北京大学
PEKING UNIVERSITY

Optimization for Deep Learning

Memory-Efficient Optimizers

Kun Yuan Peking University

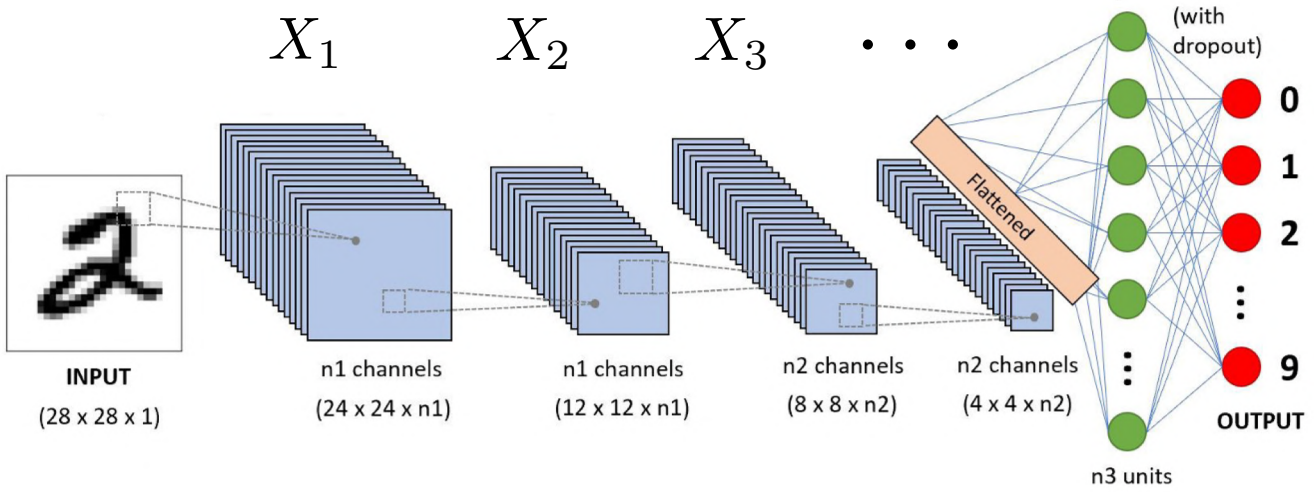
北京大学
PEKING UNIVERSITY

PART 01

Basics and Motivation

LLM pretraining is essentially solving stochastic optimization

- The model weights in neural networks are a set of matrices $\boldsymbol{X} = \{X_\ell\}_{\ell=1}^{L}$



- Let $h(\boldsymbol{X};\xi)$ be the language model; $\hat{y} = h(\boldsymbol{X};\xi)$ is the predicted token

LLM cost function:

$$\boldsymbol{X}^\star = \arg\min_{\boldsymbol{X}} \left\{ \mathbb{E}_{\xi\sim\mathcal{D}} \left[L(h(\boldsymbol{X};\xi), y) \right] \right\}$$

cross entropy

data distribution pred. token real token

LLM pretraining is essentially solving stochastic optimization

- If we define $\boldsymbol{\xi} = (\xi, y)$ and $F(\boldsymbol{X}; \boldsymbol{\xi}) = L(h(\boldsymbol{X}; \xi), y)$, the LLM problem becomes

$$\text{Stochastic optimization:} \quad \boldsymbol{X}^\star = \arg\min_{\boldsymbol{X}} \left\{ \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{D}} \left[F(\boldsymbol{X}; \boldsymbol{\xi}) \right] \right\}$$

- In other words, LLM pretraining is essentially solving a stochastic optimization problem

- Adam is the standard approach in LLM pretraining

Optimizer states

$$\boldsymbol{G}_t = \nabla F(\boldsymbol{X}_t; \boldsymbol{\xi}_t) \qquad \text{(stochastic gradient)}$$

$$\boldsymbol{M}_t = (1-\beta_1)\boldsymbol{M}_{t-1} + \beta_1 \boldsymbol{G}_t \qquad \text{(first-order momentum)}$$

$$\boldsymbol{V}_t = (1-\beta_2)\boldsymbol{V}_{t-1} + \beta_2 \boldsymbol{G}_t \odot \boldsymbol{G}_t \qquad \text{(second-order momentum)}$$

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t - \frac{\gamma}{\sqrt{\boldsymbol{V}_t} + \epsilon} \odot \boldsymbol{M}_t \qquad \text{(adaptive SGD)}$$

Memory cost to pre-train LLMs

北京大学 PEKING UNIVERSITY

Memory = **Model** + **Gradient** + **Optimizer states** + **Activations**

- Given a **model** with **P** parameters, **gradient** will consume **P** parameters, and **optimizer states** will consume **2P** parameters; **4P parameters in total.**

P $\boldsymbol{G}_t = \nabla F(\boldsymbol{X}_t; \boldsymbol{\xi}_t)$

2P
$$\begin{cases} \boldsymbol{M}_t = (1-\beta_1)\boldsymbol{M}_{t-1} + \beta_1 \boldsymbol{G}_t \\ \boldsymbol{V}_t = (1-\beta_2)\boldsymbol{V}_{t-1} + \beta_2 \boldsymbol{G}_t \odot \boldsymbol{G}_t \end{cases}$$

P $\boldsymbol{X}_{t+1} = \boldsymbol{X}_t - \frac{\gamma}{\sqrt{\boldsymbol{V}_t} + \epsilon} \odot \boldsymbol{M}_t$

Optimizer states introduces significant memory cost

Memory cost to pre-train LLMs

北京大学 PEKING UNIVERSITY

Memory = **Model** + **Gradient** + **Optimizer states** + **Activations**

- Activations are auxiliary variables to facilitate the gradient calculations

Consider a linear neural network

$$z_i = X_i z_{i-1}, \forall\, i = 1, \cdots, L$$
$$f = \mathcal{L}(z_i; y)$$

The gradient is derived as follows

$$\frac{\partial f}{\partial X_i} = \frac{\partial f}{\partial z_i} z_{i-1}^\top$$

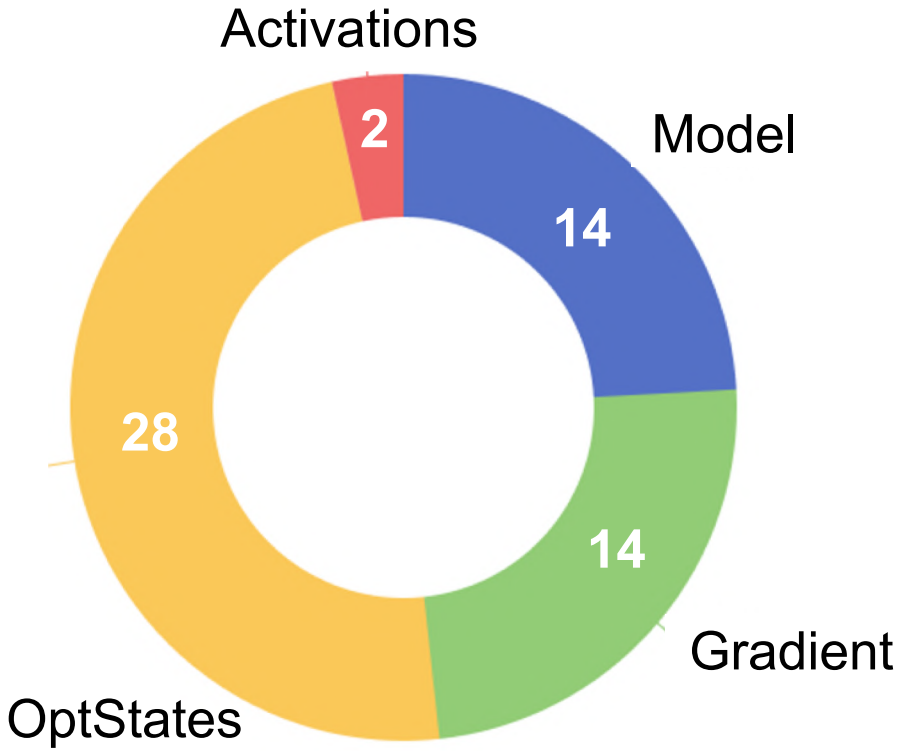
Need to store activations $z_1, z_2, \cdots, z_L$

- The size of activations depends on sequence length and batch size

Minimum memory requirement: LLaMA 7B

- Pretrain LLaMA 7B model (BF16) from scratch with **a single batch size** requires

  - Parameters: 7B
  - Model storage: 7B * 2 Bytes = 14 GB
  - Gradient storage: 14 GB
  - Optimizer states: 28 GB (using Adam)
  - Activation storage: 2 GB [Zhao et. al., 2024]
  - In total: **58 GB**



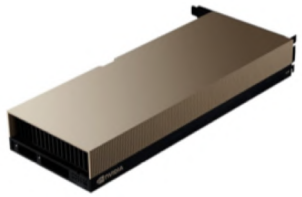
Zhao et. al., *GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection*, ICML 2024

Minimum memory requirement: LLaMA 7B

北京大学 PEKING UNIVERSITY

- Pretrain LLaMA 7B model (BF16) from scratch with **a single batch size** requires
  - Parameters: 7B
  - Model storage: 7B * 2 Bytes = 14 GB
  - Gradient storage: 14 GB
  - Optimizer states: 28 GB (using Adam)
  - Activation storage: 2 GB [Zhao et. al., 2024]
  - In total: **58 GB**

RTX 4090: 24GB



A100 80G

The minimum requirement is A100 * 1

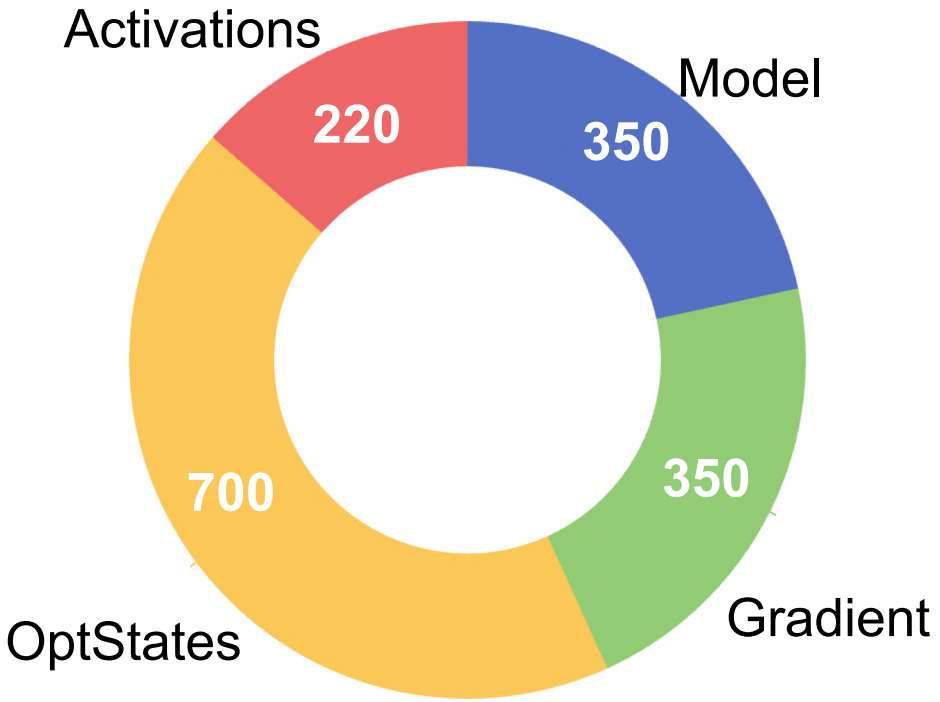
This slide is from Jiawei Zhao's talk on "Memory-efficient LLM Training"

Minimum memory requirement: GPT-3

北京大学 PEKING UNIVERSITY

- Pretrain GPT-3 model (BF16) from scratch with **a single batch size** requires

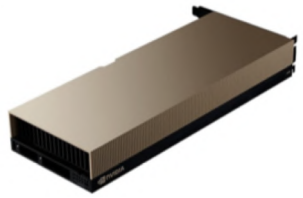
  - Parameters: 175B
  - Model storage: 175B * 2 Bytes = 350 GB
  - Gradient storage: 350 GB
  - Optimizer states: 700 GB (using Adam)
  - Activation storage: ~220 GB
  - In total: **1620 GB**



Minimum memory requirement: GPT-3

北京大学 PEKING UNIVERSITY

- Pretrain GPT-3 model (BF16) from scratch with **a single batch size** requires
  - Parameters: 175B
  - Model storage: 175B * 2 Bytes = 350 GB
  - Gradient storage: 350 GB
  - Optimizer states: 700 GB (using Adam)
  - Activation storage: ~220 GB
  - In total: **1620 GB**



A100 80G

x 21

The minimum requirement is A100 * 21

Very expensive!

Memory-efficient optimizers are in urgent need!

With memory-efficient algorithms, we can

- **Train larger models** on limited computing resources
- Use a larger training batch size to **improve throughput**

北京大学
PEKING UNIVERSITY

PART 02

GaLore: Gradient Low-Rank Projection

GaLore: Gradient Low-Rank Projection

北京大学 PEKING UNIVERSITY

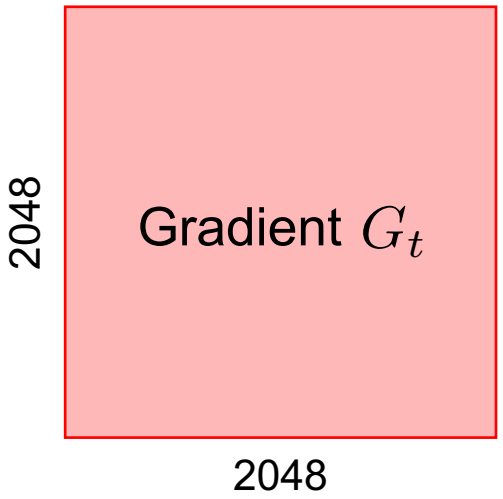
GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection

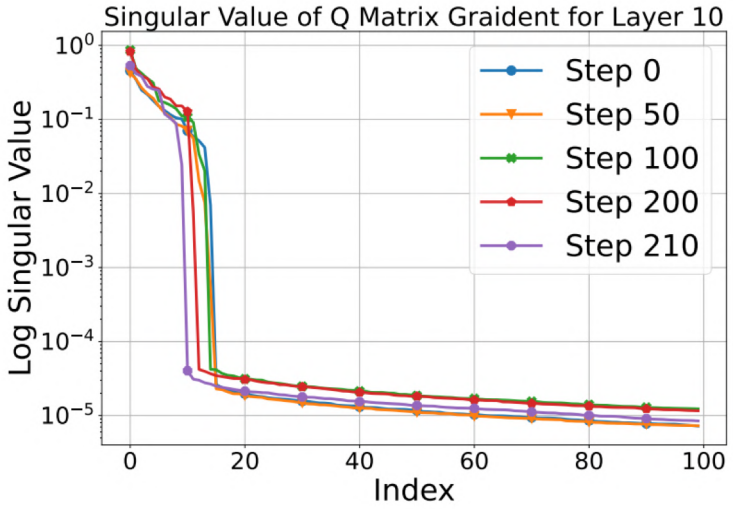
Jiawei Zhao [1] **Zhenyu Zhang** [3] **Beidi Chen** [2 4] **Zhangyang Wang** [3] **Anima Anandkumar** [* 1] **Yuandong Tian** [* 2]

- GaLore is a novel approach for reducing memory consumption from optimizer states
- Memory-efficient without severe performance degradation

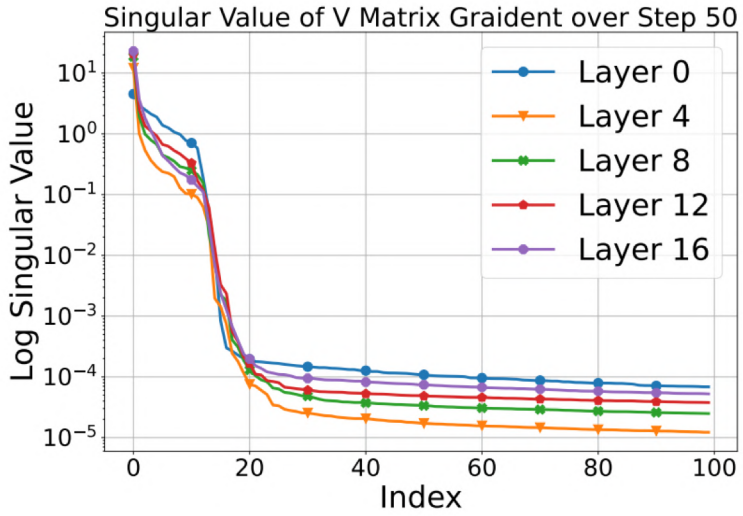
GaLore: Gradient Low-Rank Projection

- Observation: gradient in LLMs becomes **low-rank** during training





remain low-rank across iterations



remain low-rank across layers

- Given a gradient matrix with dimensions 2048 by 2048, around **top 10** eigenvalues dominate

- How to utilize the low-rank structure in gradients?

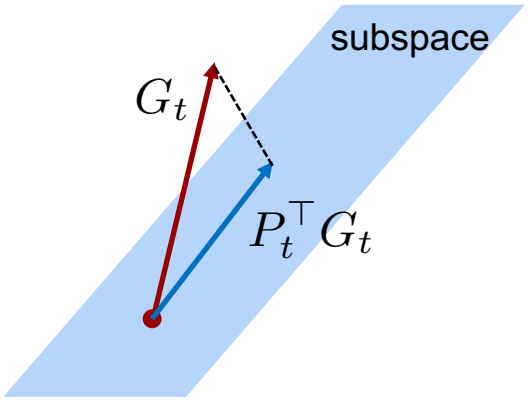
GaLore: Gradient Low-Rank Projection

- Main idea: Projecting gradient onto the low-rank subspace

- Given a gradient $G_t \in \mathbb{R}^{m \times n}$ and a projection $P_t \in \mathbb{R}^{m \times r}$, we project Gradient onto low-rank subspace

$$G_t \in \mathbb{R}^{m \times n} \quad P_t^\top G_t \quad g_t = P_t^\top G_t \in \mathbb{R}^{r \times n}$$

Low-rank projection



- Since $r \ll m$, the low-rank gradient g_t has much smaller parameters than G_t

GaLore: Gradient Low-Rank Projection

北京大学 PEKING UNIVERSITY

- Low-rank optimizer states:

$$
\begin{aligned}
\boldsymbol{g}_t &= \boldsymbol{P}_t^\top \boldsymbol{G}_t && \triangleright \text{ dims r x n} \\
\boldsymbol{m}_t &= (1-\beta_1)\boldsymbol{m}_{t-1} + \beta_1 \boldsymbol{g}_t && \triangleright \text{ dims r x n} \\
\boldsymbol{v}_t &= (1-\beta_2)\boldsymbol{v}_{t-1} + \beta_2 \boldsymbol{g}_t \odot \boldsymbol{g}_t && \triangleright \text{ dims r x n} \\
\boldsymbol{\delta}_t &= \frac{\gamma}{\sqrt{\boldsymbol{v}_t} + \epsilon} \odot \boldsymbol{m}_t && \triangleright \text{ dims r x n}
\end{aligned}
$$

- Parameter updates:

$$
\boldsymbol{X}_{t+1} = \boldsymbol{X}_t - \boldsymbol{P}_t \boldsymbol{\delta}_t \qquad \triangleright \text{ dims m x n}
$$

Simplified as

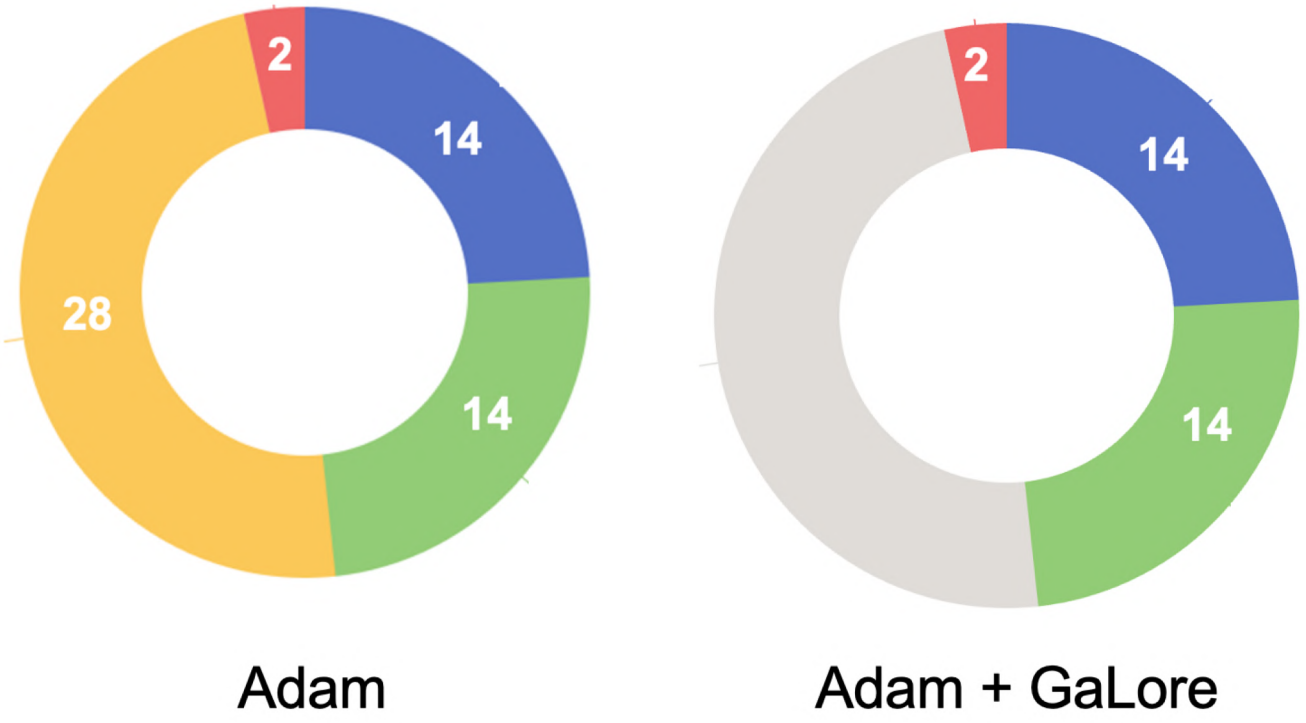
$$
\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \boldsymbol{P}_t \boldsymbol{\rho}(\boldsymbol{P}_t^\top \boldsymbol{G}_t)
$$

- Memory cost: Model $\boldsymbol{X}$, Gradient $\boldsymbol{G}$, Projection $\boldsymbol{P}$, OptStates $\boldsymbol{m}, \boldsymbol{v}$ and activations

trivial memory cost

GaLore: Gradient Low-Rank Projection

LLaMA 7B



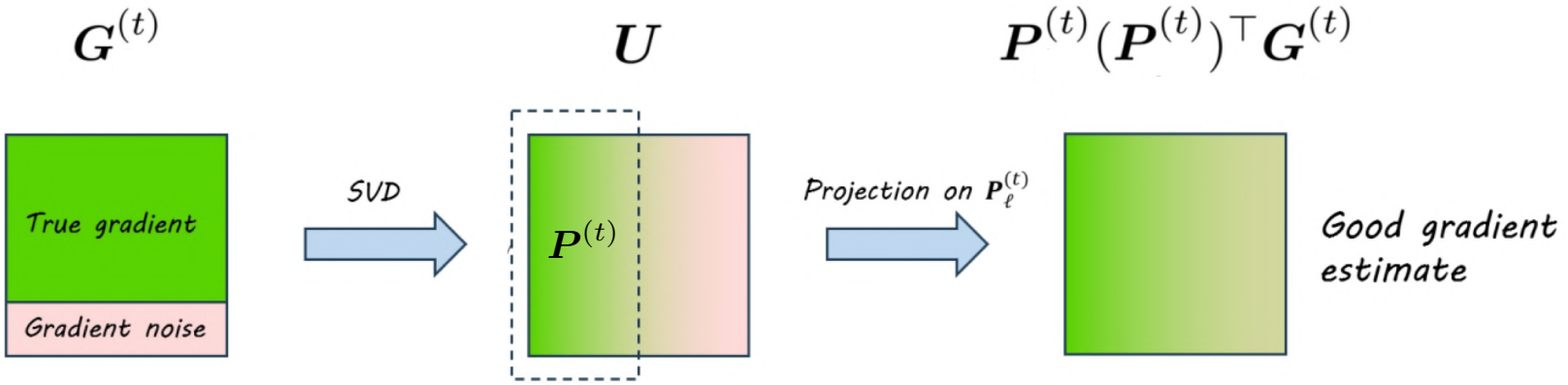
GaLore: Gradient Low-Rank Projection

- Recall the GaLore update: $\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \boldsymbol{P}_t \boldsymbol{\rho}(\boldsymbol{P}_t^\top \boldsymbol{G}_t)$

- How to find the projection matrix? **SVD decomposition!**

$$G_t = U\Sigma V^\top \longrightarrow P_t = \boxed{U[:, :r]} \in \mathbb{R}^{m \times r}$$

Select the first r columns



[Subspace Optimization for Large Language Models with Convergence Guarantees]

GaLore: Gradient Low-Rank Projection

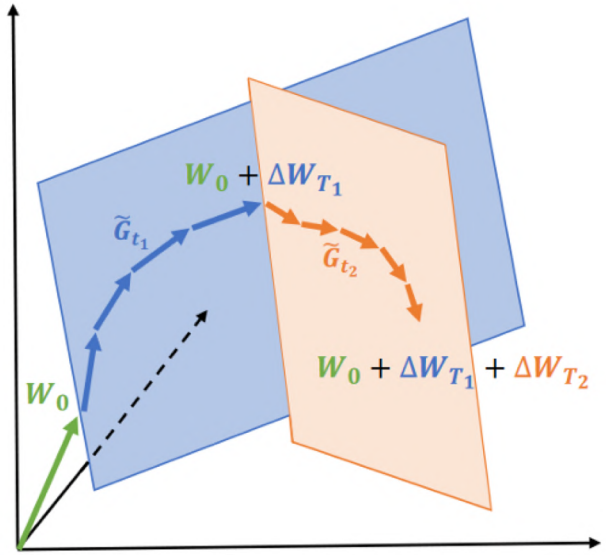
- It is computationally expensive to perform SVD in each iteration

- **Lazy SVD**: perform SVD every τ iterations

(The complete GaLore algorithm)

$$\begin{cases} \boldsymbol{P}_t \leftarrow \mathrm{SVD}(\boldsymbol{G}_t) & \text{if } t \bmod \tau = 0 \\ \boldsymbol{P}_t \leftarrow \boldsymbol{P}_{t-1} & \text{otherwise} \end{cases}$$

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \boldsymbol{P}_t \boldsymbol{\rho}(\boldsymbol{P}_t^\top \boldsymbol{G}_t)$$



[GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection]

GaLore: Gradient Low-Rank Projection

北京大学 PEKING UNIVERSITY

Pretraining LLaMA on C4 dataset

	60M	130M	350M	1B
Full-Rank	34.06 (0.36G)	25.08 (0.76G)	18.80 (2.06G)	15.56 (7.80G)
GaLore	**34.88** (0.24G)	**25.36** (0.52G)	**18.95** (1.22G)	**15.64** (4.38G)
Low-Rank	78.18 (0.26G)	45.51 (0.54G)	37.41 (1.08G)	142.53 (3.57G)
LoRA	34.99 (0.36G)	33.92 (0.80G)	25.58 (1.76G)	19.21 (6.17G)
ReLoRA	37.04 (0.36G)	29.37 (0.80G)	29.08 (1.76G)	18.33 (6.17G)
r/d_{model}	128 / 256	256 / 768	256 / 1024	512 / 2048
Training Tokens	1.1B	2.2B	6.4B	13.1B

GaLore looks great

But does GaLore provably converge to the desired solution?

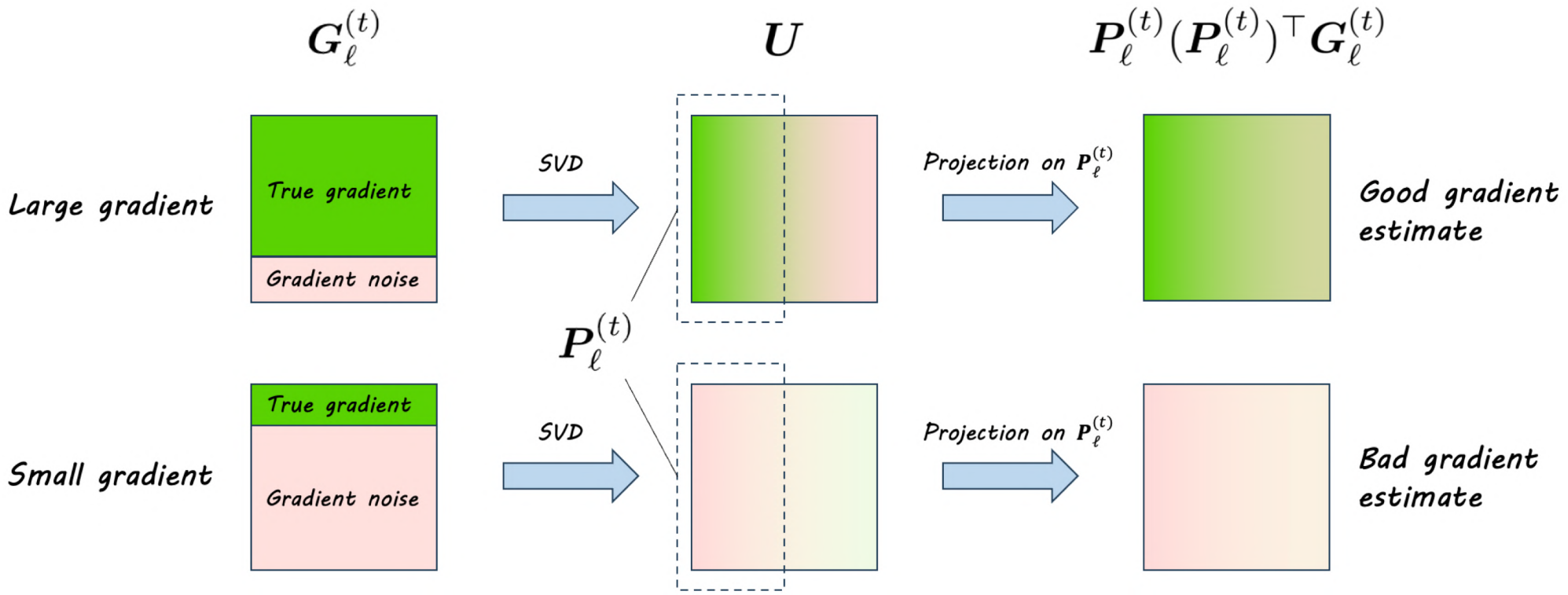
Not Necessarily True!

Y. He, P. Li, Y. Hu, C. Chen, and K. Yuan, *Subspace Optimization for Large Language Models with Convergence Guarantees,* ICML 2025

PART 03

GaLore is NOT GUARANTEED to converge

Intuition behind GaLore's non-convergence

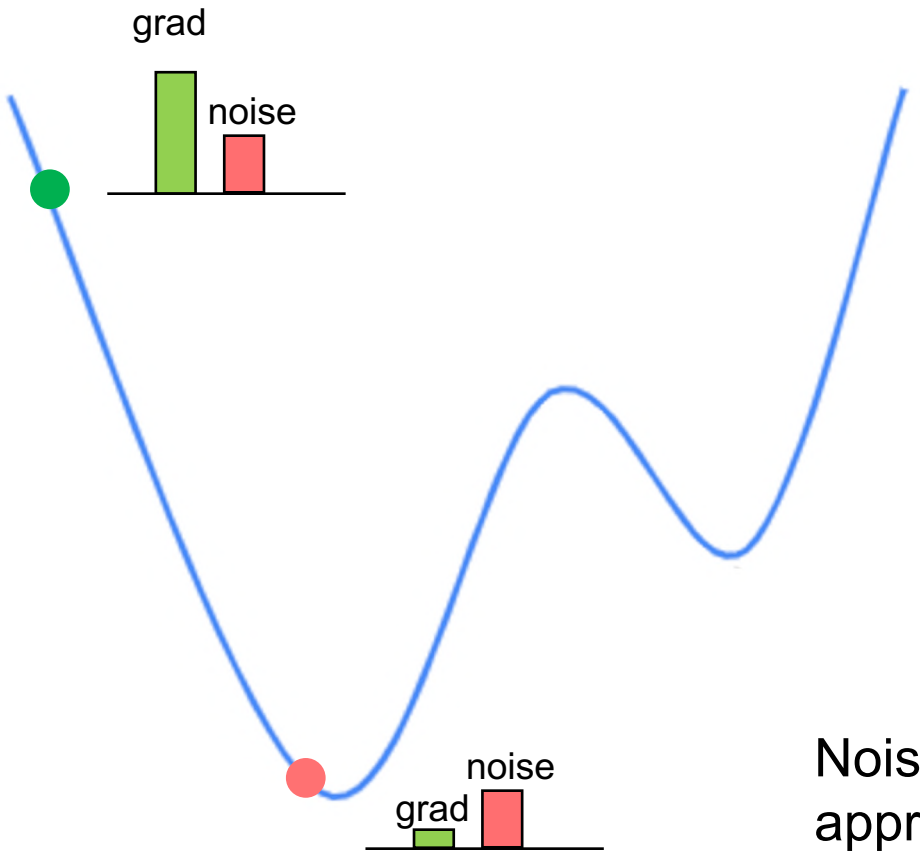


When gradient noise dominates the stochastic gradient, SVD captures **noise-dominated** subspace!

All gradient information is lost !

Is noise-dominant scenario common? Yes!

Gradient dominates during the initial stages



Noise dominates when approaching the local minimum

SVD 投影的理论缺陷：反例构造

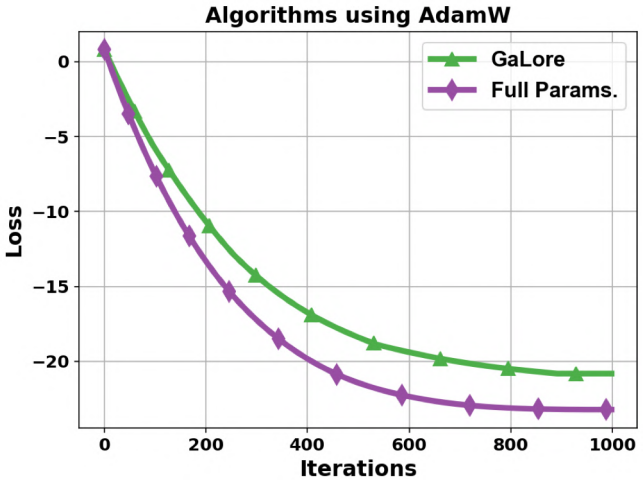
Counter-Example. We consider the following quadratic problem with gradient noise:

$$f(\boldsymbol{X}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{X}\|_F^2 + \langle \boldsymbol{B}, \boldsymbol{X}\rangle_F, \quad \nabla F(\boldsymbol{X};\xi) = \nabla f(\boldsymbol{X}) + \xi\sigma\boldsymbol{C}, \tag{1}$$

where $\boldsymbol{A} = (\boldsymbol{I}_{n-r} \quad 0) \in \mathbb{R}^{(n-r)\times n}$, $\boldsymbol{B} = \begin{pmatrix} \boldsymbol{D} & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{n\times n}$ with $\boldsymbol{D} \in \mathbb{R}^{(n-r)\times(n-r)}$ generated randomly, $\boldsymbol{C} = \begin{pmatrix} 0 & 0 \\ 0 & \boldsymbol{I}_r \end{pmatrix} \in \mathbb{R}^{n\times n}$, ξ is a random variable uniformly sampled from $\{1, -1\}$ per iteration, and σ is used to control the gradier oblem

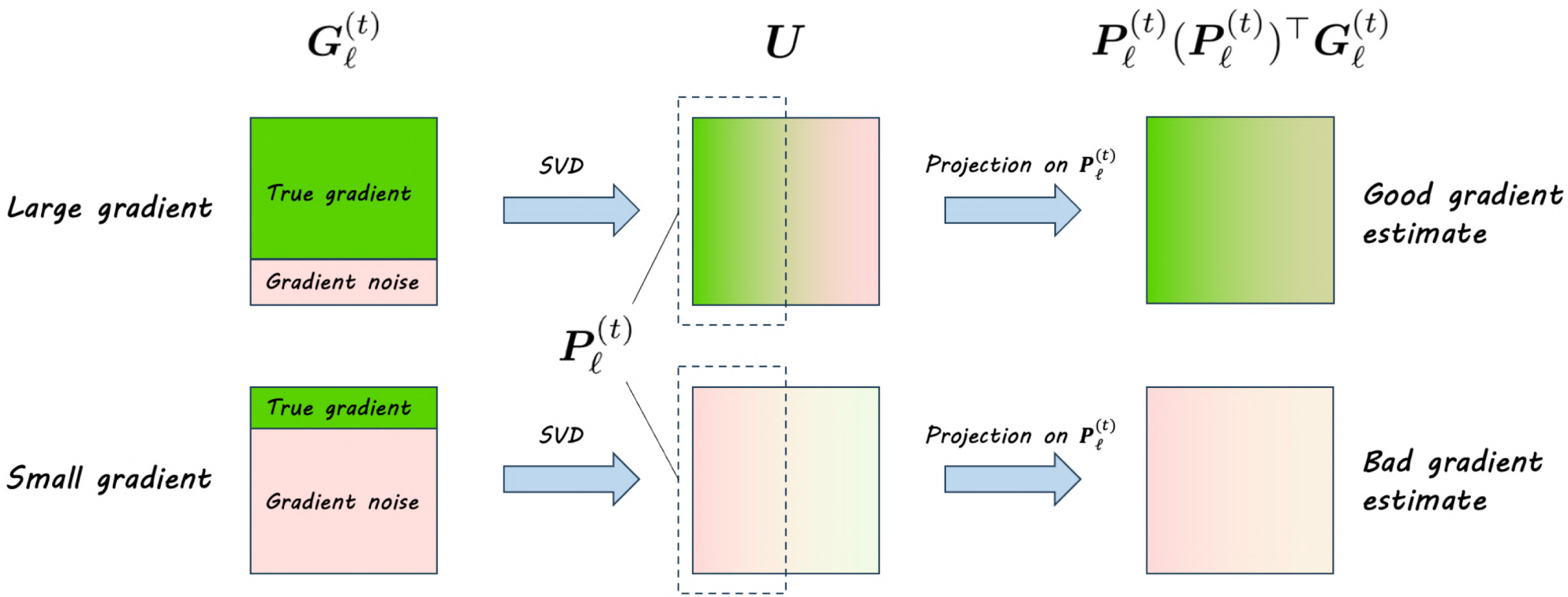
Theorem (Non-convergence of GaLore): There exists an objective function $f : \mathbb{R}^d \to \mathbb{R}$ satisfying Assumptions 1 and 2, a stochastic gradient oracle (F, D) satisfying Assumption 3, an initial point $\boldsymbol{x}^{(0)} \in \mathbb{R}^d$, and a constant $\epsilon_0 > 0$ such that for any rank $r_\ell < \min\{m_\ell, n_\ell\}$, subspace changing frequency τ, any optimizer ρ that inputs a subspace gradient of shape $r_\ell \times n_\ell$ and outputs a subspace update direction of the same shape, and for any $t > 0$, it holds that

$$\|\nabla f(\boldsymbol{x}^{(t)})\|_2^2 \geq \epsilon_0.$$



GaLore does **NOT** converge to desired solutions

Under what conditions can GaLore converge?



GaLore can converge if we can avoid the noise-dominant scenarios

Condition I: noise-free

- Consider GaLore with deterministic gradient: $G_\ell^{(t)} = \nabla_\ell f(\mathbf{x}^{(t)})$

Theorem 2 (Convergence rate of deterministic GaLore). *Under Assumptions 1-2, if the number of iterations $T \geq 64/(3\underline{\delta})$ and we choose*

$$\beta_1 = 1, \quad \tau = \left\lceil \frac{64}{3\underline{\delta}\beta_1} \right\rceil, \quad \text{and} \quad \eta = \left(4L + \sqrt{\frac{80L^2}{3\underline{\delta}\beta_1^2}} + \sqrt{\frac{80\tau^2 L^2}{3\underline{\delta}}} + \sqrt{\frac{16\tau L^2}{3\beta_1}} \right)^{-1},$$

GaLore using deterministic gradients and MSGD with MP converges as

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}[\|\nabla f(\boldsymbol{x}^{(t)})\|_2^2] = \mathcal{O}\left(\frac{L\Delta}{\underline{\delta}^{5/2} T} \right),$$

where $\Delta = f(\boldsymbol{x}^{(0)}) - \inf_{\boldsymbol{x}} f(\boldsymbol{x})$ and $\underline{\delta} := \min_\ell \frac{r_\ell}{\min\{m_\ell, n_\ell\}}$.

- Noise-free GaLore **converges** at rate $\mathcal{O}(1/T)$.

Condition II: Large batch-size

- Consider GaLore with large-batch stochastic gradient: $G_\ell^{(t)} = \frac{1}{\mathcal{B}} \sum_{b=1}^{\mathcal{B}} \nabla_\ell F(\mathbf{x}^{(t)}; \xi^{(t,b)})$
- Batch-size $\mathcal{B}$ increases with iteration T, e.g., $\mathcal{B} = \mathcal{O}(\sqrt{T})$

Theorem 3 (Convergence rate of large-batch GaLore). *Under Assumptions 1-3, if* $T \geq 2 + 128/(3\underline{\delta}) + (128\sigma)^2/(9\sqrt{\underline{\delta}}L\Delta)$ *and we choose* $\tau = \lceil 64/(3\underline{\delta}\beta_1) \rceil$, $\mathcal{B} = \lceil 1/(\underline{\delta}\beta_1) \rceil$,

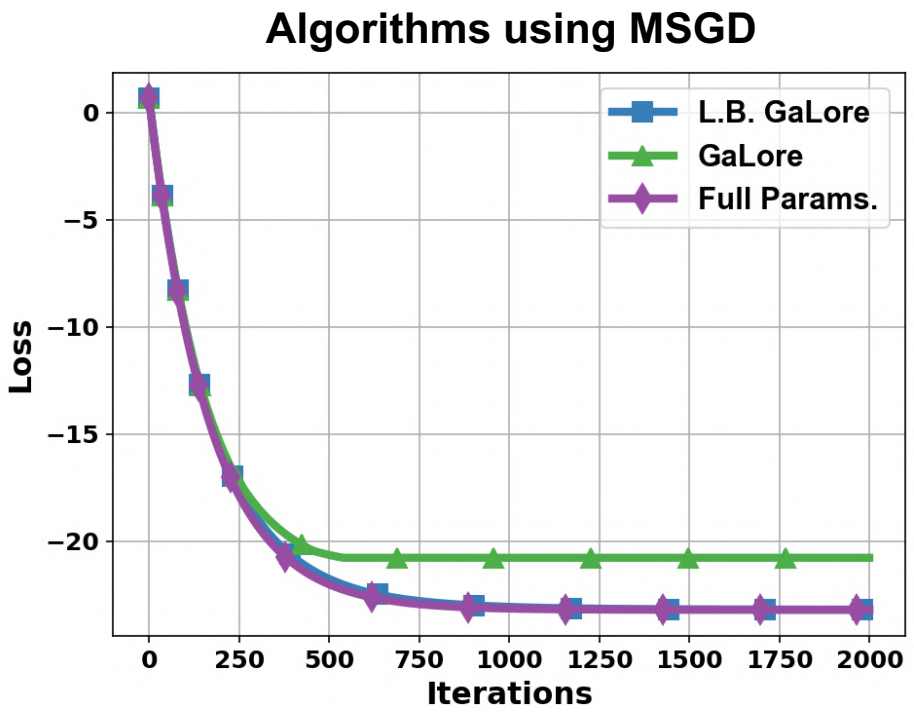
$$\beta_1 = \left(1 + \sqrt{\frac{\underline{\delta}^{3/2}\sigma^2 T}{L\Delta}}\right)^{-1}, \quad \text{and} \quad \eta = \left(4L + \sqrt{\frac{80L^2}{3\underline{\delta}\beta_1^2}} + \sqrt{\frac{40\tau^2 L^2}{\underline{\delta}}} + \sqrt{\frac{32\tau L^2}{3\beta_1}}\right)^{-1},$$

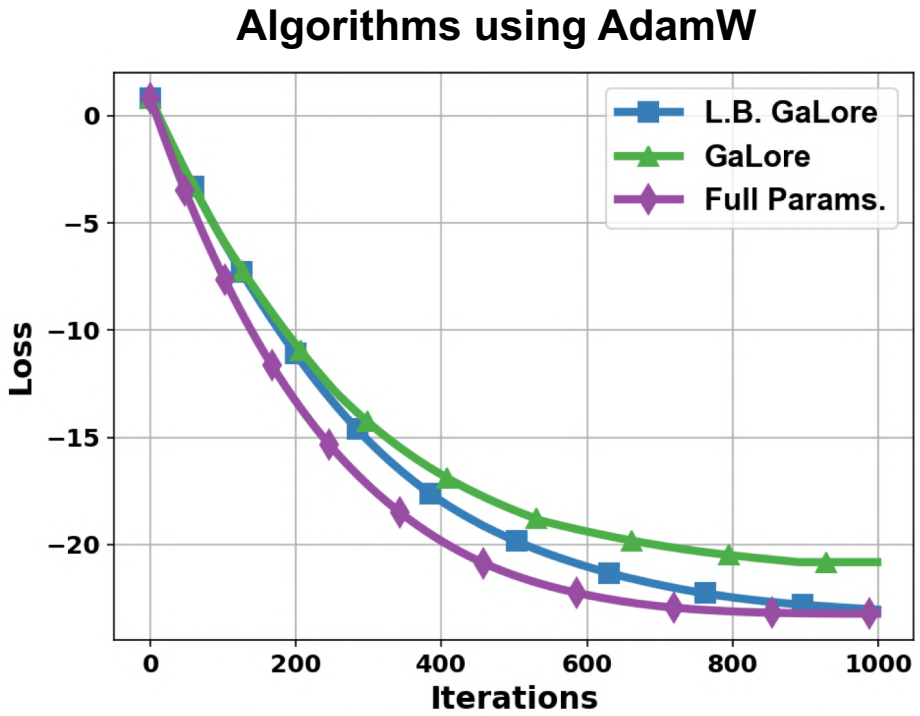
GaLore using large-batch gradients and MSGD with MP converges as

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}[\|\nabla f(\boldsymbol{x}^{(t)})\|_2^2] = \mathcal{O}\left(\frac{L\Delta}{\underline{\delta}^{5/2}T} + \sqrt{\frac{L\Delta\sigma^2}{\underline{\delta}^{7/2}T}}\right),$$

where $\Delta = f(\boldsymbol{x}^{(0)}) - \inf_{\boldsymbol{x}} f(\boldsymbol{x})$ *and* $\underline{\delta} := \min_\ell \frac{r_\ell}{\min\{m_\ell, n_\ell\}}$.

Numerical simulations





However, neither noise-free nor large-batch is practical for LLMs settings

PART 04

GoLore: Gradient RANDOM Low-rank projection

GoLore intuition

- In LLM settings, gradient noise exists and batch-size does not increase with iterations
- The root reason that GaLore has convergence issues is the SVD-incurred subspace
- Random projection can possibly capture gradient information when noise dominates

(Stiefel manifold) $\mathrm{St}_{m,r} = \{\boldsymbol{P} \in \mathbb{R}^{m\times r} \mid \boldsymbol{P}^\top \boldsymbol{P} = I_r\}.$

Proposition 1 (Chikuse (2012), Theorem 2.2.1)**.** *A random matrix* $\boldsymbol{X}$ *uniformly distributed on* $\mathrm{St}_{m,r}$ *is expressed as* $\boldsymbol{X} = \boldsymbol{Z}(\boldsymbol{Z}^\top \boldsymbol{Z})^{-1/2}$*, where the elements of an* $m \times r$ *random matrix* $\boldsymbol{Z}$ *are independent and identically distributed as normal* $\mathcal{N}(0, 1)$.

Following Prop. 1, we can sample random projections from Stiefel manifold

GoLore:基于随机均匀投影的子空间训练算法

北京大学 PEKING UNIVERSITY

- Gradient random Low-rank projection **(GoLore)** 将梯度投影至随机

$$P_t \sim \mathcal{U}(\mathrm{St}_{m,r})$$

Lemma 5 (Error of GoLore's projection). *Let* $\boldsymbol{P} \sim \mathcal{U}(\mathrm{St}_{m,r})$, $\boldsymbol{Q} \sim \mathcal{U}(\mathrm{St}_{n,r})$, *it holds for all* $\boldsymbol{G} \in \mathbb{R}^{m\times n}$ *that*

$$\mathbb{E}[\boldsymbol{P}\boldsymbol{P}^\top] = \frac{r}{m}\cdot \boldsymbol{I}, \quad \mathbb{E}[\boldsymbol{Q}\boldsymbol{Q}^\top] = \frac{r}{n}\cdot \boldsymbol{I},$$

and

$$\mathbb{E}[\|\boldsymbol{P}\boldsymbol{P}^\top \boldsymbol{G} - \boldsymbol{G}\|_F^2] = \left(1-\frac{r}{m}\right)\|\boldsymbol{G}\|_F^2, \quad \mathbb{E}[\|\boldsymbol{G}\boldsymbol{Q}\boldsymbol{Q}^\top - \boldsymbol{G}\|_F^2] = \left(1-\frac{r}{n}\right)\|\boldsymbol{G}\|_F^2.$$

$$\mathbb{E}[PP^\top G] = \mathbb{E}[PP^\top]\cdot\mathbb{E}[G] = \frac{r}{m}\nabla F(X)$$

The low-rank randomized projected gradient is an unbiased estimate of the true gradient

GoLore converges to desired solutions

Theorem 4 (Convergence rate of GoLore). *Under Assumptions 1-3, for any* $T \geq 2 + 128/(3\underline{\delta}) + (128\sigma)^2/(9\sqrt{\underline{\delta}}L\Delta)$, *if we choose* $\tau = \lceil 64/(3\underline{\delta}\beta_1) \rceil$,

$$\beta_1 = \left(1 + \sqrt{\frac{\underline{\delta}^{3/2}\sigma^2 T}{L\Delta}}\right)^{-1}, \quad \text{and} \quad \eta = \left(4L + \sqrt{\frac{80L^2}{3\underline{\delta}\beta_1^2}} + \sqrt{\frac{80\tau^2 L^2}{3\underline{\delta}}} + \sqrt{\frac{16\tau L^2}{3\beta_1}}\right)^{-1},$$

GoLore using small-batch stochastic gradients and MSGD with MP converges as

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[\|\nabla f(\boldsymbol{x}^{(t)})\|_2^2] = \mathcal{O}\left(\frac{L\Delta}{\underline{\delta}^{5/2}T} + \sqrt{\frac{L\Delta\sigma^2}{\underline{\delta}^{7/2}T}}\right),$$

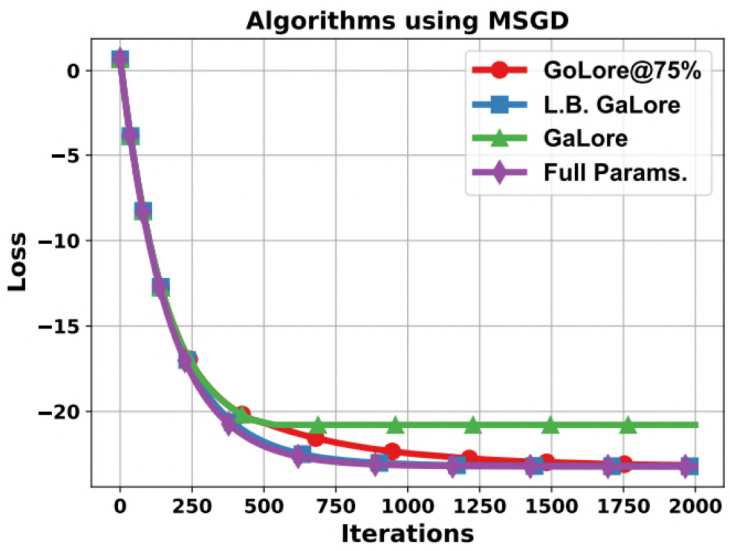
where $\Delta = f(\boldsymbol{x}^{(0)}) - \inf_{\boldsymbol{x}} f(\boldsymbol{x})$ *and* $\underline{\delta} := \min_\ell \frac{r_\ell}{\min\{m_\ell, n_\ell\}}$.

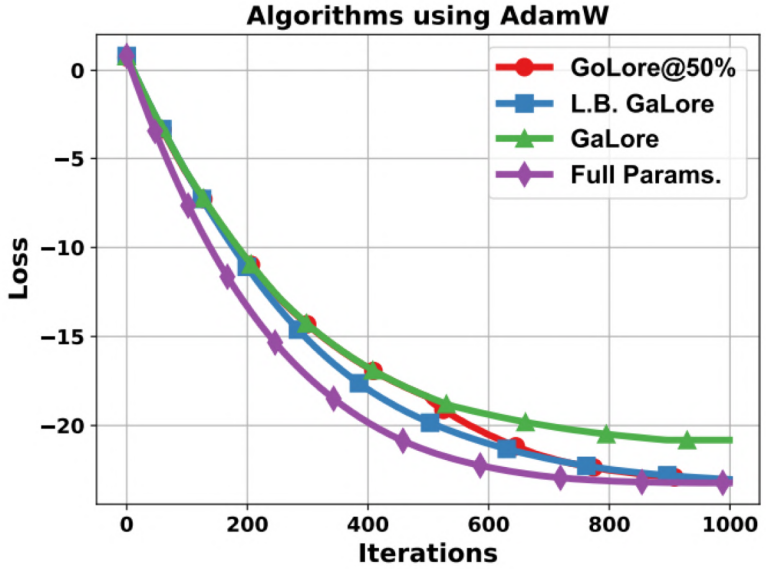
- Theoretically, GoLore **converges** at rate $\mathcal{O}(1/\sqrt{T})$.

A hybrid strategy: GaLore + GoLore

- SVD projection is preferred in initial stages: effectively capture gradient information
- Random projection is preferred when approaching solutions: avoid losing gradient information

GoLore@x% = GaLore (first (100-x)% iters) + GoLore (last x% iters)

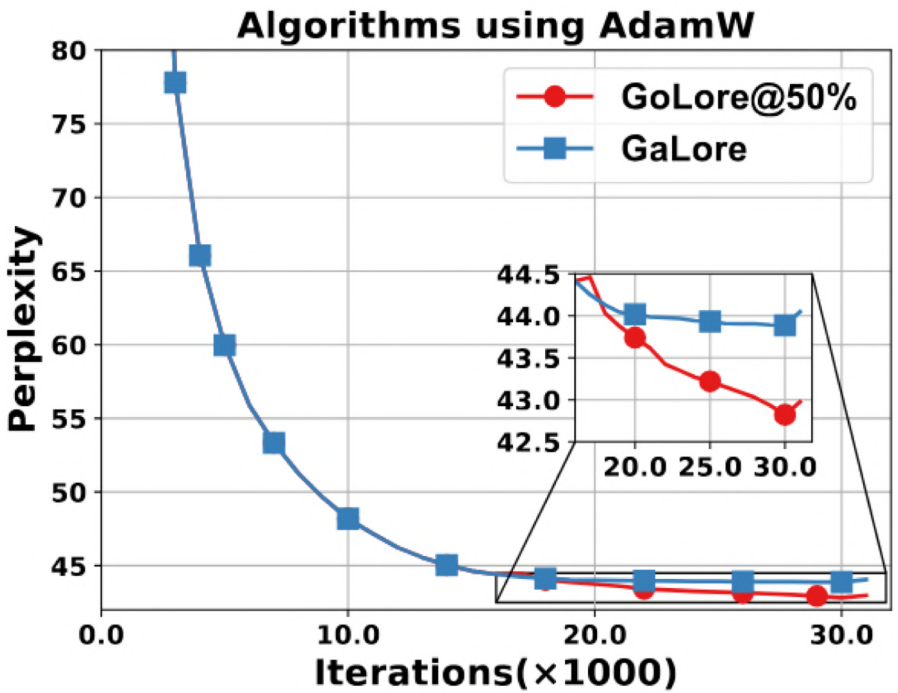




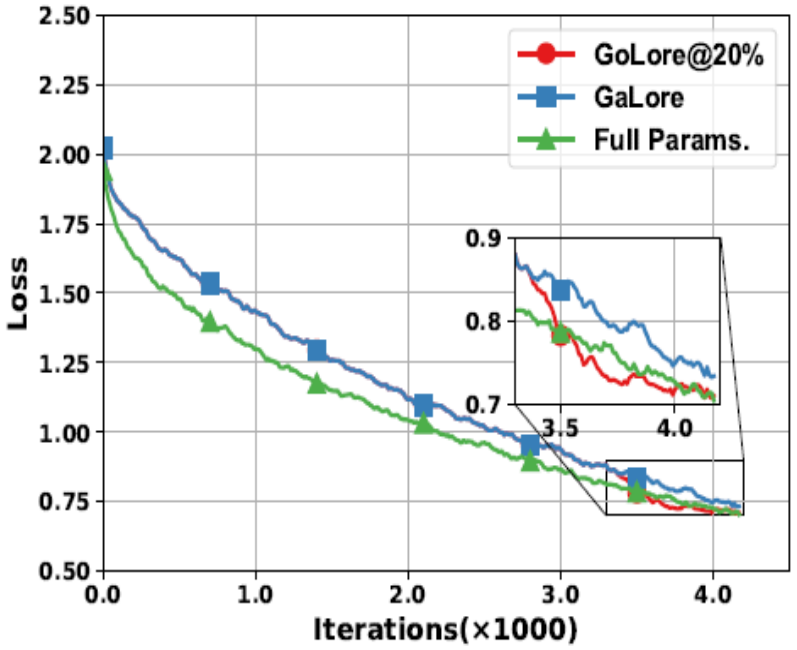
Experimental results

北京大学 PEKING UNIVERSITY

Pre-train LLaM2-60M on C4



Fine-tuning LLaMA2-7B on WinoGrande:



Experimental results

北京大学 PEKING UNIVERSITY

- Fine-tuning RoBERTa-BASE on GLUE benchmark:

Algorithm	CoLA	STS-B	MRPC	RTE	SST2	MNLI	QNLI	QQP	Avg
Full Params.	62.07	90.18	92.25	78.34	94.38	87.59	92.46	91.90	86.15
GaLore	61.32	90.24	92.55	77.62	**94.61**	86.92	92.06	90.84	85.77
GoLore@20%	**61.66**	**90.55**	**92.93**	**78.34**	**94.61**	**87.02**	**92.20**	**90.91**	**86.03**

- GoLore shows **superior** performance than GaLore in the above experiments.

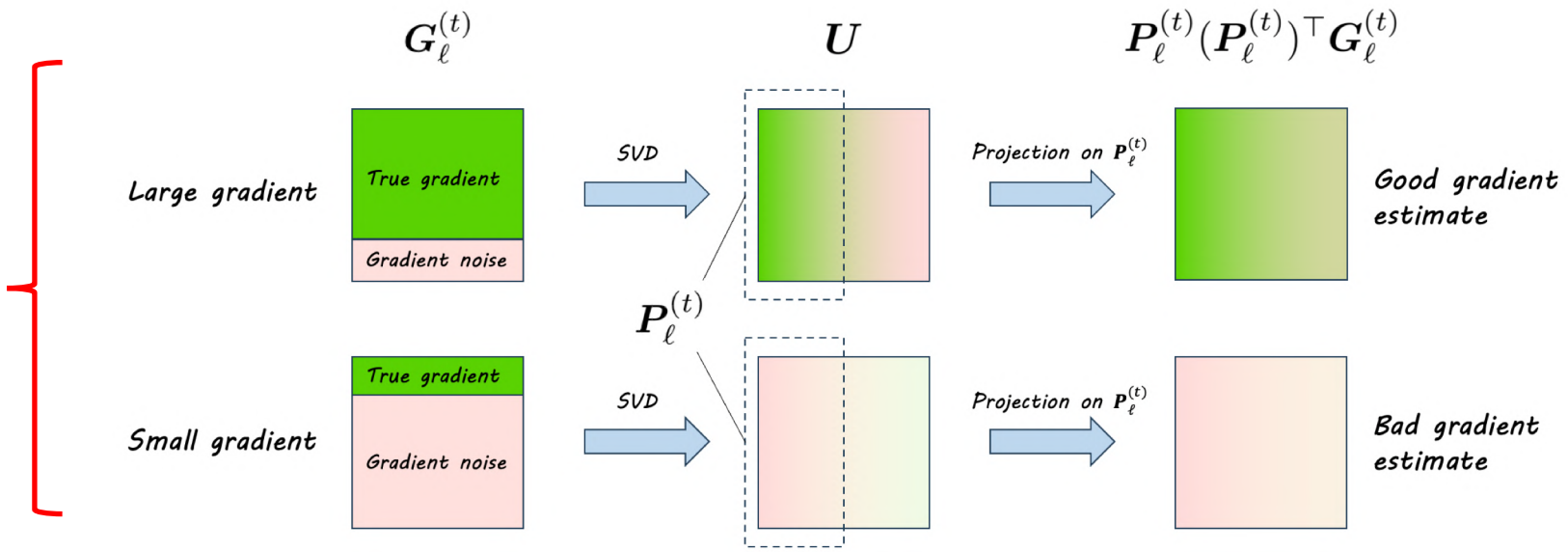
Summary

- Gradient low-rank projection can effectively save optimizer states
- GaLore cannot converge to desired solutions due to SVD projections
- Random projections enable GaLore to converge

paper

Github

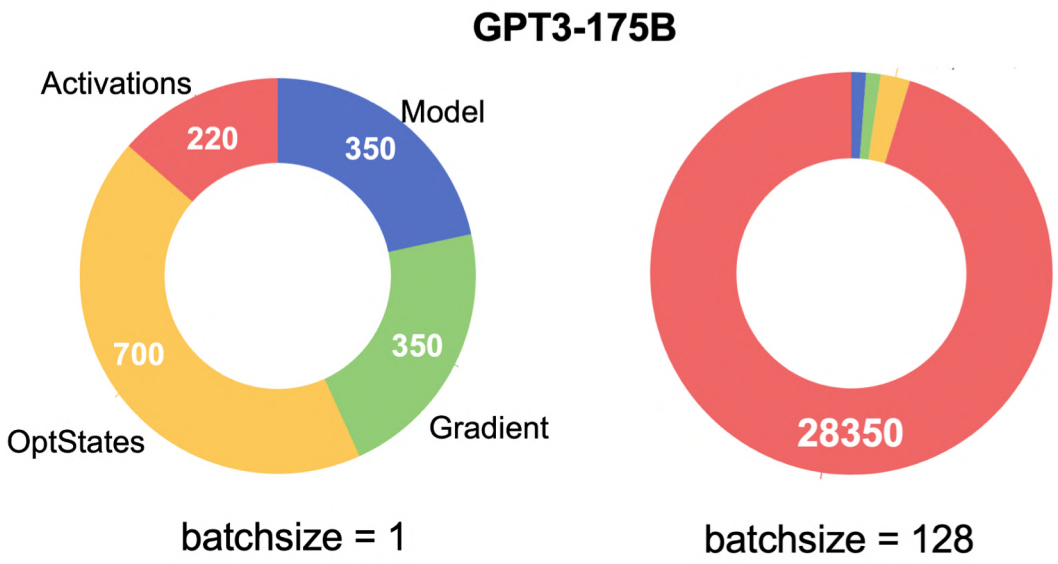
Take-home message



Limitations in GaLore/GoLore

Memory = **Model** + **Gradient** + **Optimizer states** + **Activations**

- When batch-size is large, activations will dominate the memory



GaLore cannot save activations

$$\boldsymbol{g}_t = \boldsymbol{P}_t^\top \boldsymbol{G}_t$$
$$\boldsymbol{m}_t = (1-\beta_1)\boldsymbol{m}_{t-1} + \beta_1 \boldsymbol{g}_t$$
$$\boldsymbol{v}_t = (1-\beta_2)\boldsymbol{v}_{t-1} + \beta_2 \boldsymbol{g}_t \odot \boldsymbol{g}_t$$
$$\boldsymbol{\delta}_t = \frac{\gamma}{\sqrt{\boldsymbol{v}_t} + \epsilon} \odot \boldsymbol{m}_t$$

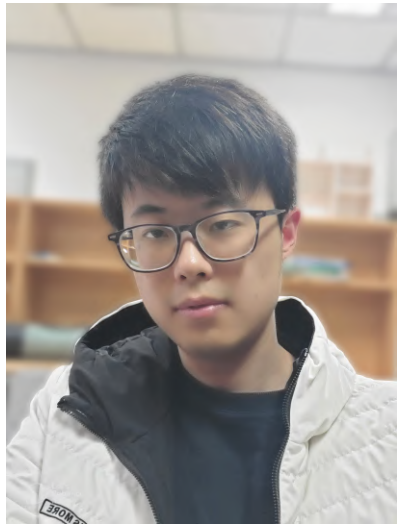
北京大学
PEKING UNIVERSITY

A Memory Efficient Randomized Subspace Optimization Method for Training Large Language Models

Kun Yuan (袁 坤)

Center for Machine Learning Research @ Peking University

Joint work with



Yiming Chen
(PKU)

Yuan Zhang
(PKU)

Yin Liu
(PKU)

Zaiwen Wen
(PKU)

Y. Chen, Y. Zhang, Y. Liu, **K. Yuan***, Z. Wen, *A Memory Efficient Randomized Subspace Optimization Method for Training Large Language Models,* ICML 2025

北京大学
PEKING UNIVERSITY

PART 01

Randomized Subspace Optimization

Randomized Subspace Optimization (RSO)

- Consider the optimization problem for the LLM training:

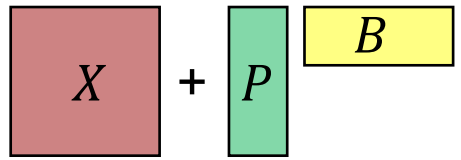
$$\min_{\boldsymbol{X} \in \mathbb{R}^{m \times n}} f(\boldsymbol{X}) := \mathbb{E}_{\boldsymbol{\xi}}[F(\boldsymbol{X}; \boldsymbol{\xi})].$$

- We solve small problems sequentially; optimizing over **one subspace** (spanned by P) at a time:

Sample P for each iteration k

$$\tilde{\boldsymbol{B}}^k \approx \operatorname*{argmin}_{\boldsymbol{B} \in \mathbb{R}^{r \times n}} \left\{ f(\boldsymbol{X}^k + \boldsymbol{P}^k \boldsymbol{B}) + \frac{1}{2\eta^k} \|\boldsymbol{B}\|^2 \right\},$$

$$\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k \tilde{\boldsymbol{B}}^k.$$



Here $X \in R^{m \times n}$, $P \in R^{m \times r}$ and $B \in R^{r \times n}$. Dimension $r \ll \min\{m, n\}$. The new variable B has much fewer parameters than X

Randomized Subspace Optimization (RSO)

北京大学 PEKING UNIVERSITY

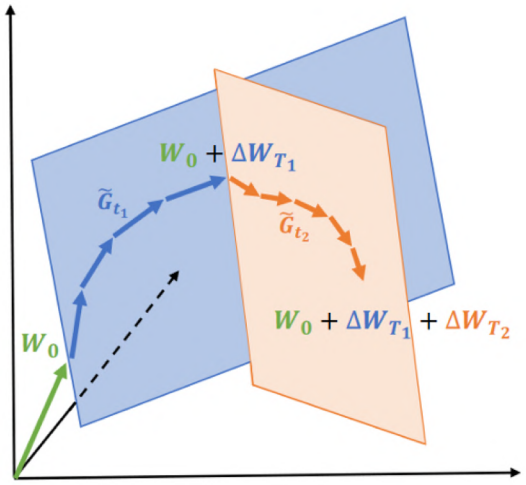
Can be solved with any optimizer

Promote strong convexity

$$\tilde{\boldsymbol{B}}^k \approx \underset{\boldsymbol{B}\in\mathbb{R}^{r\times n}}{\operatorname{argmin}} \left\{ f(\boldsymbol{X}^k + \boldsymbol{P}^k \boldsymbol{B}) + \frac{1}{2\eta^k}\|\boldsymbol{B}\|^2 \right\}, \tag{4a}$$

$$\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k \tilde{\boldsymbol{B}}^k. \tag{4b}$$

(Zhao et. al. ICML 2023)



Algorithm 1: Randomized Subspace Optimization

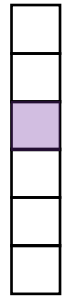
Input: Initialization $\boldsymbol{X}^0$
Output: Solution $\boldsymbol{X}^K$
for $k = 0, 1, \ldots, K-1$ **do**
  Sample $\boldsymbol{P}^k$ according to a given distribution;
  Solve subproblem (4a) and obtain the approximate solution $\tilde{\boldsymbol{B}}^k$ using a given optimizer such as Adam;
  Update the weights by $\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k \tilde{\boldsymbol{B}}^k$;

RSO is essentially a generalized method of coordinate minimization

- Recall the random **coordinate** minimization:

$$b_k^\star = \arg\min_{b\in\mathbb{R}} \{f(x^k + b\cdot e_{i_k})\}, \text{ where } i_k \sim \mathcal{U}\{1,\dots,d\},$$
$$x^{k+1} = x^k + b_k^\star \cdot e_{i_k},$$

b is the increment in the i-th coordinate

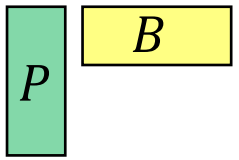


where the variable x is a vector in the above method

- Random subspace optimization can be regarded as a generalized method; not in the coordinate subspace but in the low-rank subspace

$$\tilde{\boldsymbol{B}}^k \approx \underset{\boldsymbol{B}\in\mathbb{R}^{r\times n}}{\operatorname{argmin}} \left\{ f(\boldsymbol{X}^k + \boldsymbol{P}^k\boldsymbol{B}) + \frac{1}{2\eta^k}\|\boldsymbol{B}\|^2 \right\},$$
$$\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k\tilde{\boldsymbol{B}}^k.$$

B is the increment in the subspace spanned by P^k



RSO saves gradient and optimizer state

Memory = **Model** + **Gradient** + **Optimizer states** + **Activations**

$$\tilde{\boldsymbol{B}}^k \approx \underset{\boldsymbol{B}\in\mathbb{R}^{r\times n}}{\operatorname{argmin}} \left\{ f(\boldsymbol{X}^k + \boldsymbol{P}^k\boldsymbol{B}) + \frac{1}{2\eta^k}\|\boldsymbol{B}\|^2 \right\},$$
$$\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k\tilde{\boldsymbol{B}}^k.$$

Let $\boldsymbol{h}(\boldsymbol{B}) = f(\boldsymbol{X}^k + \boldsymbol{P}^k\boldsymbol{B}) + \frac{1}{2\eta_k}\|\boldsymbol{B}\|_F^2$

We solve $\tilde{\boldsymbol{B}}^k \approx \arg\min_{\boldsymbol{B}}\{h(\boldsymbol{B})\}$ using Adam

For $t = 1, 2, \cdots, \tau$ — Adam for solving $h(B)$

$$\boldsymbol{g}_t = \nabla_{\boldsymbol{B}} h(\boldsymbol{B}_t)$$
$$\boldsymbol{m}_t = (1-\beta_1)\boldsymbol{m}_{t-1} + \beta_1\boldsymbol{g}_t$$
$$\boldsymbol{v}_t = (1-\beta_2)\boldsymbol{v}_{t-1} + \beta_2\boldsymbol{g}_t \odot \boldsymbol{g}_t$$
$$\boldsymbol{B}_{t+1} = \boldsymbol{B}_t - \frac{\gamma}{\sqrt{\boldsymbol{v}_t + \epsilon}} \odot \boldsymbol{m}_t$$

RSO saves gradient and optimizer state

$$h(B) = f(X^k + P^k B) + \frac{1}{2\eta_k}\|B\|_F^2$$

Memory = **Model** + **Gradient** + **Optimizer states** + **Activations**

Adam for solving $f(X)$

$$G_t = \nabla F(X_t; \xi_t) \quad \text{(m, n)}$$
$$M_t = (1-\beta_1)M_{t-1} + \beta_1 G_t \quad \text{(m, n)}$$
$$V_t = (1-\beta_2)V_{t-1} + \beta_2 G_t \odot G_t \quad \text{(m, n)}$$
$$X_{t+1} = X_t - \frac{\gamma}{\sqrt{V_t + \epsilon}} \odot M_t \quad \text{(m, n)}$$

Adam for solving $h(B)$

$$g_t = \nabla_B h(B_t) \quad \text{(r, n)}$$
$$m_t = (1-\beta_1)m_{t-1} + \beta_1 g_t \quad \text{(r, n)}$$
$$v_t = (1-\beta_2)v_{t-1} + \beta_2 g_t \odot g_t \quad \text{(r, n)}$$
$$B_{t+1} = B_t - \frac{\gamma}{\sqrt{v_t + \epsilon}} \odot m_t \quad \text{(r, n)}$$

- Reduce gradient and optimizer states from mn parameters to rn parameters;
- Incurs **tiny** gradient and optimizer when $r \ll m$

RSO saves activation memory

Memory = **Model** + **Gradient** + **Optimizer states** + **Activations**

Forward: $Z = YX,\ y = L(Z).$

Backward: $\frac{\partial y}{\partial X} = Y^\top \frac{\partial y}{\partial Z}$

Adam stores (s by m) activations

Forward: $Z = Y(X + PB),\ y = L(Z).$

Backward: $\frac{\partial y}{\partial B} = (YP)^\top \frac{\partial y}{\partial Z}$

RSO stores (s by r) activations

- Reduce activations from sm parameters to sr parameters;
- Incurs **tiny** activations when $r \ll m$

Memory comparison with existing optimizer

s: #tokens in a sequence n: hidden dimension b: batch size r: rank

Algorithm	Memory		
	Optimizer States	Activations	**Gradients**
RSO	$24nr$	$8bsn + 4bsr + 2bs^2$	$12nr$
GaLore	$24nr$	$15bsn + 2bs^2$	$12n^2$
LoRA	$48nr$	$15bsn + 2bs^2$	$24nr$
Adam	$24n^2$	$15bsn + 2bs^2$	$12n^2$

$$\tilde{\boldsymbol{B}}^k \approx \underset{\boldsymbol{B}\in\mathbb{R}^{r\times n}}{\operatorname{argmin}} \left\{ f(\boldsymbol{X}^k + \boldsymbol{P}^k\boldsymbol{B}) + \frac{1}{2\eta^k}\|\boldsymbol{B}\|^2 \right\},$$

$$\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k\tilde{\boldsymbol{B}}^k.$$

- Saves gradient, optimizer states, and activations simultaneously
- RSO is the most memory-efficient optimizer to our best knowledge

北京大学
PEKING UNIVERSITY

PART 02

Convergence Guarantees

Assumptions

Definition 5.1 (Expected ϵ-inexact solution). *A solution $\tilde{\boldsymbol{B}}^k$ is said to be an expected ϵ-inexact solution if it satisfies:*

$$\mathbb{E}[g^k(\tilde{\boldsymbol{B}}^k)] - g^k(\boldsymbol{B}_\star^k) \leq \epsilon,$$

where $g^k(\boldsymbol{B}) := f(\boldsymbol{X}^k + \boldsymbol{P}^k\boldsymbol{B}) + \frac{1}{2\eta^k}\|\boldsymbol{B}\|^2$, and $\boldsymbol{B}_\star^k$ is the optimal solution define as $\boldsymbol{B}_\star^k := \arg\min_{\boldsymbol{B}} g^k(\boldsymbol{B})$.

Assumption 5.2. *The objective function $f(\boldsymbol{W})$ is L-smooth, i.e., it holds for any $\boldsymbol{W}^1$ and $\boldsymbol{W}^2$ that*

$$\|\nabla f(\boldsymbol{W}^1) - \nabla f(\boldsymbol{W}^2)\| \leq L\|\boldsymbol{W}^1 - \boldsymbol{W}^2\|,$$

(smoothness)

where $\|\boldsymbol{W}\| := \sqrt{\sum_{\ell=1}^{\mathcal{L}} \|W_\ell\|_F^2}$ for any $\boldsymbol{W} = \{W_\ell\}_{\ell=1}^{\mathcal{L}}$.

Assumption 5.3. *The random matrix $\boldsymbol{P} = \{P_\ell\}_{\ell=1}^{\mathcal{L}}$ is sampled from a distribution such that $P_\ell^\top P_\ell = (m_\ell/r_\ell)I_{r_\ell}$ and $\mathbb{E}[P_\ell P_\ell^\top] = I_{m_\ell}$ for each ℓ.*

(random subspace)

RSO convergence theorem

北京大学 PEKING UNIVERSITY

Theorem

Let each subproblem in RSO iteration be solved starting from the initial point $\boldsymbol{B}^0 = \mathbf{0}$ to an expected ϵ-inexact solution $\tilde{\boldsymbol{B}}^k$ with a suitable choice of η^k. The sequence $\{\boldsymbol{X}^k\}$ generated by the RSO method satisfies the following bound:

subproblem optimizer

$$\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\|\nabla f(\boldsymbol{X}^k)\|^2 \leq \boxed{\frac{18\hat{L}\Delta_0}{K}} + \boxed{18\hat{L}\epsilon},$$

RSO framework

where $\Delta_0 := f(\boldsymbol{X}^0) - f^*$ and $\hat{L} := \max_\ell\{m_\ell/r_\ell\}L$.

- RSO converges to ϵ-accurate solution at rate $1/K$
- The accuracy is determined by the subproblem optimizer

RSO convergence theorem with concrete subproblem optimizer

北京大学

$$h(B) = f(X^k + P^k B) + \frac{1}{2\eta_k}\|B\|_F^2$$

Subproblem Solver	Subproblem Complexity	Total Complexity
Stochastic ZO Method	$O((\sum_{\ell=1}^{\mathcal{L}} n_\ell r_\ell)^2 \epsilon^{-2})$	$O((\sum_{\ell=1}^{\mathcal{L}} n_\ell r_\ell)^2 \epsilon^{-3})$
GD	$O(\log \epsilon^{-1})$	$\tilde{O}(\epsilon^{-1})$
Accelerated GD	$O(\log \epsilon^{-1})$	$\tilde{O}(\epsilon^{-1})$
SGD	$O(\epsilon^{-1})$	$O(\epsilon^{-2})$
Momentum SGD	$O(\epsilon^{-1})$	$O(\epsilon^{-2})$
Adam-family	$O(\epsilon^{-1})$	$O(\epsilon^{-2})$
Newton's method	$O(\log(\log \epsilon^{-1}))$	$\tilde{O}(\epsilon^{-1})$
Stochastic Quasi-Newton method	$O(\epsilon^{-1})$	$O(\epsilon^{-2})$

It is observed that RSO with Adam to solve subproblem has sample complexity $O(\epsilon^{-2})$, which is on the same order as vanilla Adam without subspace projection.

北京大学
PEKING UNIVERSITY

PART 03

Experiments

RSO performance： Memory comparison in Pre-train

Algorithm	60M	130M	350M	1B
Adam	34.06 (0.22G)	25.08 (0.50G)	18.80 (1.37G)	15.56 (4.99G)
GaLore	34.88 (0.14G)	25.36 (0.27G)	18.95 (0.49G)	**15.64** (1.46G)
LoRA	34.99 (0.16G)	33.92 (0.35G)	25.58 (0.69G)	19.21 (2.27G)
ReLoRA	37.04 (0.16G)	29.37 (0.35G)	29.08 (0.69G)	18.33 (2.27G)
RSO	**34.55**(0.14G)	**25.34** (0.27G)	**18.86** (0.49G)	15.86 (1.46G)
r/d_{model}	128 / 256	256 / 768	256 / 1024	512 / 2048
Training Tokens (B)	1.1	2.2	6.4	13.1

Table: Comparison of validation perplexity and estimated memory usage for optimizer states across different algorithms during the pre-training of LLaMA models of various sizes on the C4 dataset.

- Saves significant optimizer states memory ; performance degradation is within 2%
- Run for the same number of iterations/samples; performs well for small models (<350M, ~1%)
- Performance deteriorates for relatively large models ($\geq$ 1B, 2%)

RSO performance：Memory comparison in Pre-train

Practical **Total Memory** Profiling (not just optimizer)

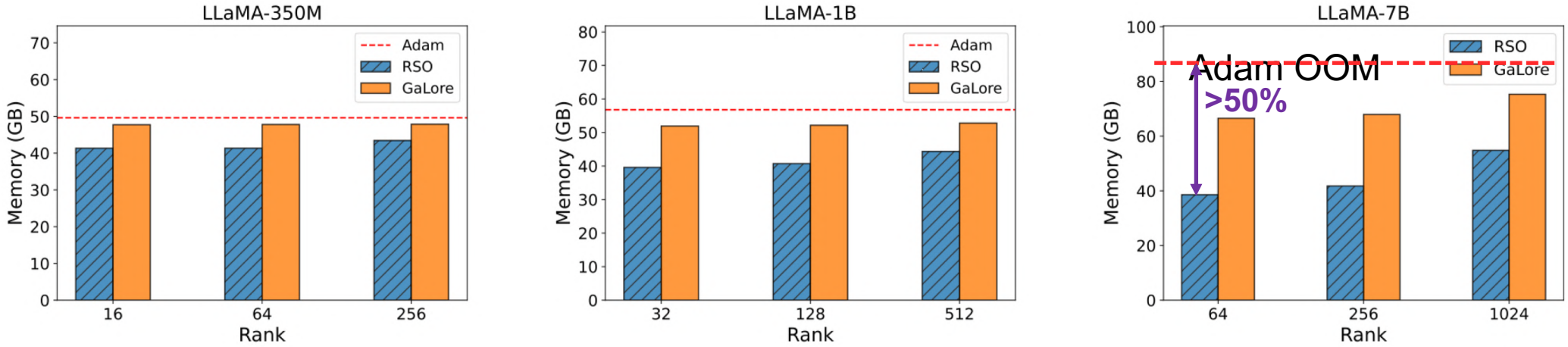


Figure 2: Comparison of peak memory usage (in GB) per device for RSO and GaLore during LLaMA training with varying ranks. All hyperparameters, except rank, are consistent with [Zhao et al., 2024a]. Adam's memory usage is reported for LLaMA-350M and LLaMA-1B but excluded for LLaMA-7B due to an out-of-memory (OOM) error.

- RSO saves gradient, optimizer, and activations, and hence it is more memory-efficient
- As rank increases, memory saving in RSO gets decreased

RSO performance：Computation comparison in Pre-train

Method	LLaMA-1B (Seconds)			LLaMA-7B (Seconds)		
	Seq 64	Seq 128	Seq 256	Seq 64	Seq 128	Seq 256
RSO	0.94	1.70	3.29	2.40	2.94	4.60
GaLore	1.12	1.84	3.35	7.86	8.26	9.12
Adam	1.11	1.81	3.32	7.84	8.23	OOM

Table 5: Comparison of iteration time (in seconds) for different methods in LLaMA training across various sequence lengths. All hyperparameters, except sequence length, follow [Zhao et al., 2024a]. LLaMA-1B runs on 4× A800 GPUs, while LLaMA-7B uses 8× A800 GPUs. SVD decomposition time in GaLore is excluded. Additionally, for LLaMA-7B with a sequence length of 256, the Adam optimizer encounters an out-of-memory (OOM) error.

- RSO achieves much faster training speed due to efficient solving for small subproblems

RSO performance：end-to-end pre-train

北京大学 PEKING UNIVERSITY

Run for the same number of iterations

Method	Memory (GB)	Training Time (h)	Perplexity
Adam	78.92	216	15.43
GaLore	75.33	134	15.59
RSO	54.81	64	15.99

Table 6: Comparison of various pre-training methods for the LLaMA-7B model on the C4 dataset. Perplexity is reported at 50K steps. The training is conducted on 8× A800 GPUs. The actual memory cost per device and the total training time are also reported. RSO and GaLore are configured with a batch size of 16, while Adam uses a batch size of 8.

- With ~3.6% performance degradation, RSO achieves **3.7x** speedup and **30%** memory saving
- It is conjectured that, given the same runtime, RSO outperforms Adam; however, we have not conducted experiments to verify this.

RSO performance：Fine-tune

北京大学 PEKING UNIVERSITY

	CoLA	STS-B	MRPC	RTE	SST2	MNLI	QNLI	QQP	Avg
Adam	62.24	90.92	91.30	79.42	94.57	87.18	92.33	92.28	86.28
GaLore (rank=4)	60.35	**90.73**	**92.25**	**79.42**	94.04	**87.00**	92.24	91.06	85.89
LoRA (rank=4)	61.38	90.57	91.07	78.70	92.89	86.82	92.18	**91.29**	85.61
RSO (rank=4)	**62.47**	90.62	**92.25**	78.70	**94.84**	86.67	**92.29**	90.94	**86.10**
GaLore (rank=8)	60.06	**90.82**	92.01	**79.78**	94.38	**87.17**	92.20	91.11	85.94
LoRA (rank=8)	61.83	90.80	91.90	79.06	93.46	86.94	92.25	91.22	85.93
RSO (rank=8)	**64.62**	90.71	**93.56**	79.42	**95.18**	86.96	**92.44**	**91.26**	**86.77**

Table 4: Evaluation of various fine-tuning methods on the GLUE benchmark using the pre-trained RoBERTa-Base model. The average score across all tasks is provided.

Summary

北京大学 PEKING UNIVERSITY

Memory = **Model** + **Gradient** + **Optimizer states** + **Activations**

- Random subspace optimization (RSO) saves **gradient, optimizer, and activations**

$$\tilde{\boldsymbol{B}}^k \approx \underset{\boldsymbol{B}\in\mathbb{R}^{r\times n}}{\operatorname{argmin}} \left\{ f(\boldsymbol{X}^k + \boldsymbol{P}^k\boldsymbol{B}) + \frac{1}{2\eta^k}\|\boldsymbol{B}\|^2 \right\},$$
$$\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k\tilde{\boldsymbol{B}}^k.$$

- RSO has strong convergence guarantees
- **Open questions**: How to improve RSO performance, especially for large LLMs?

paper

Github Code

References

- Y. He, P. Li, Y. Hu, C. Chen, and **K. Yuan***, *Subspace Optimization for Large Language Models with Convergence Guarantees,* ICML 2025

- Y. Chen, Y. Zhang, Y. Liu, **K. Yuan***, Z. Wen, *A Memory Efficient Randomized Subspace Optimization Method for Training Large Language Models,* ICML 2025

- Y. Chen, Y. Zhang, L. Cao, **K. Yuan***, Z. Wen*, Enhancing Zeroth-Order Fine-tuning for Language Models with Low-Rank Structures,* ICLR 2025

北京大学
PEKING UNIVERSITY

Thank you!

We’re hiring PostDocs in optimization, machine learning, and LLMs

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