



Memory-Efficient Zeroth-Order Optimization

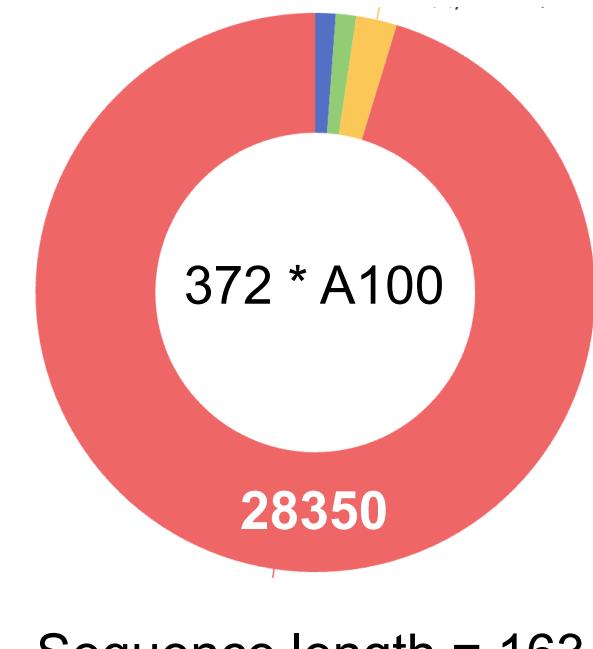
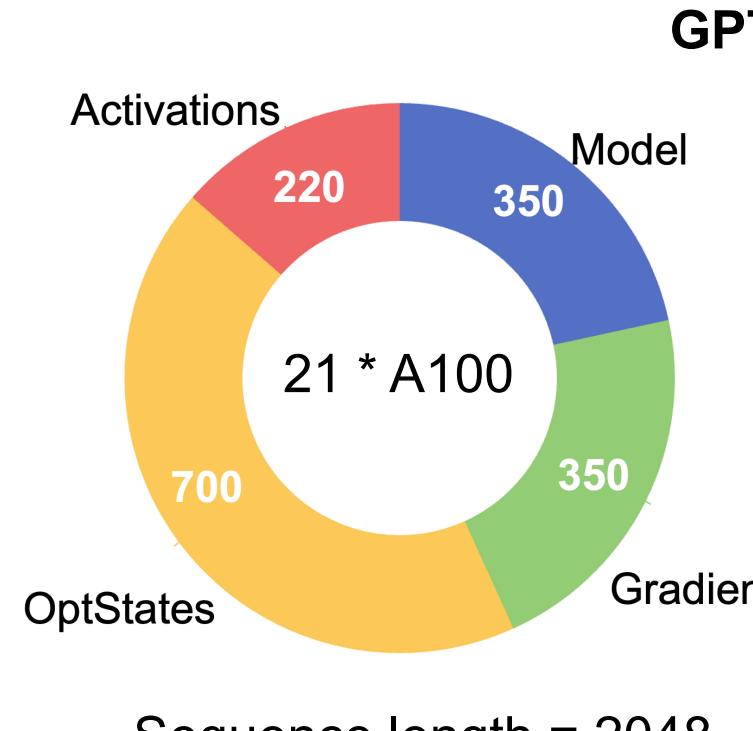
Kun Yuan (袁 坤)

Center for Machine Learning Research @ Peking University

Activation dominates when sequence length is large

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

Activation dominates when
sequence length is large



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**

How to save memory especially for long sequence or large batch scenarios?

PART 02

Memory-Efficient Zeroth-Order Optimizer (MeZO)

- Activations are auxiliary variables to facilitate the gradient calculations

Consider a linear neural network

$$z_i = X_i z_{i-1}, \forall i = 1, \dots, L$$

$$f = \mathcal{L}(z_L; y)$$

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, \dots, z_L

- **If we do not calculate gradient, we do not need activations!**
- Can we fine-tune LLMs without calculating gradients? **Zeroth-order optimization**

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

- Consider the Random GEstimator (RGE)

$$(RGE) \quad \hat{\nabla} F(\mathbf{X}; \xi) := \frac{F(\mathbf{X} + \epsilon \mathbf{Z}; \xi) - F(\mathbf{X} - \epsilon \mathbf{Z}; \xi)}{2\epsilon} \mathbf{Z}. \quad \mathbf{Z} \sim \mathcal{N}(0, 1)$$

- RGE approximates gradient with finite function value differences
- Zeroth-order stochastic gradient descent (ZO-SGD):

$$\mathbf{X}^{t+1} = \mathbf{X}^t - \alpha \hat{\nabla} F(\mathbf{X}^t; \xi^t),$$

No need for backward propagation; Just forward passes; **No need to store activations!**

Memory-efficient Zeroth-Order (MeZO) algorithm

- Do we need to store gradient in $X^{t+1} = X^t - \alpha \hat{\nabla} F(X^t; \xi^t)$? **No!**

Algorithm 1: MeZO

Require: parameters $\theta \in \mathbb{R}^d$, loss $\mathcal{L} : \mathbb{R}^d \rightarrow \mathbb{R}$, step budget T , perturbation scale ϵ , batch size B , learning rate schedule $\{\eta_t\}$

```

for  $t = 1, \dots, T$  do
    Sample batch  $\mathcal{B} \subset \mathcal{D}$  and random seed  $s$ 
     $\theta \leftarrow \text{PerturbParameters}(\theta, \epsilon, s)$ 
     $\ell_+ \leftarrow \mathcal{L}(\theta; \mathcal{B})$   $L(\theta + \epsilon Z)$ 
     $\theta \leftarrow \text{PerturbParameters}(\theta, -2\epsilon, s)$ 
     $\ell_- \leftarrow \mathcal{L}(\theta; \mathcal{B})$   $L(\theta - \epsilon Z)$ 
     $\theta \leftarrow \text{PerturbParameters}(\theta, \epsilon, s)$  ▷ Reset parameters before descent
    projected_grad  $\leftarrow (\ell_+ - \ell_-)/(2\epsilon)$ 
    Reset random number generator with seed  $s$  ▷ For sampling  $z$ 
    for  $\theta_i \in \theta$  do
         $z \sim \mathcal{N}(0, 1)$ 
         $\theta_i \leftarrow \theta_i - \eta_t * \text{projected\_grad} * z$  Zeroth-order SGD
    end
end

```

Subroutine PerturbParameters(θ, ϵ, s)

Reset random number generator with seed s

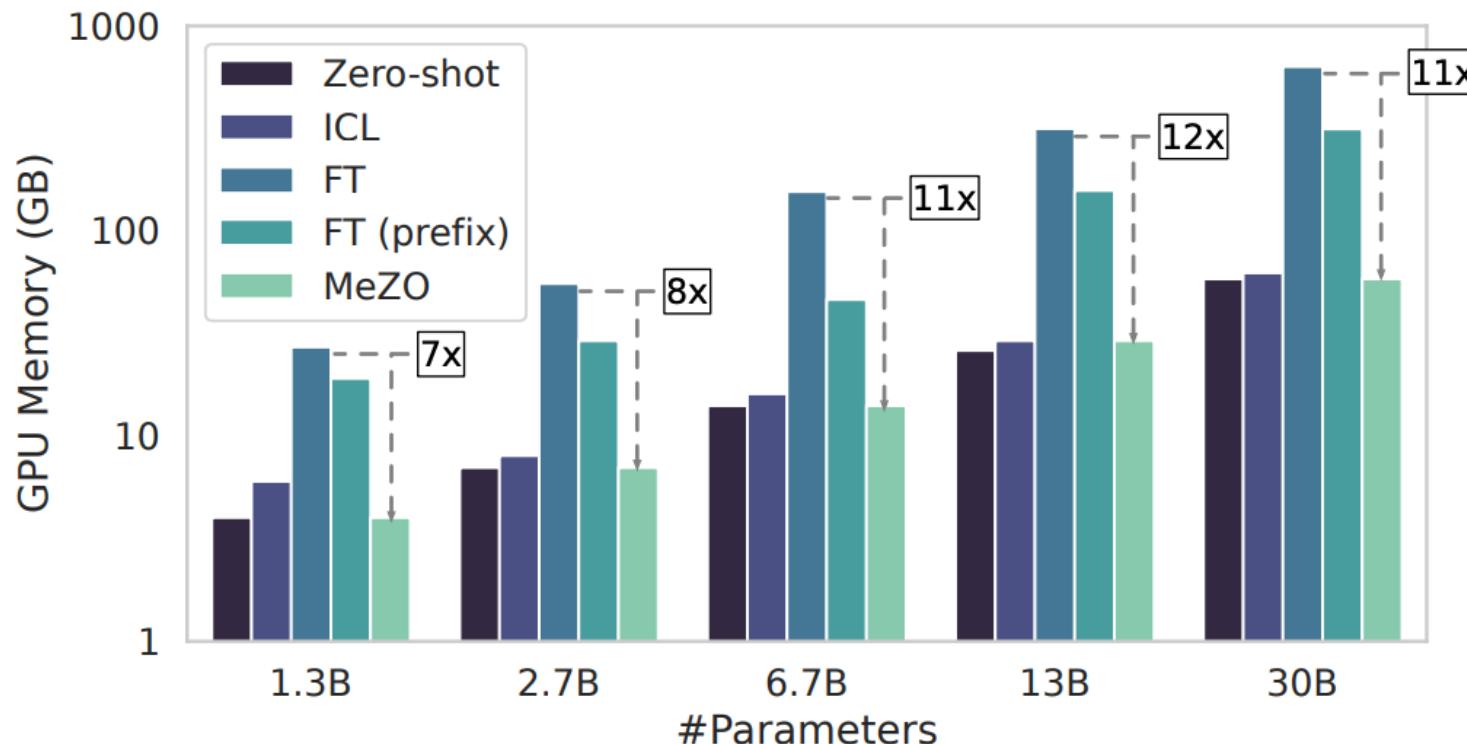
```

for  $\theta_i \in \theta$  do
     $z \sim \mathcal{N}(0, 1)$ 
     $\theta_i \leftarrow \theta_i + \epsilon z$ 
end
return  $\theta$ 

```

No need to store Z , $X + \epsilon Z$, $X - \epsilon Z$ and the gradient. Only one model is stored.

MeZO: Memory-efficient Zeroth-Order Optimization



MeZO saves significant memory compared to FT (First-order Full-parameter Fine-tuning).

Figure 3: GPU memory consumption with different OPT models and tuning methods on MultiRC (400 tokens per example on average).

MeZO: Memory-efficient Zeroth-Order Optimization



Method	Zero-shot / MeZO	ICL	Prefix FT	Full-parameter FT
1.3B	1xA100 (4GB)	1xA100 (6GB)	1xA100 (19GB)	1xA100 (27GB)
2.7B	1xA100 (7GB)	1xA100 (8GB)	1xA100 (29GB)	1xA100 (55GB)
6.7B	1xA100 (14GB)	1xA100 (16GB)	1xA100 (46GB)	2xA100 (156GB)
13B	1xA100 (26GB)	1xA100 (29GB)	2xA100 (158GB)	4xA100 (316GB)
30B	1xA100 (58GB)	1xA100 (62GB)	4xA100 (315GB)	8xA100 (633GB)
66B	2xA100 (128GB)	2xA100 (134GB)	8xA100	16xA100

Table 22: Memory usage on the MultiRC (avg #tokens=400) dataset.

MeZO: Memory-efficient Zeroth-Order Optimization



Fine-tuning LLMs

Task	SST-2	RTE	CB	BoolQ	WSC	WIC	MultiRC	COPA	ReCoRD	SQuAD	DROP
Task type	classification						– multiple choice –		— generation —		
Zero-shot	58.8	59.6	46.4	59.0	38.5	55.0	46.9	80.0	81.2	46.2	14.6
ICL	87.0	62.1	57.1	66.9	39.4	50.5	53.1	87.0	82.5	75.9	29.6
LP	93.4	68.6	67.9	59.3	63.5	60.2	63.5	55.0	27.1	3.7	11.1
MeZO	91.4	66.1	67.9	67.6	63.5	61.1	60.1	88.0	81.7	84.7	30.9
MeZO (LoRA)	89.6	67.9	66.1	73.8	64.4	59.7	61.5	84.0	81.2	83.8	31.4
MeZO (prefix)	90.7	70.8	69.6	73.1	60.6	59.9	63.7	87.0	81.4	84.2	28.9
FT (12x memory)	92.0	70.8	83.9	77.1	63.5	70.1	71.1	79.0	74.1	84.9	31.3

MeZO achieves comparable (within 1%) or better performance than FT on **6 out of 11 tasks**.

Open questions

There are two open problems to be solved:

MeZO is using SGD. Its direct extension to momentum SGD is **NOT memory-efficient**

$$\mathbf{X}^{t+1} = \mathbf{X}^t - \alpha \hat{\nabla} F(\mathbf{X}^t; \xi^t),$$

Zeroth-order SGD



$$\mathbf{M}^t = (1 - \beta) \mathbf{M}^{t-1} + \beta \hat{\nabla} F(\mathbf{X}^t; \xi^t)$$

$$\mathbf{X}^{t+1} = \mathbf{X}^t - \alpha \mathbf{M}^t$$

Zeroth-order MSGD

Need to store momentum M

Q1. Can we develop memory-efficient zeroth-order momentum SGD?

Q2. Can we further improve MeZO's performance ?

PART 03

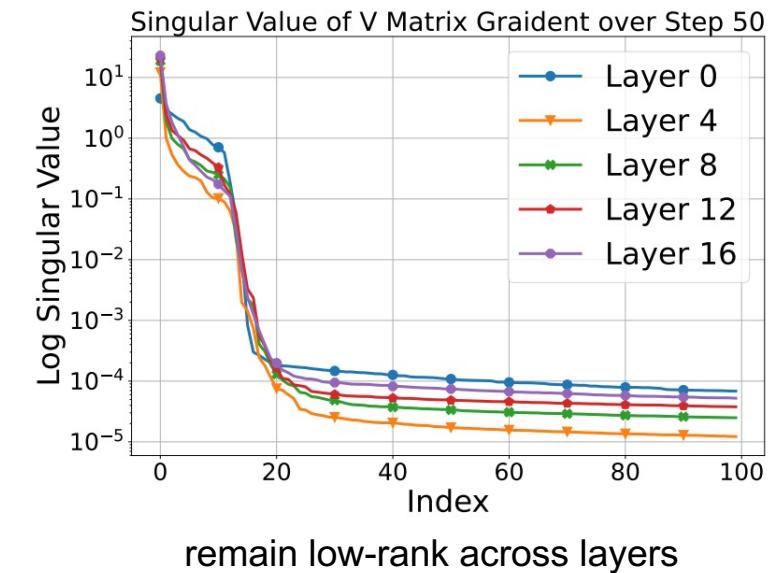
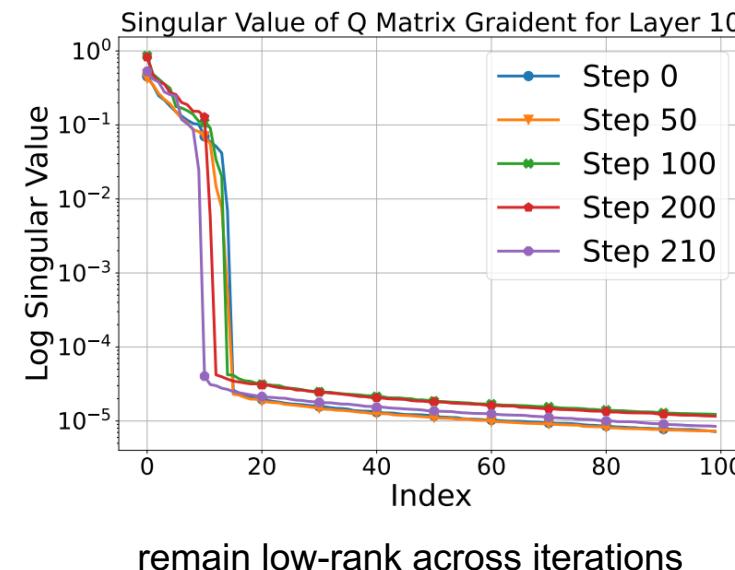
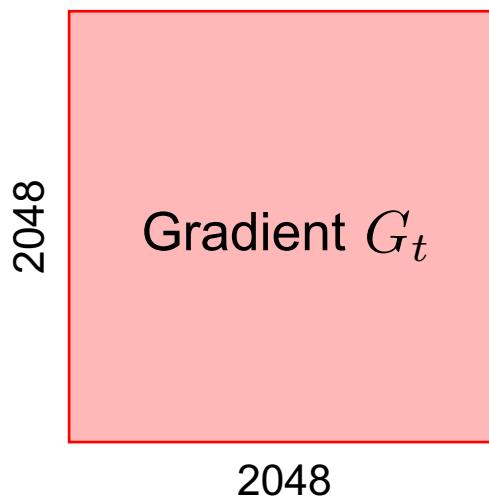
Low-Rank Zeroth-Order Optimizer (LOZO)

MeZO cannot approximate low-rank gradient

- Recall the Random GEstimator (RGE)

$$(\text{RGE}) \quad \hat{\nabla}F(\mathbf{X}; \xi) := \frac{F(\mathbf{X} + \epsilon \mathbf{Z}; \xi) - F(\mathbf{X} - \epsilon \mathbf{Z}; \xi)}{2\epsilon} \mathbf{Z}, \quad \mathbf{Z} \sim \mathcal{N}(0, 1)$$

- Since $\mathbf{Z} \sim \mathcal{N}(0, 1)$, it has almost **full-rank**. However, gradients in LLMs are low-rank



- This mismatch would result in performance degradation

Low-rank gradient estimator

- RGE cannot approximate low-rank gradient; not proper for LLMs fine-tuning

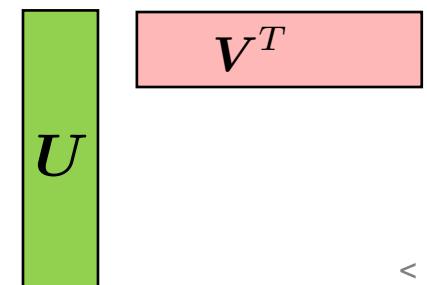
$$(RGE) \quad \hat{\nabla}F(\mathbf{X}; \xi) := \frac{F(\mathbf{X} + \epsilon \mathbf{Z}; \xi) - F(\mathbf{X} - \epsilon \mathbf{Z}; \xi)}{2\epsilon} \mathbf{Z}. \quad \mathbf{Z} \sim \mathcal{N}(0, 1)$$

- We propose a **Low-rank Gradient Estimator (LGE)**

$$(LGE) \quad \hat{\nabla}F(\mathbf{X}; \xi) := \frac{F(\mathbf{X} + \epsilon \mathbf{U} \mathbf{V}^T; \xi) - F(\mathbf{X} - \epsilon \mathbf{U} \mathbf{V}^T; \xi)}{2\epsilon} (\mathbf{U} \mathbf{V}^T / \mathbf{r}).$$

where $\mathbf{U} \sim \mathcal{N}(0, 1) \in \mathbb{R}^{m \times r}, \mathbf{V} \sim \mathcal{N}(0, 1) \in \mathbb{R}^{n \times r}$

- Apparently, $\mathbf{U} \mathbf{V}^T$ is a low-rank matrix to approximate the true gradient



LOZO: Low-Rank Zeroth-Order SGD

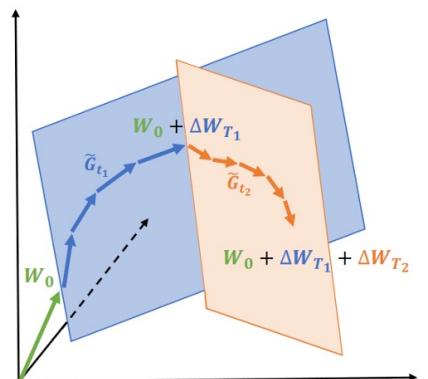
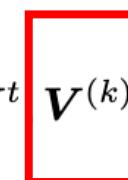
- LGE estimator: $\text{LGE}(\mathbf{X}, \mathbf{U}, \mathbf{V}, \mathbf{r}, \epsilon, \xi) := \frac{F(\mathbf{X} + \epsilon \mathbf{U} \mathbf{V}^T; \xi) - F(\mathbf{X} - \epsilon \mathbf{U} \mathbf{V}^T; \xi)}{2\epsilon} (\mathbf{U} \mathbf{V}^T / \mathbf{r})$.
- LOZO vanilla version:

$$\mathbf{X}^{t+1} = \mathbf{X}^t - \alpha \hat{\nabla} F(\mathbf{X}^t; \xi^t) \quad \text{where} \quad \hat{\nabla} F(\mathbf{X}^t; \xi^t) = \text{LGE}(\mathbf{X}^t, \mathbf{U}^t, \mathbf{V}^t, \mathbf{r}, \epsilon, \xi^t).$$

- If U^t and V^t are resampled per iteration, the subspace shifts too quickly; hurt performance
- **Lazy sampling**: explore each subspace sufficiently

$$\mathbf{X}^{t+1} = \mathbf{X}^t - \alpha \hat{\nabla} F(\mathbf{X}^t; \xi^t), \quad \text{where} \quad \hat{\nabla} F(\mathbf{X}^t; \xi^t) = \text{LGE}(\mathbf{X}^t, \mathbf{U}^t, \boxed{\mathbf{V}^{(k)}} \mathbf{r}, \epsilon, \xi^t).$$

Sample V every τ iterations; search the space spanned by V sufficiently



Does this heuristic algorithm converge?

LOZO is essentially a zeroth-order subspace optimization method

- Recall the random **coordinate** minimization:

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

$$x^{k+1} = x^k + b_k^* \cdot e_{i_k}, \quad b \text{ is the increment in the } i\text{-th coordinate}$$

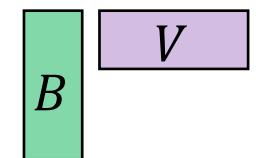
where the variable x is a vector in the above method

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

$$\mathbf{B}_k^* = \arg \min_{\mathbf{B}} \{f(\mathbf{X}^{(k)} + \mathbf{B}(\mathbf{V}^{(k)})^T)\}, \text{ where } \mathbf{V}^{(k)} \text{ follows a normal distribution,}$$

$$\mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} + \mathbf{B}_k^*(\mathbf{V}^{(k)})^T.$$

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



LOZO is essentially a zeroth-order subspace optimization method

- Solve the inner subspace problem with zeroth-order SGD for ν iterations:

$$\mathbf{B}^{(k,s+1)} = \mathbf{B}^{(k,s)} - \gamma \hat{\nabla}_{\mathbf{B}} F(\tilde{\mathbf{X}}^{(k)} + \mathbf{B}^{(k,s)} (\mathbf{V}^{(k)})^T; \xi^{(k,s)}), \quad s = 0, \dots, \nu - 1, \quad (1)$$

$$\tilde{\mathbf{X}}^{(k+1)} = \tilde{\mathbf{X}}^{(k)} + \mathbf{B}^{(k,\nu)} (\mathbf{V}^{(k)})^T. \quad (2)$$

- Let the $\mathbf{U}^{(k,s)}$ be the perturbation, the gradient can be estimated with standard zeroth-order approach

$$\begin{aligned} & \hat{\nabla}_{\mathbf{B}} F(\tilde{\mathbf{X}}^{(k)} + \mathbf{B}^{(k,s)} (\mathbf{V}^{(k)})^T; \xi^{(k,s)}) \\ &= \frac{F(\tilde{\mathbf{X}}^{(k)} + (\mathbf{B}^{(k,s)} + \epsilon \mathbf{U}^{(k,s)})(\mathbf{V}^{(k)})^T; \xi^{(k,s)}) - F(\tilde{\mathbf{X}}^{(k)} + (\mathbf{B}^{(k,s)} - \epsilon \mathbf{U}^{(k,s)})(\mathbf{V}^{(k)})^T; \xi^{(k,s)})}{2\epsilon} \mathbf{U}^{(k,s)}. \end{aligned}$$

- Let $\tilde{\mathbf{X}}^{(k,s+1)} = \tilde{\mathbf{X}}^{(k,s)} + \mathbf{B}^{(k,s)} \mathbf{V}^{(k)}$, algorithm (1)-(2) can be rewritten as follows

$$\mathbf{X}^{t+1} = \mathbf{X}^t - \alpha \hat{\nabla} F(\mathbf{X}^t; \xi^t), \quad \text{where} \quad \hat{\nabla} F(\mathbf{X}^t; \xi^t) = \frac{F(\mathbf{X} + \epsilon \mathbf{U} \mathbf{V}^T; \xi) - F(\mathbf{X} - \epsilon \mathbf{U} \mathbf{V}^T; \xi)}{2\epsilon} (\mathbf{U} \mathbf{V}^T / r).$$

LOZO is essentially a zeroth-order subspace optimization method

Assumption 4.1. For any ξ , the function $F(\mathbf{X}; \xi)$ is differentiable with respect to \mathbf{X} . Furthermore,

- The gradient $\nabla F(\mathbf{X}; \xi)$ is uniformly L -Lipschitz continuous, i.e., $\forall \mathbf{X}, \mathbf{Y}$,

$$\|\nabla F(\mathbf{X}; \xi) - \nabla F(\mathbf{Y}; \xi)\| \leq L\|\mathbf{X} - \mathbf{Y}\|, \quad \forall \xi,$$

where $\|\mathbf{X}\| := \sqrt{\sum_{\ell=1}^{\mathcal{L}} \|X_\ell\|_F^2}$ for any $\mathbf{X} = \{X_\ell\}_{\ell=1}^{\mathcal{L}}$.

- The stochastic gradient is unbiased and has bounded variance, i.e., $\forall \mathbf{X}$,

$$\mathbb{E}[\nabla F(\mathbf{X}; \xi)] = \nabla f(\mathbf{X}) \quad \text{and} \quad \mathbb{E}\|\nabla F(\mathbf{X}; \xi) - \nabla f(\mathbf{X})\|^2 \leq \sigma^2.$$

Assumption 4.2. The random matrix $\mathbf{V} = \{V_\ell\}_{\ell=1}^{\mathcal{L}}$ is drawn from a distribution such that $V_\ell^T V_\ell = n_\ell I$ and $\mathbb{E}[V_\ell V_\ell^T] = r_\ell I$ for each ℓ .

Theorem 4.4. Under Assumptions 4.1 and 4.2, and letting $T = K\nu$, with suitable choices of α and ϵ , the sequence of the $k\nu$ -th variables $\{\mathbf{X}^{k\nu}\}$ generated by LOZO converges at the following rate:

$$\frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E} \|\nabla f(\mathbf{X}^{k\nu})\|^2 \leq O \left(\sqrt{\frac{\Delta_0 L \tilde{d} \sigma^2}{T}} + \frac{\Delta_0 L d \nu}{T} \right),$$

where $\Delta_0 := f(\mathbf{X}^0) - f^*$, $\tilde{d} = \sum_{\ell=1}^L (m_\ell n_\ell^2 / r_\ell)$ and $d = \sum_{\ell=1}^L m_\ell n_\ell$.

LOZO learns the full-parameter even if we sample low-rank perturbations

LOZO converges at rate $O(1/\sqrt{T})$, the same rate as vanilla first-order SGD

Memory-efficient momentum LOZO

Can we develop memory-efficient zeroth-order momentum SGD?



- Momentum can significantly improve the convergence rate.

$$\mathbf{M}^t = \beta \mathbf{M}^{t-1} + (1 - \beta) \hat{\nabla} F(\mathbf{X}^t; \xi^t), \quad \mathbf{X}^{t+1} = \mathbf{X}^t - \alpha \mathbf{M}^t,$$

But it incurs significant memory cost since M has the same dimension as ∇F

- However, we have $\nabla F = UV^T$ in LOZO. This structure can help us save memory

$$\begin{aligned}\mathbf{M}^t &= \beta \mathbf{M}^{t-1} + (1 - \beta) \mathbf{U}^t (\mathbf{V}^{(k)})^\top \\ \mathbf{X}^{t+1} &= \mathbf{X}^t - \alpha \mathbf{M}^t\end{aligned}$$

Store M ; Memory expensive (m,n)



$$\begin{aligned}\mathbf{N}^t &= \beta \mathbf{N}^{t-1} + (1 - \beta) \mathbf{U}^t \\ \mathbf{X}^{t+1} &= \mathbf{X}^t - \alpha \mathbf{N}^t (\mathbf{V}^{(k)})^\top\end{aligned}$$

Store N ; Memory efficient (m, r)



PART 04

Experiments

Memory consumption: RoBERTa-large

Algorithm	MNLI		SNLI	
	Accuracy (%)	Memory Usage (GB)	Accuracy (%)	Memory Usage (GB)
LOZO	61.6	2.83	73.4	2.83
LOZO-M	62.7	2.84	74.0	2.84
MeZO	56.7	3.00	68.5	3.00
MeZO-M	58.9	5.89	69.6	5.89
MeZO-Adam	62.6	7.42	72.7	7.42

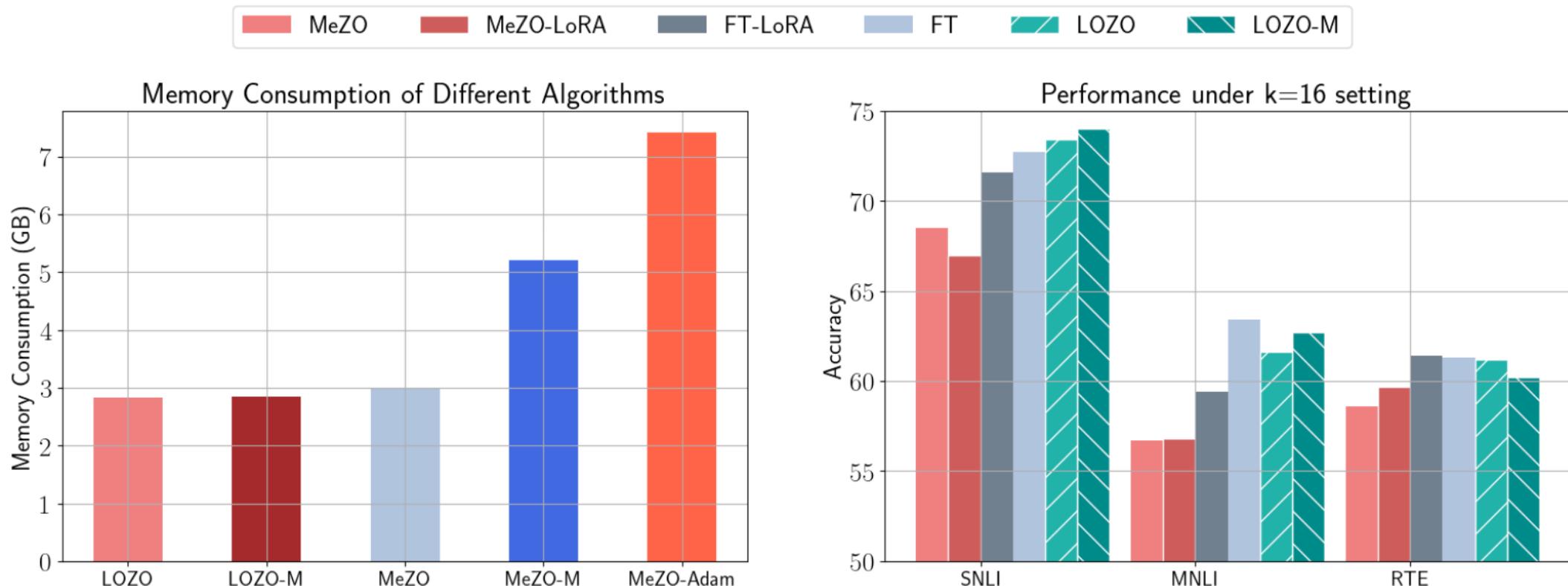
- LOZO outperforms MeZO in performance due to the capture of low-rank structure
- LOZO-M outperforms LOZO in performance; demonstrating the necessity to involve momentum
- LOZO-M is significantly more memory-efficient than MeZO-M due to the low-rank momentum

Experiments

Task	LLaMA-7B		LLaMA-70B	
	Memory	Consumed GPUs	Memory	Consumed GPUs
LOZO	14.1 GB	1× A800	135.5 GB	2× A800
MeZO	14.3 GB	1× A800	136.0 GB	2× A800
FT-LoRA	32.7 GB	1× A800	187.2 GB	3× A800
FT	281.6 GB	4× A800	640 + GB	> 8× A800

Table 12: Comparison of memory costs for LOZO, MeZO, and gradient-based methods on LLaMA models of varying scales for the MultiRC task with a per-device batch size of 1. Due to limited computational resources, the results for FT on LLaMA-70B are approximate.

Numerical performance



Left: GPU memory usage comparison between LOZO and MeZO with their respective variants. Right: Performance of MeZO, MeZO-LoRA, FT, FT-LoRA, LOZO, and LOZO-M on RoBERTa-large across three tasks (SNLI, MNLI, and RTE).

Experiments

Outperforms MeZO in most tasks

Task	SST-2	RTE	CB	BoolQ	WSC	WiC	MultiRC	COPA	ReCoRD	SQuAD	DROP
Zero-shot	58.8	59.6	46.4	59.0	38.5	55.0	46.9	80.0	81.0	46.2	14.6
ICL	87.0	62.1	57.1	66.9	39.4	50.5	53.1	87.0	82.3	75.9	29.5
MeZO	91.3	68.2	66.1	68.1	61.5	59.4	59.4	88.0	81.3	81.8	31.3
MeZO-LoRA	89.6	67.9	67.8	73.5	63.5	60.2	61.3	84.1	81.5	82.1	31.3
LOZO	91.7	70.4	69.6	71.9	63.5	60.8	63	89.0	81.3	84.9	30.7
FT	91.8	70.9	84.1	76.9	63.5	70.1	71.1	79.0	74.1	84.9	31.3

Table 2: Experiments on OPT-13B (with 1000 examples). ICL: in-context learning; FT: full fine-tuning with Adam. The best results are shown in **bold** except for FT.

Experiments

Tests with larger models

Task	SST-2	RTE	BoolQ	WSC	WiC	SQuAD
30B zero-shot	56.7	52.0	39.1	38.5	50.2	46.5
30B ICL	81.9	66.8	66.2	56.7	51.3	78.0
30B MeZO	90.7	64.3	68.2	63.5	56.3	86.1
30B LOZO	92.8	65.3	72.3	64.4	57.2	85.6
66B zero-shot	57.5	67.2	66.8	43.3	50.6	48.1
66B ICL	89.3	65.3	62.8	52.9	54.9	81.3
66B MeZO	92.0	71.5	73.8	64.4	57.8	84.0
66B LOZO	92.5	74.0	74.5	63.5	59.4	85.8

Table 3: Experiments on OPT-30B and OPT-66B on SuperGLUE benchmark. Our results show that LOZO is superior on most tasks compared to the other baselines. The best results are shown in **bold**.

Experiments

Training losses

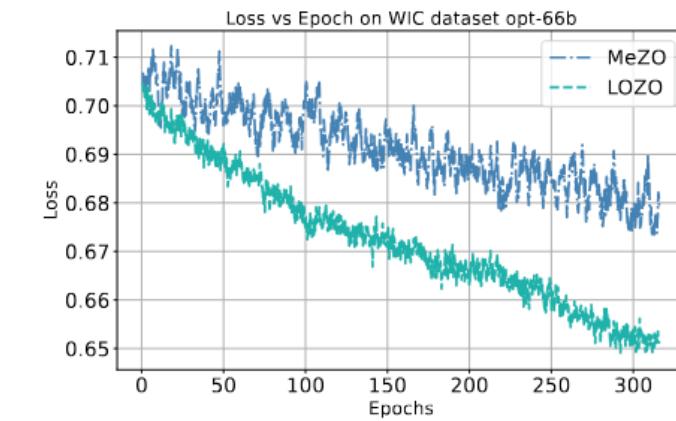
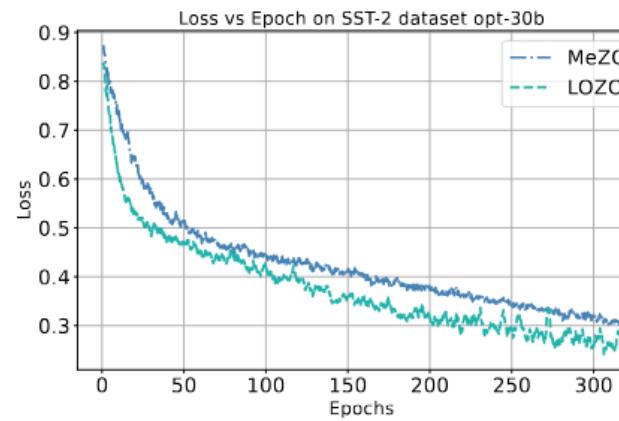
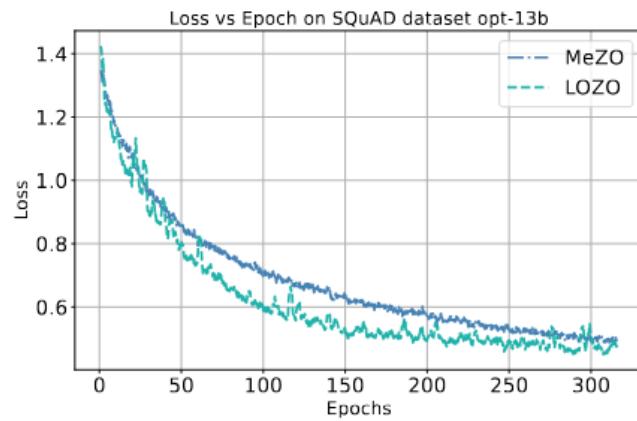


Figure 3: *Left:* Loss curves of OPT-13B on SQuAD dataset. *Middle:* Loss curves of OPT-30B on SST-2 dataset. *Right:* Loss curves of OPT-66B on WIC dataset.

Summary

- Activation dominates memory when batch size or sequence length is large
- Zeroth-order optimization can save activation
- MeZO cannot capture the low-rank structure in LLM gradients; we propose LOZO as an effective recipe
- LOZO outperforms MeZO and has strong theoretical guarantees
- LOZO-M is the first memory-efficient zeroth-order momentum optimizer

**Take-home
message**

Low-rank structures are critical in LLMs pretraining and finetuning