北京大學
PEKING UNIVERSITY

Subspace Training in LLMs

Kun Yuan (袁 坤)

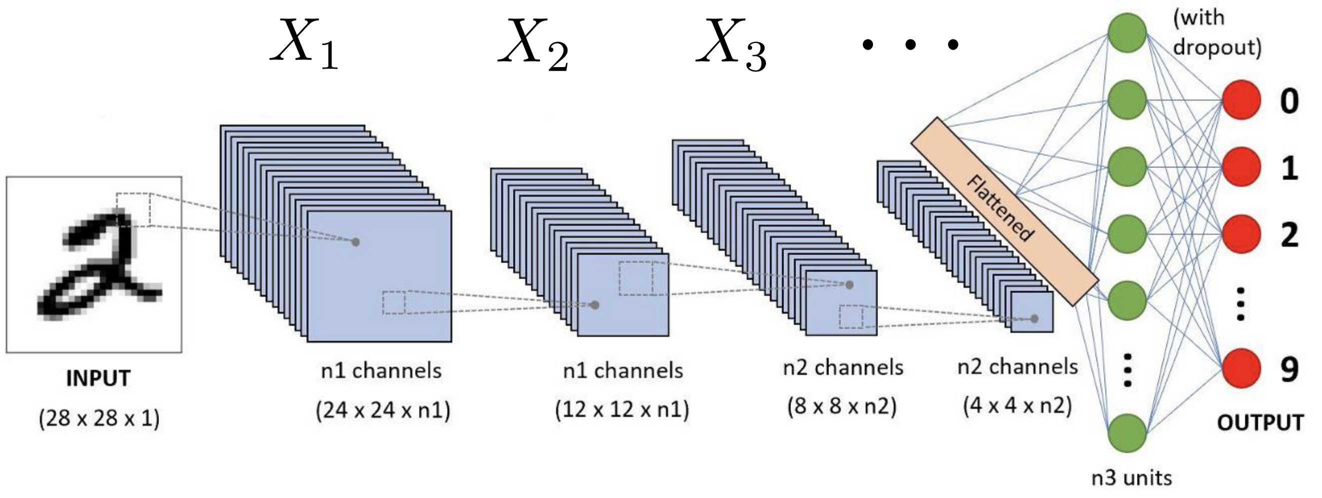
Center for Machine Learning Research @ Peking University

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:

$$\boldsymbol{X}^\star = \arg\min_{\boldsymbol{X}} \Big\{ \underbrace{\mathbb{E}_{\xi\sim\mathcal{D}}}_{\text{data distribution}} \big[\overbrace{L}^{\text{cross entropy}}(\underbrace{h(\boldsymbol{X};\xi)}_{\text{pred. token}}, \underbrace{y}_{\text{real token}}) \big] \Big\}$$

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

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

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

- Adam is the standard approach in LLM pretraining

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

Optimizer states

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

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

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

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
$$G_t = \nabla F(X_t; \xi_t)$$

2P
$$\begin{cases} M_t = (1-\beta_1) M_{t-1} + \beta_1 G_t \\ V_t = (1-\beta_2) V_{t-1} + \beta_2 G_t \odot G_t \end{cases}$$

P
$$X_{t+1} = X_t - \frac{\gamma}{\sqrt{V_t} + \epsilon} \odot M_t$$

Optimizer states introduces significant memory cost

Memory cost to pre-train LLMs

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

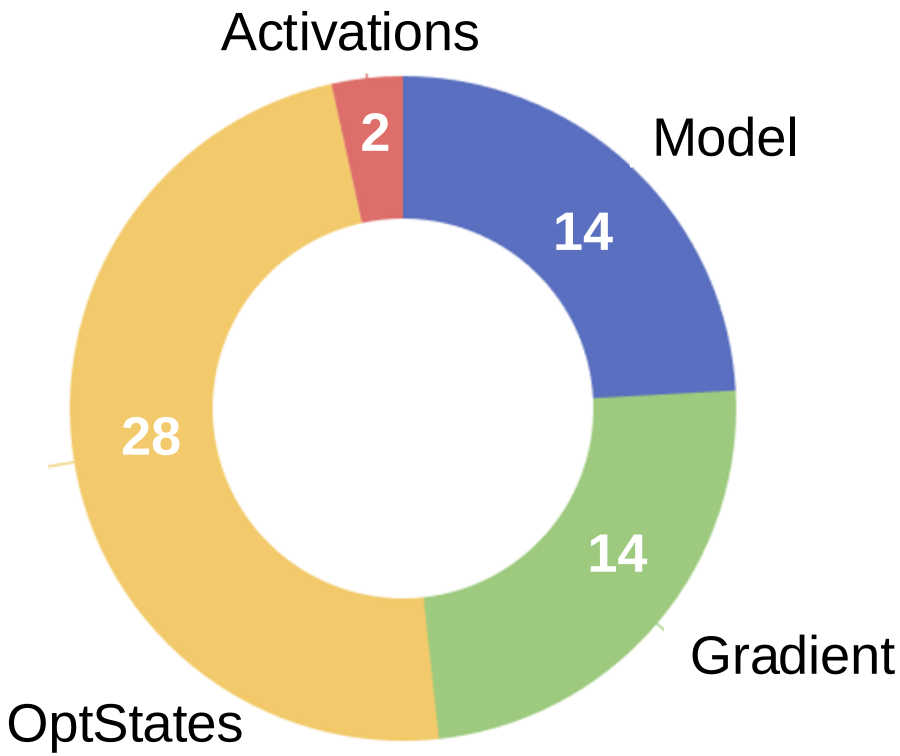
$$\frac{\partial f}{\partial X_i} = \frac{\partial f}{\partial z_i} z_{i-1}^\top$$

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

- 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

- 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

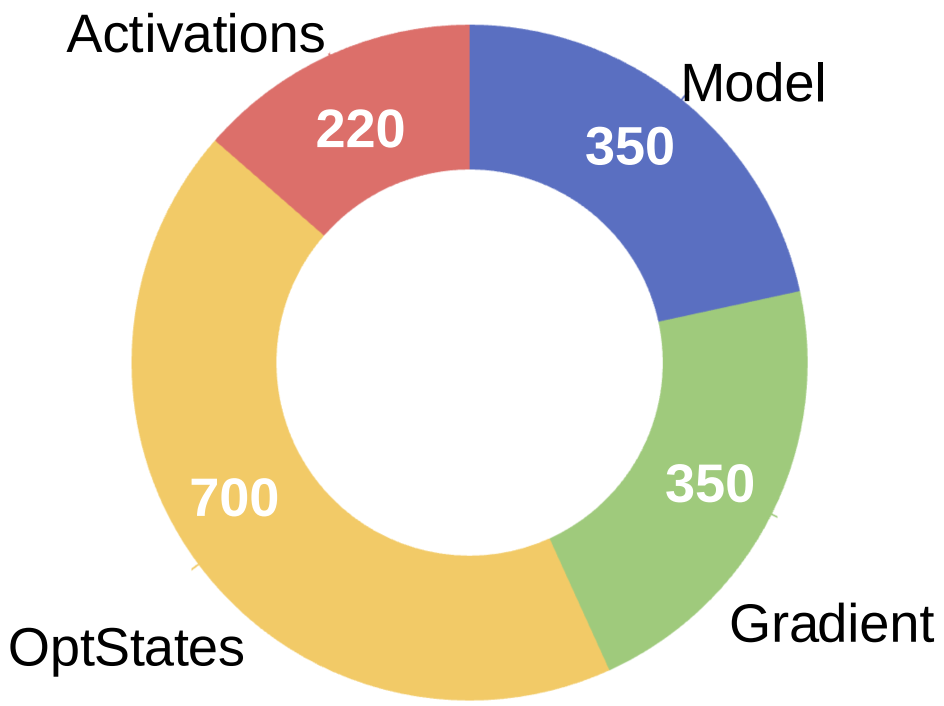
This slide is from Jiawei Zhao's talk on "Memory-efficient LLM Training"

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

- 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**

x 21

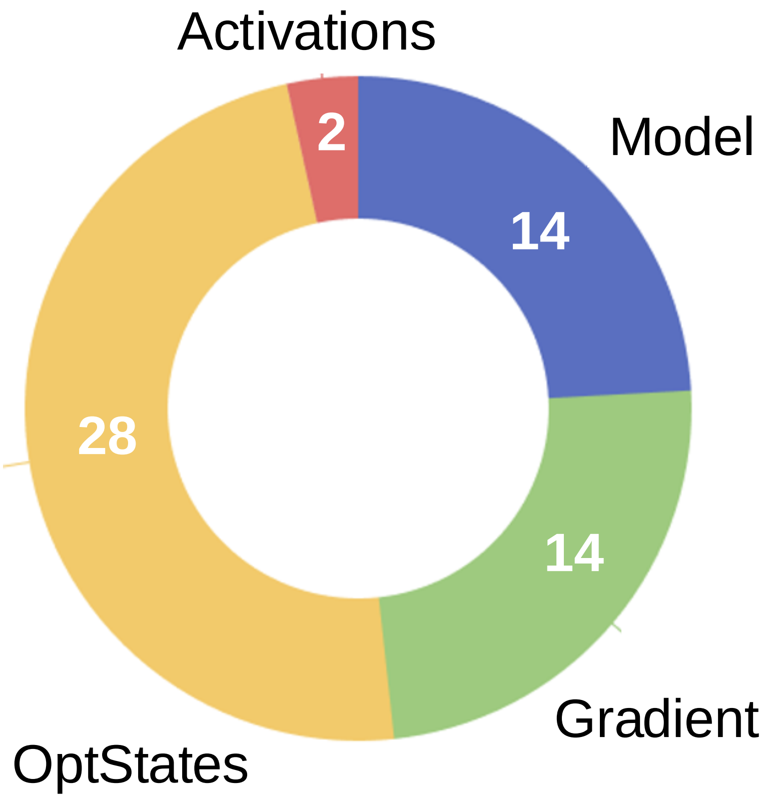
A100 80G

The minimum requirement is A100 * 21

Very expensive!

Memory-efficient algorithm is 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**
- Activation-incurred memory is **relatively minor** when using a single batch size
- Gradient-incurred memory can be **removed** by layer-wise calculation and dropping
- How to save memory caused by optimizer states?



PART 02

Subspace training

Coordinate descent

- Consider the following optimization problem

$$\min_{\mathbf{x}} \quad f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_s)$$

where $\mathbf{x} \in \mathbb{R}^d$ is decomposed into s block variables $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_s$

- Block coordinate descent

$$\mathbf{x}_j^k = \arg\min_{\mathbf{x}_{i_k}} f(\mathbf{x}_{i_k}, \mathbf{x}_{\neq i_k}^{k-1}), \qquad \mathbf{x}_j^k = \mathbf{x}_j^{k-1} \quad \text{if} \quad j \neq i_k$$

where $\mathbf{E}_{i_k} = [\mathbf{0}; \mathbf{0}; \cdots; \mathbf{I}_{i_k}; \cdots; \mathbf{0}] \in \mathbb{R}^{d \times d_{i_k}}$. The above recursion can be rewritten as

$$\mathbf{b}^k = \arg\min_{\mathbf{b}} \; f(\mathbf{x}^{k-1} + \mathbf{E}_{i_k}\mathbf{b}), \qquad \mathbf{x}^k = \mathbf{x}^{k-1} + \mathbf{E}_{i_k}\mathbf{b}^k$$

- We only minimize a small block variable $\mathbf{b}^k$ each time, which saves memory

Subspace optimization

- Now we consider optimization with matrix variables

$$\min_{X \in \mathbb{R}^{m \times n}} \quad f(X)$$

Minimize X directly would result in large memory cost

- Inspired by block coordinate descent, we consider the subspace optimization

$$B^k = \arg\min_{B \in \mathbb{R}^{r \times n}} \quad f(X^{k-1} + P^k B), \qquad X^k = X^{k-1} + P^k B^k$$

where $P^k \in \mathbb{R}^{m \times r}$ is a randomly chosen matrix.

- **Instead of solving X directly, we solve subproblems with a smaller matrix variable**

Subspace optimization: GD variant

$$B^k = \arg\min_{B\in\mathbb{R}^{r\times n}} f(X^{k-1} + P^k B), \quad X^k = X^{k-1} + P^k B^k$$

- Now we consider solving the subproblem with GD, the above problem becomes

$$\boxed{\begin{aligned} &B^{(k,t)} = B^{(k,t-1)} - \gamma (P^k)^\top \nabla_X f(X^{k-1} + P^k B^{(k,t-1)}), \ \forall\, t = 1, 2, \cdots, \tau \\ &\quad X^k = X^{k-1} + P^k B^{(k,\tau)} \end{aligned}}$$

- We let $X^{(k-1,t)} = X^{k-1} + P^k B^{(k,t)}$, the above method reduces to

$$\boxed{\begin{aligned} &X^{(k-1,t)} = X^{(k-1,t-1)} - \gamma P^k (P^k)^\top \nabla_X f(X^{(k-1,t-1)}), \qquad \forall t = 1, \cdots, \tau \\ &X^{(k,0)} = X^{(k-1,\tau)} \end{aligned}}$$

Subspace optimization: GD variant

- We let $X^{(k-1,t)} = X^{k-1} + P^k B^{(k,t)}$, the above method reduces to

$$\begin{aligned} X^{(k-1,t)} &= X^{(k-1,t-1)} - \gamma P^k (P^k)^\top \nabla_X f(X^{(k-1,t-1)}), \qquad \forall t = 1, \cdots, \tau \\ X^{(k,0)} &= X^{(k-1,\tau)} \end{aligned}$$

- The above algorithm can be rewritten as follows

$$X^t = X^{t-1} - \gamma P^t (P^t)^\top \nabla f(X^{t-1}), \qquad \forall t = 1, \cdots, T$$

$$P^t = \begin{cases} \text{Sample new } P & \text{if} \quad \bmod (t, \tau) = 0 \\ P^{t-1} & \text{otherwise} \end{cases}$$

Subspace optimization: advanced variant

北京大学 PEKING UNIVERSITY

$$B^k = \arg\min_{B\in\mathbb{R}^{r\times n}} f(X^{k-1} + P^k B), \quad X^k = X^{k-1} + P^k B^k$$

- Now we consider solving the subproblem with advanced approach $\rho(\cdot)$

$$B^{(k,t)} = B^{(k,t-1)} - \gamma \boldsymbol{\rho}\Big((P^k)^\top \nabla_X f(X^{k-1} + P^k B^{(k,t-1)})\Big), \quad \forall t = 1, 2, \ldots, \tau$$

$$X^k = X^{k-1} + P^k B^{(k,\tau)}$$

- The operator $\rho(\cdot)$ can be either Momentum GD or Adam

(momentum)

$$m^t = (1-\beta)m^{t-1} + \beta g^t$$
$$\rho(g^t) = m^t$$

$$m^t = (1-\beta_1)m^{t-1} + \beta_1 g^t$$
$$v^t = (1-\beta_2)v^{t-1} + \beta_2 g^t \odot g^t$$
$$\rho(g^t) = \frac{m^t}{\sqrt{v^t} + \epsilon}$$

(Adam)

Subspace optimization: advanced variant

$$B^{(k,t)} = B^{(k,t-1)} - \gamma \boldsymbol{\rho}\Big((P^k)^\top \nabla_X f(X^{k-1} + P^k B^{(k,t-1)})\Big), \quad \forall t = 1, 2, \ldots, \tau$$

$$X^k = X^{k-1} + P^k B^{(k,\tau)}$$

- The above algorithm can be rewritten as follows

$$X^t = X^{t-1} - \gamma P^t \boldsymbol{\rho}\Big((P^t)^T \nabla f(X^{t-1})\Big), \quad \forall t = 1, \ldots, T$$

$$P^t = \begin{cases} \text{Sample new } P & \text{if} \quad \bmod(t, \tau) = 0 \\ P^{t-1} & \text{otherwise} \end{cases}$$

北京大學 PEKING UNIVERSITY

PART 03

GaLore Algorithm

GaLore

- In stochastic scenario, the algorithm in Page 18 recovers GaLore

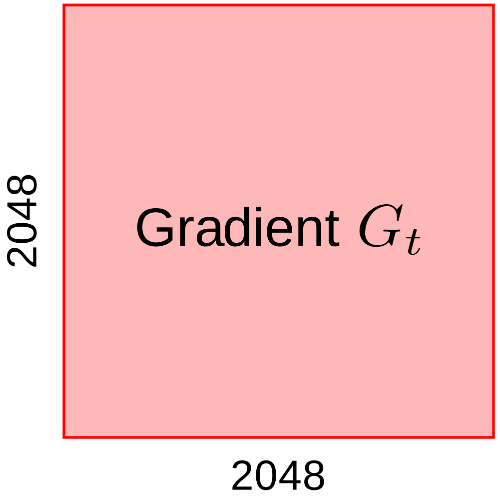
GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection

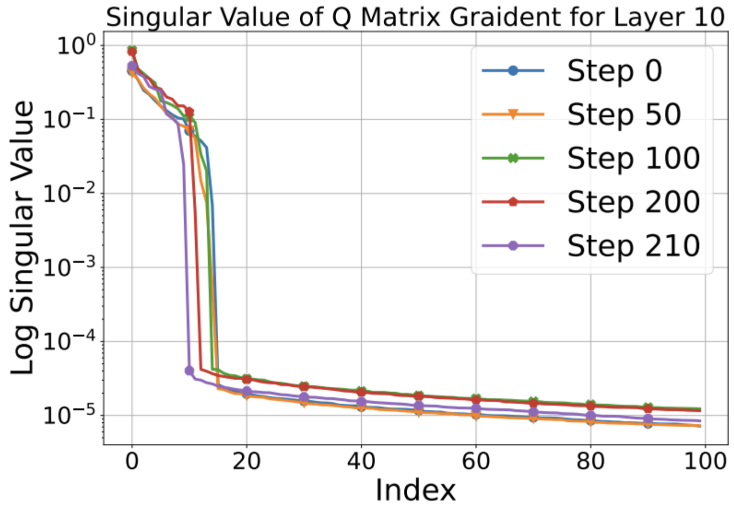
Jiawei Zhao [1] **Zhenyu Zhang** [3] **Beidi Chen** [2,4] **Zhangyang Wang** [3] **Anima Anandkumar** [*,1] **Yuandong Tian** [*,2]

- GaLore is a novel approach for reducing memory consumption from optimizer states
- The first algorithm that enables LLaMA-7B pre-training on a single 4090 GPU (24GB)
- Memory-efficient without severe performance degradation

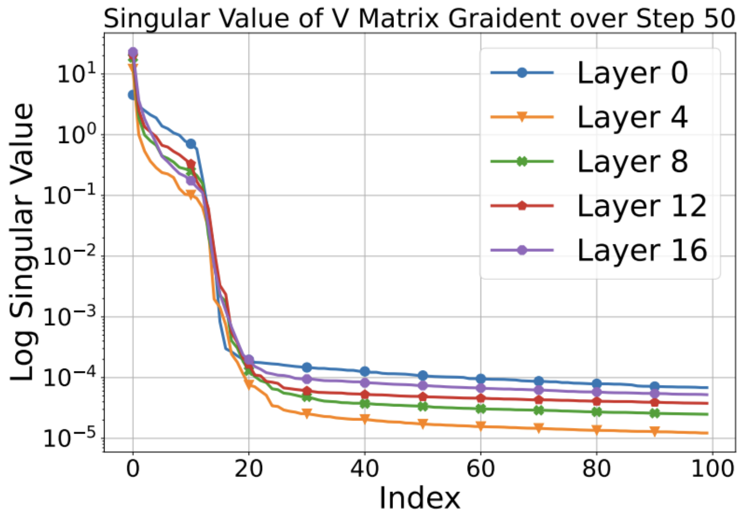
GaLore: Gradient Low-Rank Projection

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





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?

Y. Chen, et. al., *Enhancing Zeroth-Order Fine-tuning for Language Models with Low-Rank Structures*, 2024

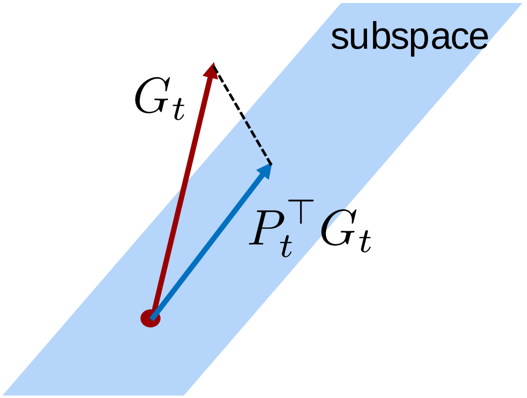
GaLore: Gradient Low-Rank Projection

- 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

北京大學 PEKING UNIVERSITY

- Low-rank optimizer states:

$$g_t = P_t^\top G_t$$ ▷ dims r x n

$$m_t = (1-\beta_1) m_{t-1} + \beta_1 g_t$$ ▷ dims r x n

$$v_t = (1-\beta_2) v_{t-1} + \beta_2 g_t \odot g_t$$ ▷ dims r x n

$$\delta_t = \frac{\gamma}{\sqrt{v_t} + \epsilon} \odot m_t$$ ▷ dims r x n

- Parameter updates:

$$X_{t+1} = X_t - P_t \delta_t$$ ▷ dims m x n

Simplified as

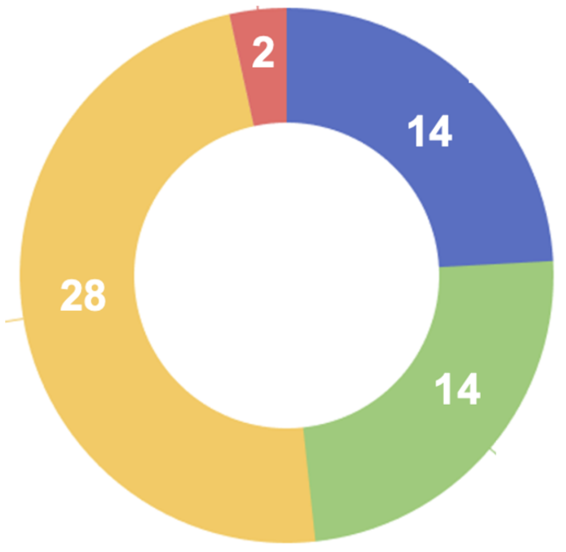
$$X_{t+1} = X_t + P_t \rho(P_t^\top G_t)$$

- Memory cost: Model X, Gradient G, Projection P, OptStates m, v and activations

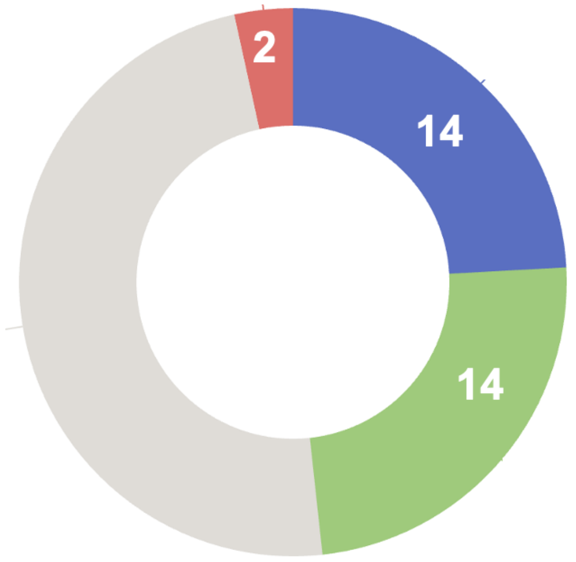
trivial memory cost

GaLore: Gradient Low-Rank Projection

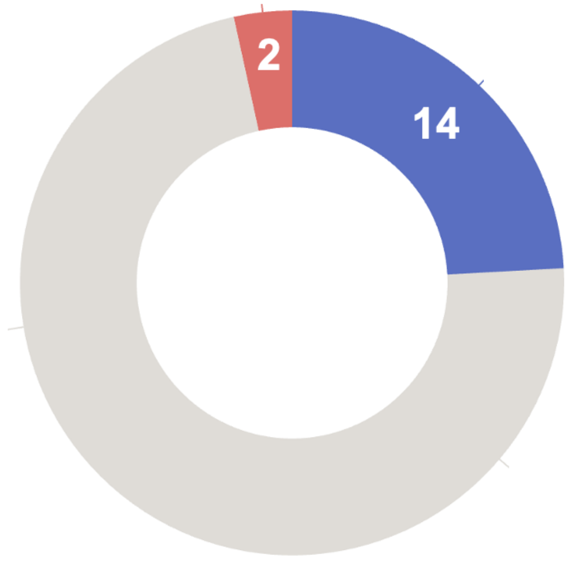
LLaMA 7B



Adam



Adam + GaLore



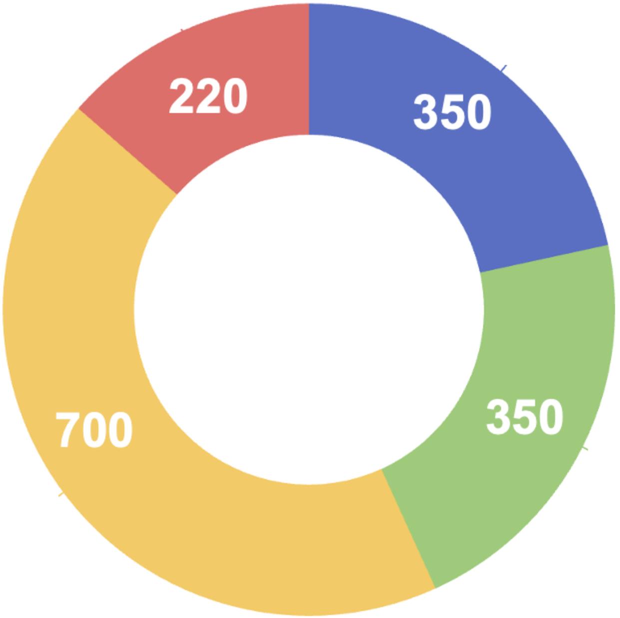
Adam + GaLore + Layerwise gradient dropping (LWGD)

RTX 4090 affordable !!

LWGD is from the paper *Full Parameter Fine-tuning for Large Language Models with Limited Resources*

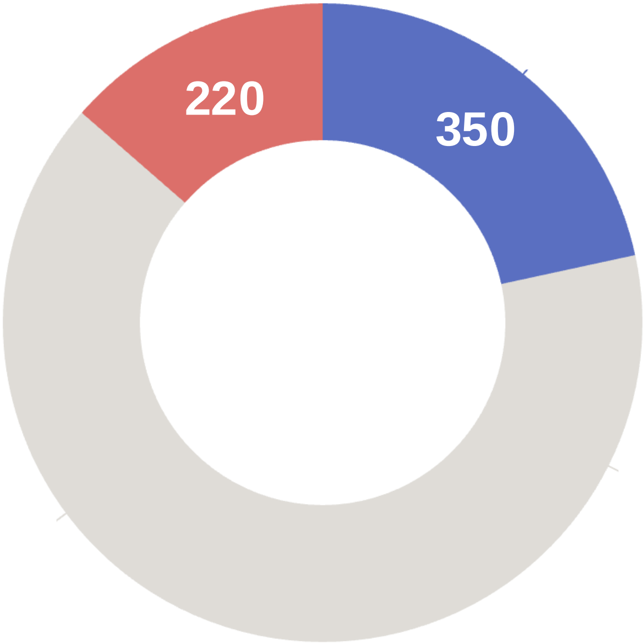
GaLore: Gradient Low-Rank Projection

GPT3 175B



Adam

A100 * 21



Adam + GaLore + LWGD

A100 * 8

GaLore: Gradient Low-Rank Projection

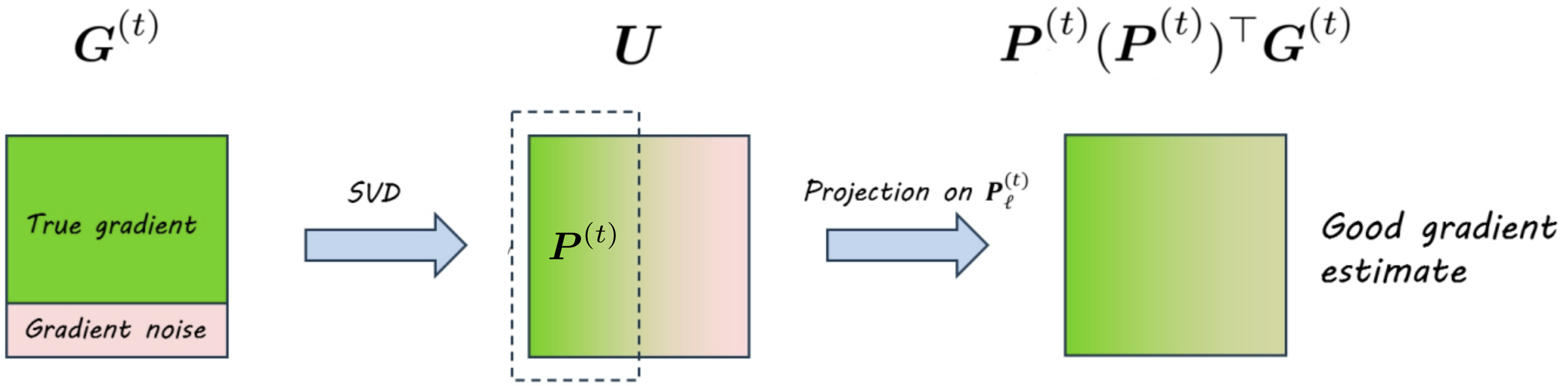
PEKING UNIVERSITY 北京大学

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

- 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

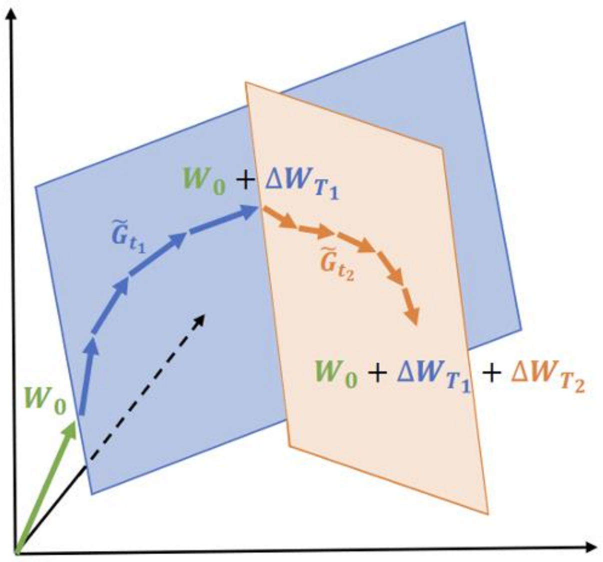
- It is computationally expensive to perform SVD in each iteration

- **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

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: Gradient Low-Rank Projection

Fine-tuning RoBERTa-Base on GLUE

	Memory	CoLA	STS-B	MRPC	RTE	SST2	MNLI	QNLI	QQP	Avg
Full Fine-Tuning	747M	62.24	90.92	91.30	79.42	94.57	87.18	92.33	92.28	86.28
GaLore (rank=4)	253M	60.35	**90.73**	**92.25**	**79.42**	**94.04**	**87.00**	**92.24**	91.06	**85.89**
LoRA (rank=4)	257M	**61.38**	90.57	91.07	78.70	92.89	86.82	92.18	**91.29**	85.61
GaLore (rank=8)	257M	60.06	**90.82**	**92.01**	**79.78**	**94.38**	**87.17**	92.20	91.11	**85.94**
LoRA (rank=8)	264M	**61.83**	90.80	91.90	79.06	93.46	86.94	**92.25**	**91.22**	85.93

GaLore looks great

But does GaLore provably converge to the desired solution?

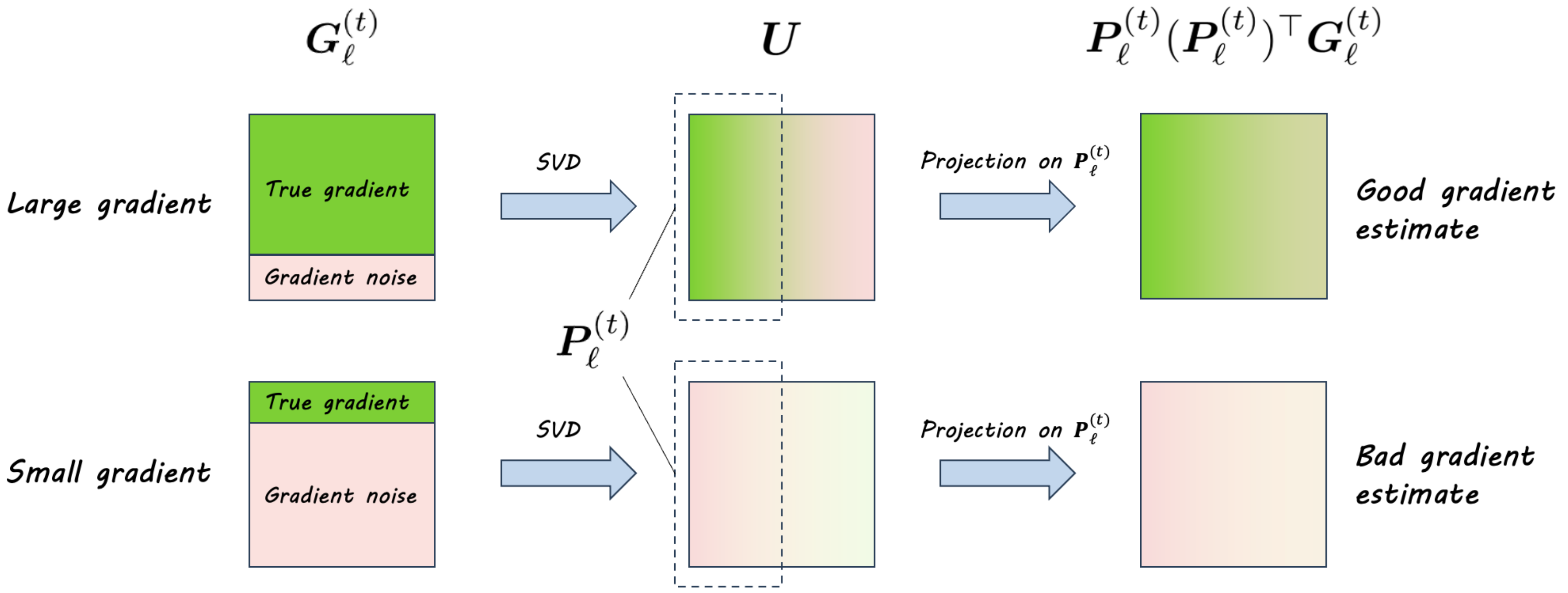
Not Necessarily True!

Y. He, P. Li, Y. Hu, C. Chen, and K. Yuan, *Subspace Optimization for Large Language Models with Convergence Guarantees,* 2024

PART 04

Non-convergence and convergence in GaLore

Intuition behind GaLore's non-convergence

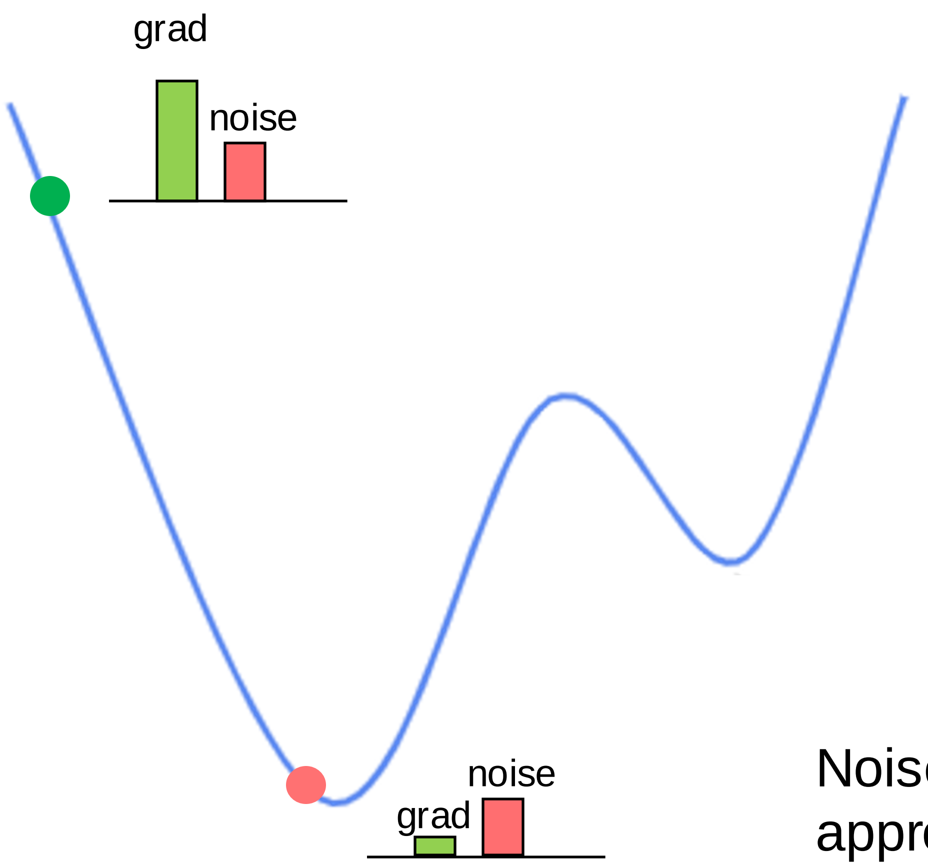


When gradient noise dominates the stochastic gradient, SVD captures **noise-dominated** subspace!

All gradient information is lost !

Is non-convergence common? Yes!

Gradient dominates during the initial stages



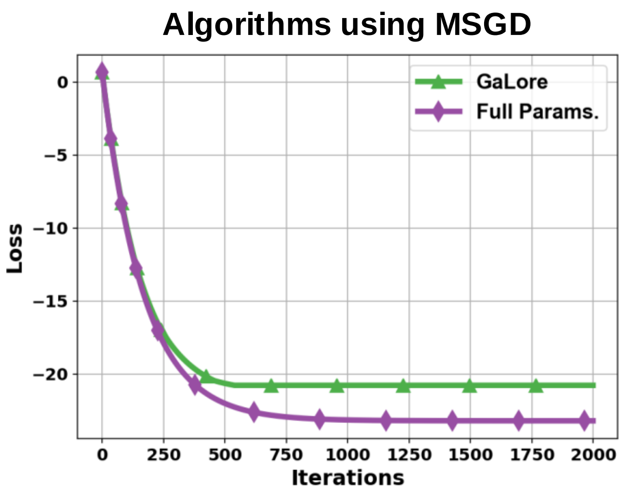
Noise dominates when approaching the local minimum

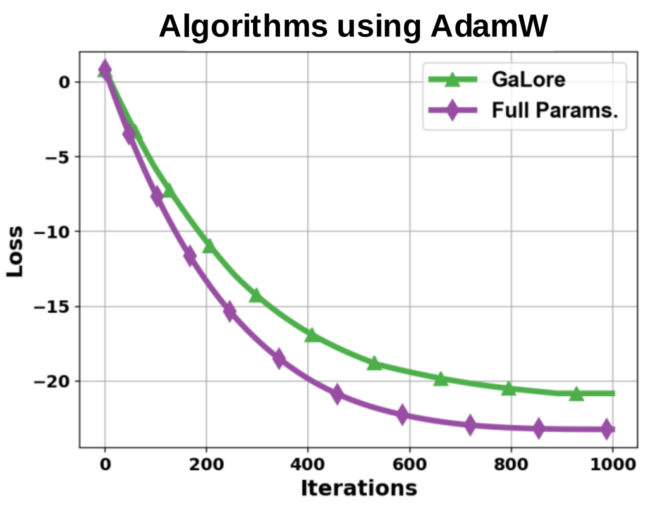
Counter-example

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

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

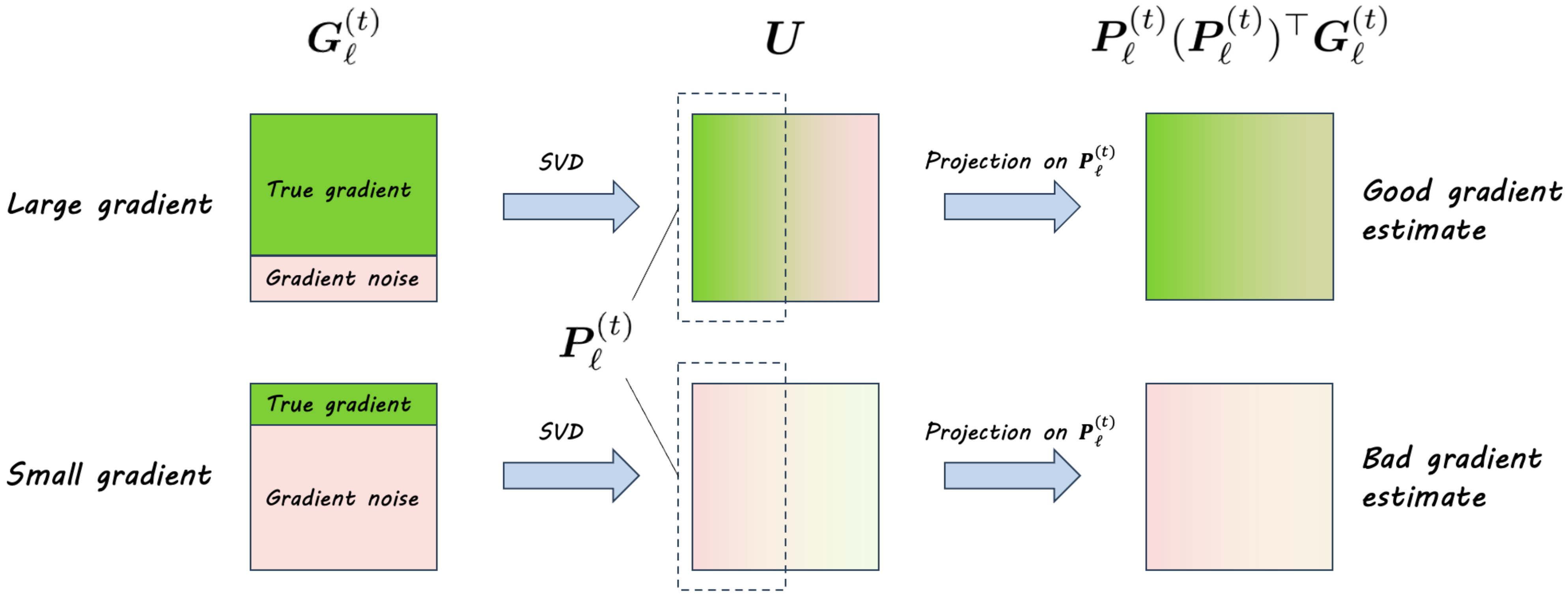
where $\boldsymbol{A} = (\boldsymbol{I}_{n-r} \quad 0) \in \mathbb{R}^{(n-r)\times n}$, $\boldsymbol{B} = \begin{pmatrix} \boldsymbol{D} & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{n\times n}$ with $\boldsymbol{D} \in \mathbb{R}^{(n-r)\times(n-r)}$ generated randomly, $\boldsymbol{C} = \begin{pmatrix}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





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*

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

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* $\tau = \lceil 64/(3\underline{\delta}\beta_1) \rceil$, $\mathcal{B} = \lceil 1/(\underline{\delta}\beta_1) \rceil$,

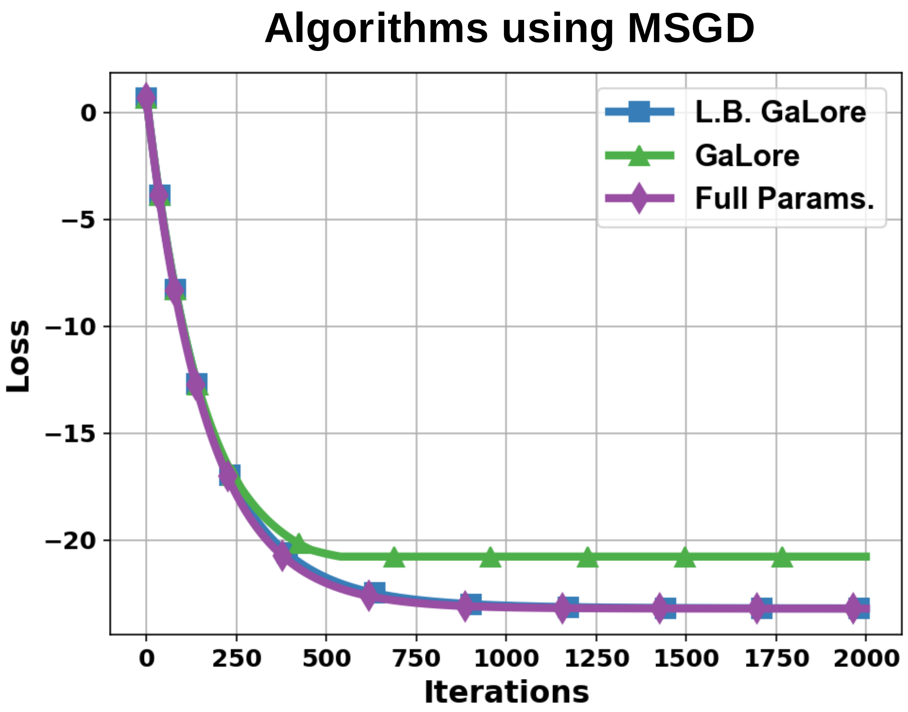
$$\beta_1 = \left(1 + \sqrt{\frac{\underline{\delta}^{3/2}\sigma^2 T}{L\Delta}}\right)^{-1}, \quad \text{and} \quad \eta = \left(4L + \sqrt{\frac{80L^2}{3\underline{\delta}\beta_1^2}} + \sqrt{\frac{40\tau^2 L^2}{\underline{\delta}}} + \sqrt{\frac{32\tau L^2}{3\beta_1}}\right)^{-1},$$

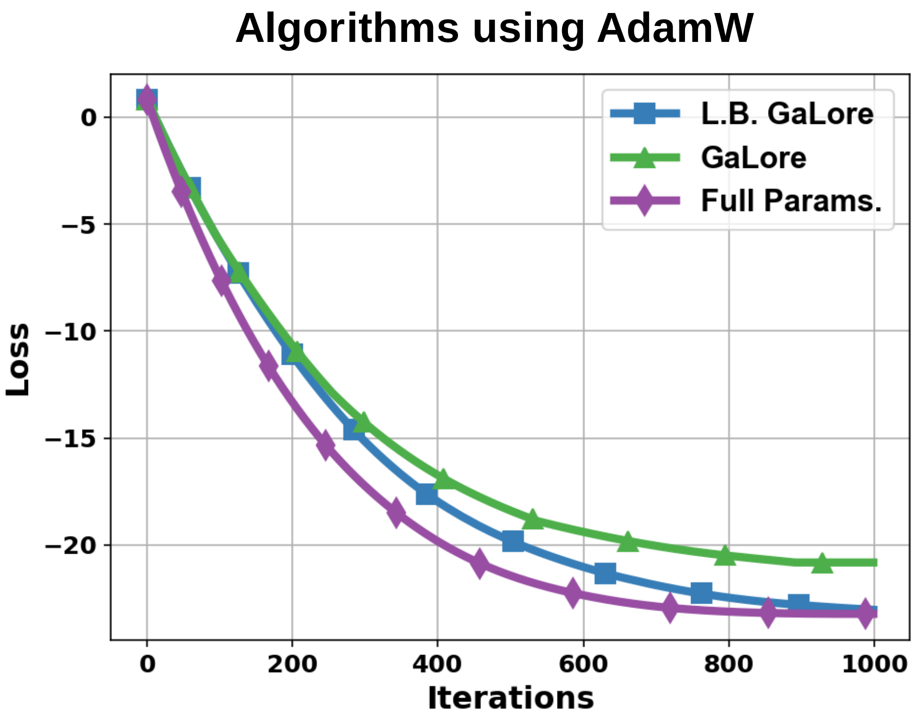
GaLore using large-batch gradients and MSGD with MP converges as

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

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 05

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

Stiefel random projection injects contractive error

PEKING UNIVERSITY 北京大學

- Instead of SVD, GoLore samples the projection matrix uniformly on the Stiefel manifold:

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

- The following Lemma illustrates the projection error in GoLore:

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

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

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* $\tau = \lceil 64/(3\underline{\delta}\beta_1) \rceil$,

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

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 $\mathcal{O}(1/\sqrt{T})$.

Convergence analysis for subspace GD

- The minimization problem $\min_{X \in \mathbb{R}^{m\times n}} f(X)$
- Subspace gradient descent: $X^t = X^{t-1} - \gamma P^t (P^t)^\top \nabla f(X^{t-1}), \qquad \forall t = 1, \cdots, T$
- For simplicity, we assume there is no lazy update on the projection matrix P
- Assumptions on projection matrix P: $(P^t)^\top P^t = I_r$ and $\mathbb{E}[P^t (P^t)^\top] = \frac{r}{m} I_m$

$$
\begin{aligned}
f(X^t) &\le f(X^{t-1}) + \langle \nabla f(X^{t-1}), X^t - X^{t-1} \rangle + \frac{L}{2} \|X^t - X^{t-1}\|^2 \\
&\le f(X^{t-1}) - \gamma \langle \nabla f(X^{t-1}), P^t (P^t)^\top \nabla f(X^{t-1}) \rangle + \frac{\gamma^2 L}{2} \|P^t (P^t)^\top \nabla f(X^{t-1})\|^2 \\
&= f(X^{t-1}) - \gamma \|(P^t)^\top \nabla f(X^{t-1})\|^2 + \frac{\gamma^2 L}{2} \|(P^t)^\top \nabla f(X^{t-1})\|^2 \\
&= f(X^{t-1}) - \gamma (1 - \frac{\gamma L}{2}) \|(P^t)^\top \nabla f(X^{t-1})\|^2
\end{aligned}
$$

Convergence analysis for subspace GD

- Taking expectation on the random projection matrix P, we have

$$
\begin{aligned}
\mathbb{E}f(X^t) &\le \mathbb{E}f(X^{t-1}) - \gamma(1 - \frac{\gamma L}{2})\frac{r}{m}\mathbb{E}\|\nabla f(X^{t-1})\|^2 \\
&\le \mathbb{E}f(X^{t-1}) - \frac{\gamma r}{2m}\mathbb{E}\|\nabla f(X^{t-1})\|^2
\end{aligned}
$$

- From the above inequality, we achieve

$$
\|\nabla f(X^{t-1})\|^2 \le \frac{2m\big(\mathbb{E}f(X^{t-1}) - \mathbb{E}f(X^t)\big)}{\gamma r}
$$

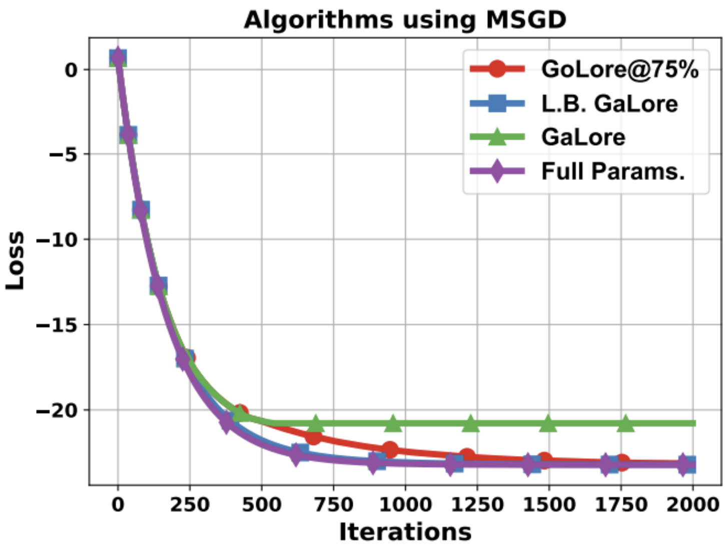
and therefore

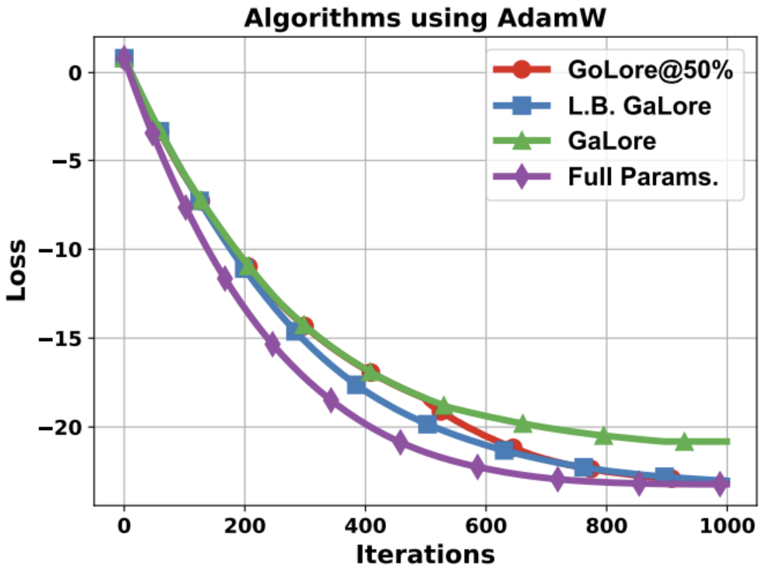
$$
\frac{1}{T+1}\sum_{t=0}^{T}\|\nabla f(X^t)\|^2 \le \frac{2mL\big(f(X^0) - f^\star\big)}{r(T+1)}
$$

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

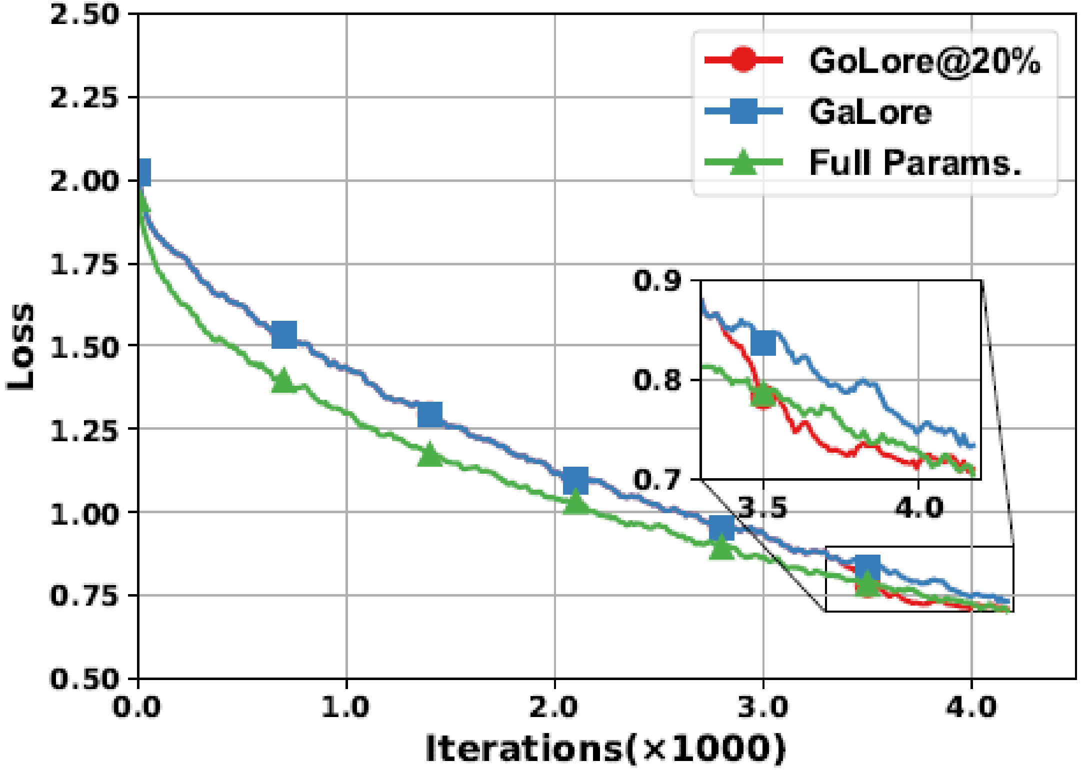
GoLore@x% = GaLore (first (100-x)% iters) + GoLore (last x% iters)





Experimental results

Fine-tuning LLaMA2-7B on WinoGrande:



Experimental results

- 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.

Improving the computational efficiency

- GaLore/GoLore always computes the full gradient before compressing them into subspaces.
- Can we compute the compressed gradient directly, without computing the full gradient?

Original Implementation

New Implementation

$$y = Wx \xrightarrow{W = W_0 + BA} y = W_0 x + BAx$$

$$\nabla_W \mathcal{L} = (\nabla_y \mathcal{L}) x^\top \quad \text{Backpropagated gradient} \quad \nabla_A \mathcal{L} = B^\top (\nabla_y \mathcal{L}) x^\top$$

$$W \leftarrow W + B\rho(B^\top(\nabla_W \mathcal{L})) \quad \Leftrightarrow \quad A \leftarrow A + \rho(\nabla_A \mathcal{L})$$

Improving the computational efficiency

- Comparing the computational complexities:

GaLore Implementation	Memory	Computation
(Zhao et al., 2024)	$mn + rm + rn + bm$	$6bmn + 4rmn + 2mn + 3rn$
Our ReLoRA-like version	$mn + rm + 2rn + bm + br$	$4bmn + 4brm + 6brn + 5rn$

- When $r \ll \min\{m, n\}$, this new version reduces the computation complexity from $(6b + 4r + 2)mn$ to $4bmn$, with minimal memory overhead.

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

Take-home message