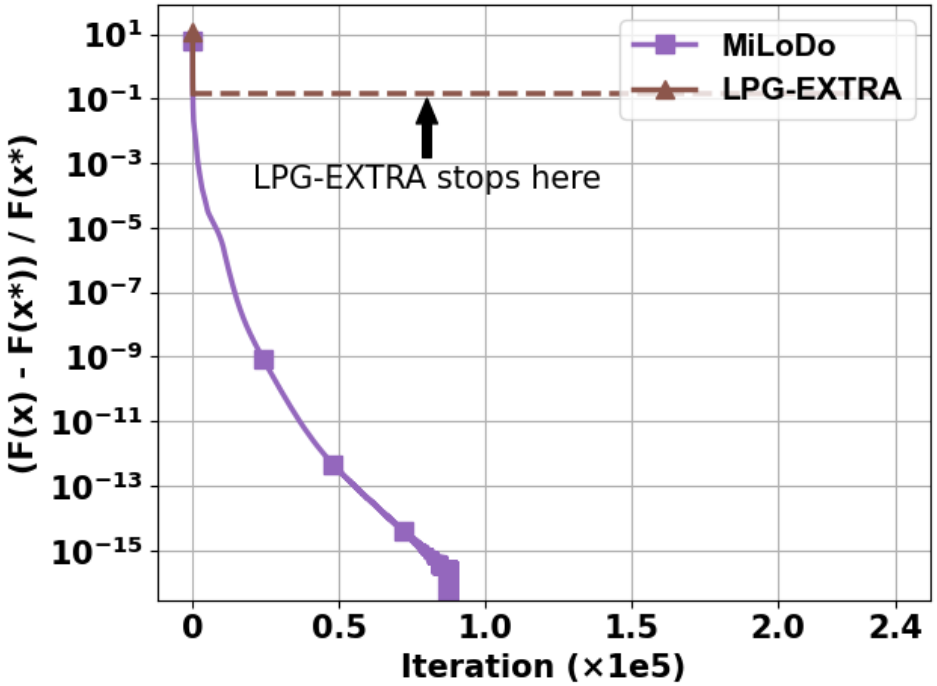
LASSO problem (10,10,5,0)

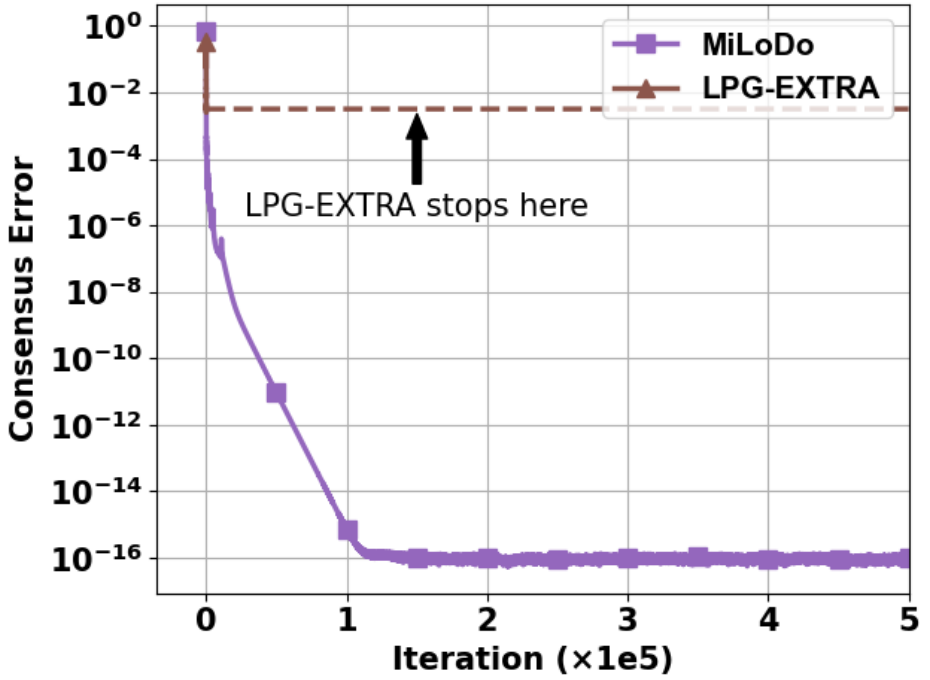


# MiLoDo vs algorithm unrolling

LPG-EXTRA: PG-EXTRA with neural network unrolling; can only run 100 iterations




H. Wang, Y. Shen, Z. Wang, D. Li, J. Zhang, K. B. Letaief, and J. Lu. Decentralized statistical inference with unrolled graph neural networks, IEEE CDC, 2021

# Significant acceleration

In tests on LASSO and logistic optimization problems, MiLoDo achieves approximately **1.8 to 2.5 times speedup** compared to other hand-crafted optimization algorithms.

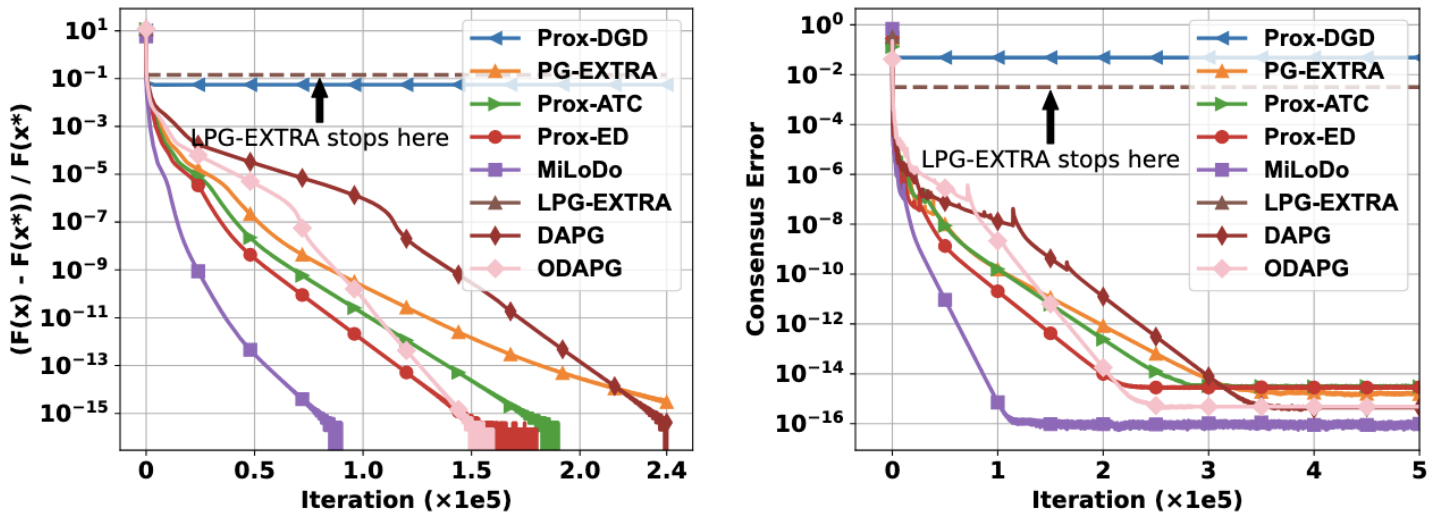



Figure 2: MiLoDo-optimizer trained on synthetic LASSO$(10, 300, 10, 0.1)$ and tested on unseen LASSO$(10, 300, 10, 0.1)$ instances.

**The first learned optimizer that can beat man-crafted optimizer in decentralized optimization!**

# Generalization across different dimensions

The MiLoDo optimizer, trained on low-dimensional problems, successfully **generalizes to high-dimensional problems** during inference, demonstrating exceptional scalability.

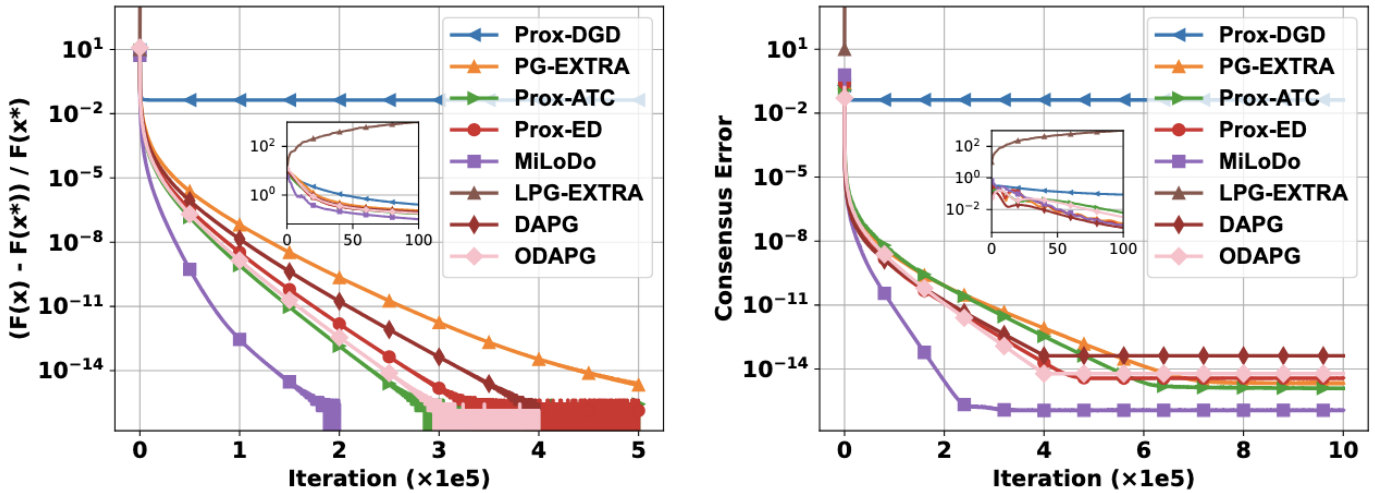



Figure 3: MiLoDo-optimizer trained on synthetic LASSO$(10, 300, 10, 0.1)$ and tested on synthetic LASSO$(10, 30000, 1000, 0.1)$.

**Generalization from d = 300 to d = 30000**

# Generalization from short training to long inference

The MiLoDo optimizer, limited to **100 iterations** during training, can extend to over **100,000 iterations** during inference. This exceptional scalability significantly enhances its practical value.

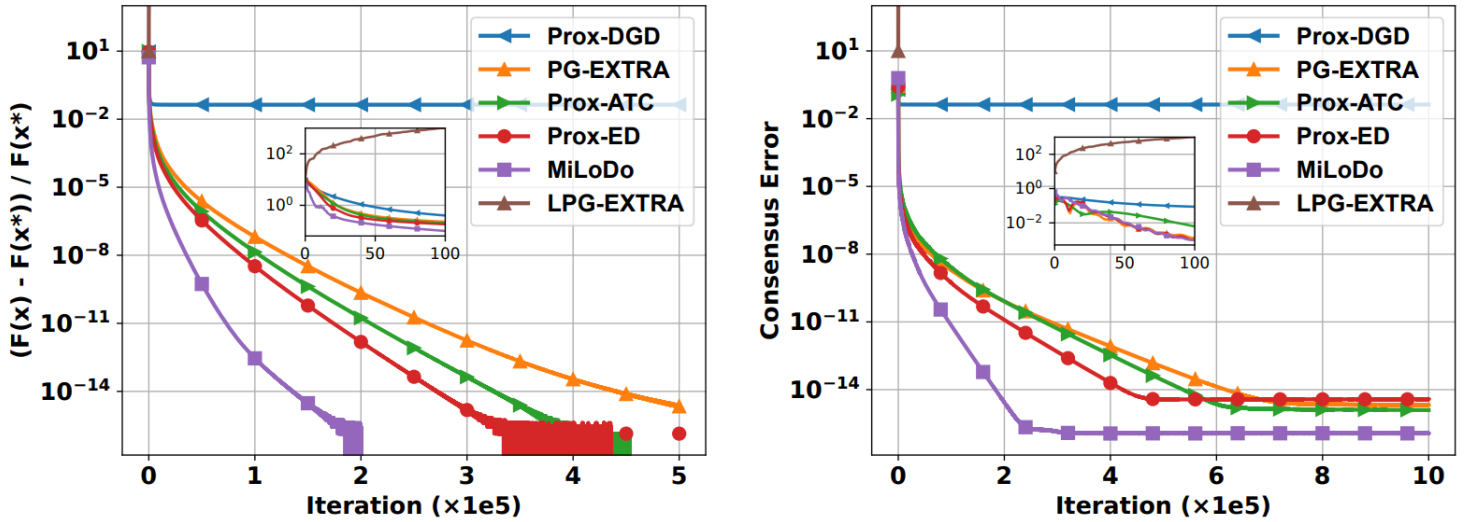



Figure 3: MiLoDo-optimizer trained on synthetic LASSO$(10, 300, 10, 0.1)$ and tested on synthetic LASSO$(10, 30000, 1000, 0.1)$.

**Trained to operate for 100 iterations, but performs well for 10000 iterations when solving unseen problems !**

# Generalization across different tasks

MiLoDo generalizes across task types. Trained on **LASSO problems**, it accurately converges on **logistic regression tasks** of various dimensions.

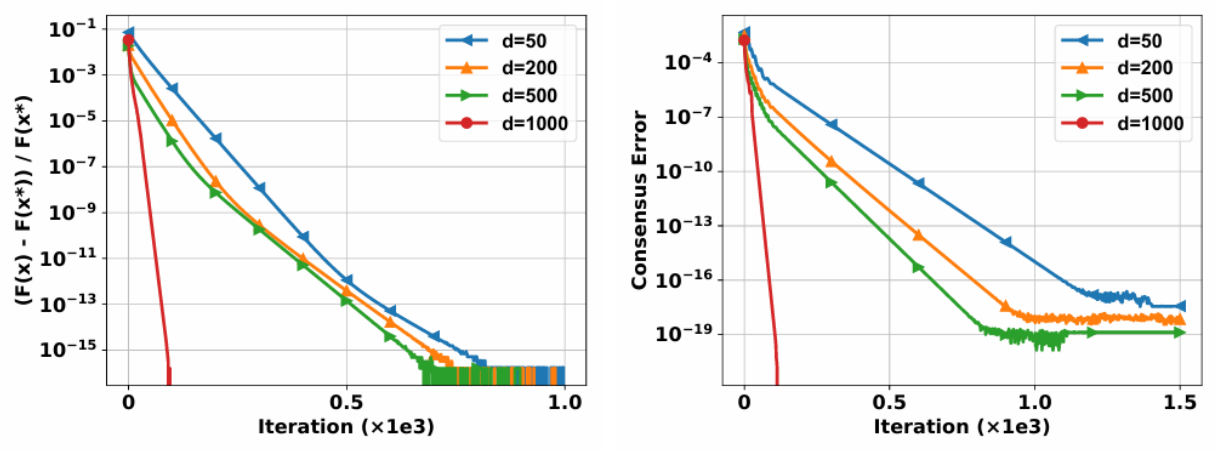



Figure 5: MiLoDo-optimizer trained on meta training set and tested on Logistic$(10, d, 100, 0.1)$ with $d \in \{50, 200, 500, 1000\}$.

**Trained to LASSO but generalizes to logistic regression!**

# Out-of-Distribution Testing

Trained on **synthetic datasets**, MiLoDo shows clear advantages when tested on the challenging **real-world BSDS500 dataset**.

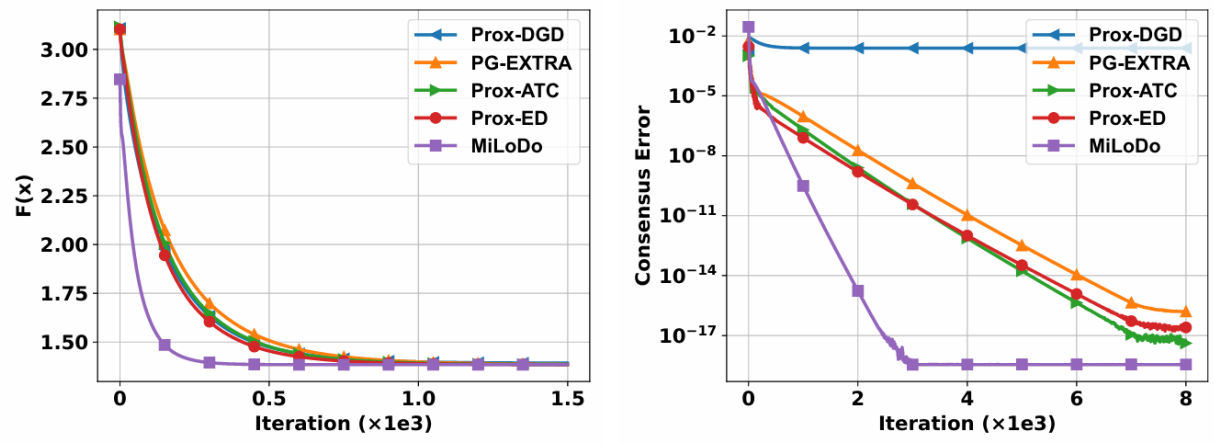
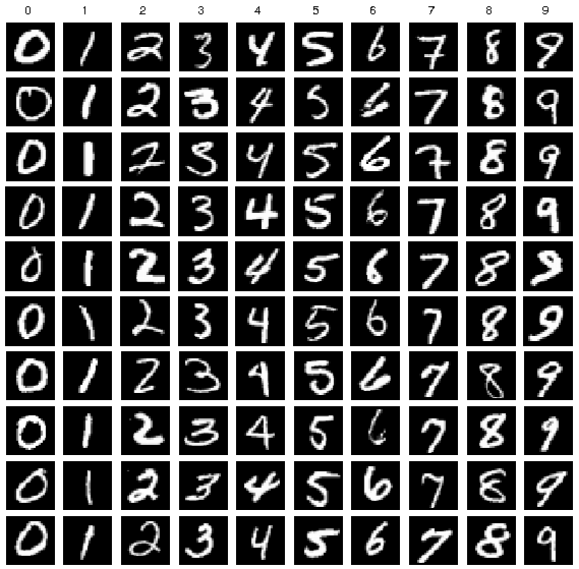
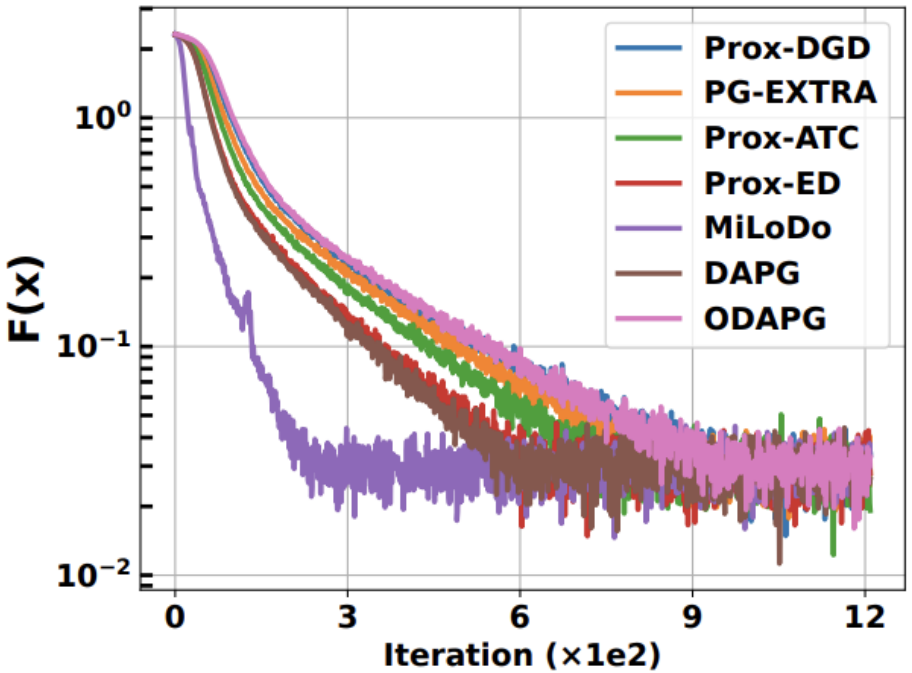


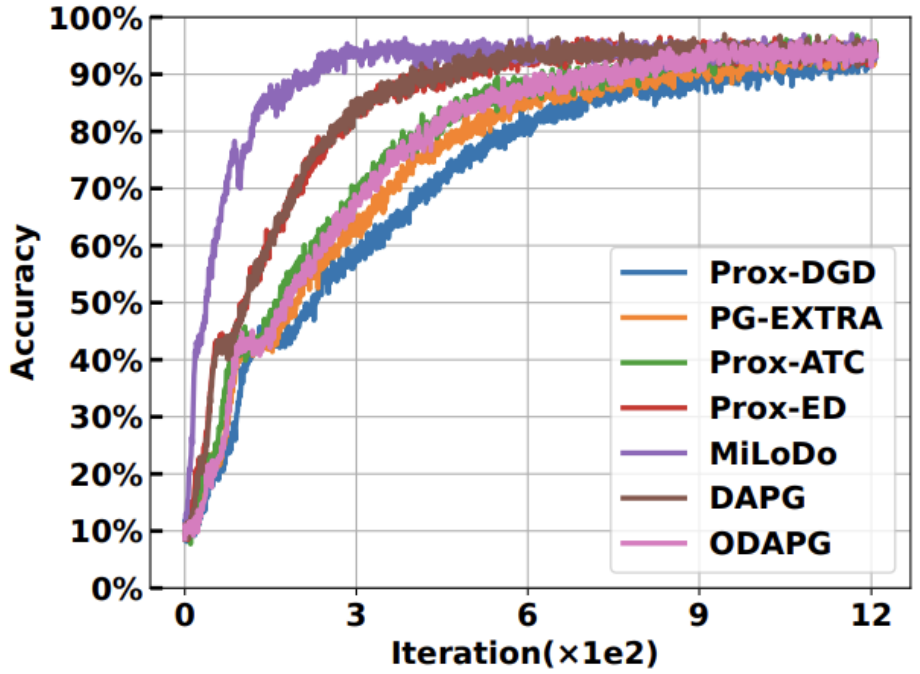

Figure 4: MiLoDo-optimizer trained on meta training set and tested on LASSO$(10, 200, 10, 0.05)$ with real dataset BSDS500[29].

**Trained on synthetic datasets but generalizes to real dataset !**

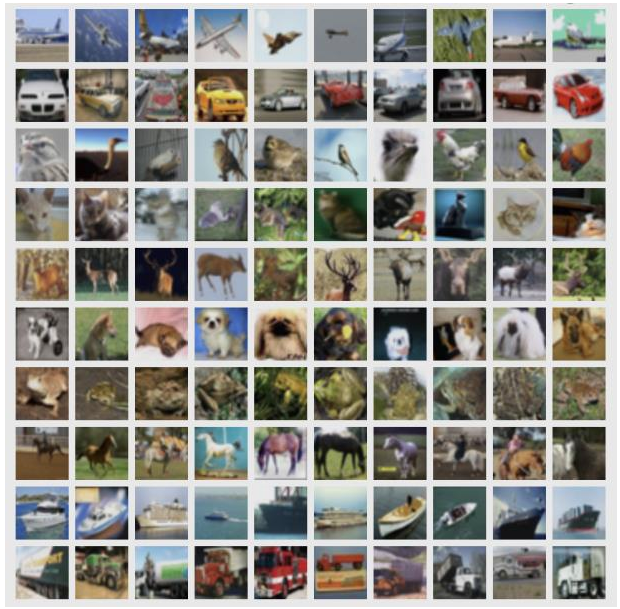
# Multi-layer neural network (MLP)

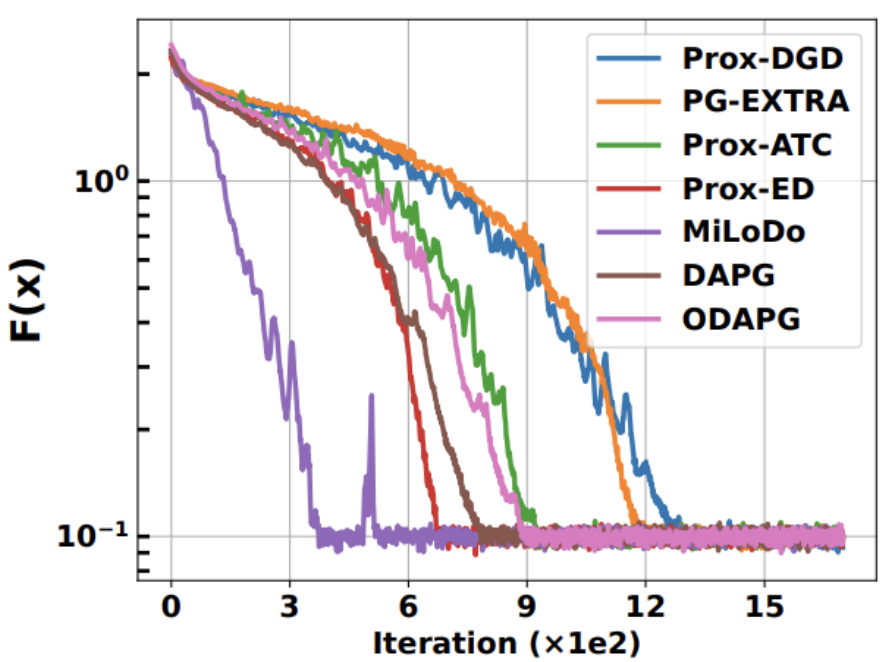


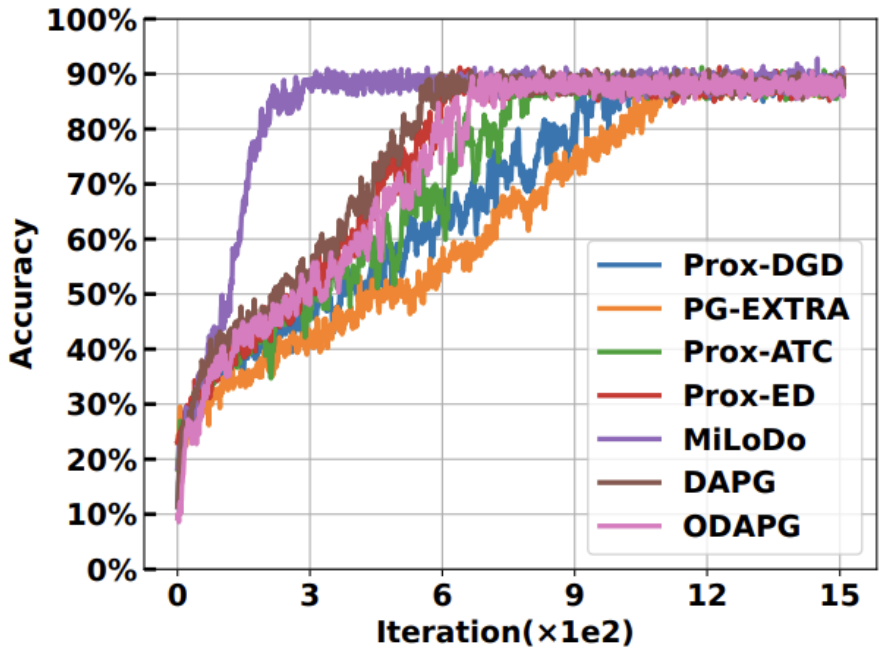



# ResNet on Cifar-10





# MiLoDo runtime

Table 1: Runtime Comparison on LASSO$(10, 30000, 1000, 0.1)$.

| | | Stopping condition: Gap $< 10^{-7}$ | | Stopping condition: Gap $< 10^{-15}$ | |
|---|---|---|---|---|---|
| | Time/Iters | Iters | Total Time | Iters | Total Time |
| MiLoDo | 5.91 ms | 2.45e+04 | 144.80 s | 1.62e+05 | 957.42 s |
| Prox-ED | 4.99 ms | 6.22e+04 | 310.38 s | 3.07e+05 | 1531.93 s |

- Increases the per-iteration computational cost by 18.4%
- Achieves a convergence speedup by 1.6x ~ 2.1x

**Saves the total time significantly**

# Summary

Can we develop an effective L2O algorithm for decentralized optimization that is

- **Memory efficient?** **Yes! MiLoDo avoids unrolling the algorithms**
- **Easy to train?** **Yes! MiLoDo introduces math structures and narrow down search space**
- **Enforcing consensus constraint?** **Yes! MiLoDo has theoretically guarantees**
- **Strong generalization?** **Yes! The math principles are critical to ensure strong generalization**

# Take-home message

北京大学 PEKING UNIVERSITY

**Mathematical structures are fundamental in Learning-to-Optimize!**

# Future work

北京大学 PEKING UNIVERSITY

- How to generalize across different network sizes and topologies?

- How to develop L2O framework for decentralized **stochastic** optimization which is crucial in deep learning tasks?

- We find the potential of pretraning + finetuning in this work. Can we develop foundational large models for decentralized optimization?

# Recent work on optimization for Large language models (LLMs)

Subspace Optimization for Large Language Models with Convergence Guarantees

https://arxiv.org/pdf/2410.11289

Enhancing Zeroth-Order Fine-Tuning for Language Models with Low-rank Structures

https://arxiv.org/pdf/2410.07698

**Significantly save memory cost to pre-train or fine-tune LLMs**

北京大学
PEKING UNIVERSITY

# Thank you!

**Paper address:** https://arxiv.org/pdf/2410.01700

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