Unlocking LoRA's Capabilities for Fine-tuning Large Language Models with Only Forward Pass Supplementary Material

Ziyi Wang[1], Yi Du[1], Shengcheng Ye[1], Sunzhe Yang[1], Faming Fang[1], and Guixu Zhang[1]

[1]East China Normal University

A The complete proof of $\mathrm{Var}(\Delta W) \propto (A^2 + B^2)$

The weight update of LoRA can be formulated as:

$$\Delta W = (A + \Delta A) \cdot (B + \Delta B) - A \cdot B = \Delta A \cdot B + A \cdot \Delta B + \Delta A \cdot \Delta B, \tag{1}$$

where ΔW is the change in weights, A and B are the two matrices within the LoRA module, ΔA and ΔB are the changes in parameters A and B, respectively. In practical applications, the higher-order term $\Delta A \cdot \Delta B$ is usually small and can be neglected. Therefore, the weight change can be approximated as:

$$\Delta W \approx \Delta A \cdot B + A \cdot \Delta B. \tag{2}$$

We calculate the variance of ΔW:

$$\mathrm{Var}(\Delta W) = \mathrm{Var}(\Delta A \cdot B + A \cdot \Delta B). \tag{3}$$

The formula for estimating the gradient by the Random Gradient Estimator (RGE) can be written as:

$$\hat{\nabla} f(\mathbf{x}) = \frac{1}{q} \sum_{i=1}^{q} \left[\frac{f(\mathbf{x} + \mu \mathbf{u}_i) - f(\mathbf{x} - \mu \mathbf{u}_i)}{2\mu} \mathbf{u}_i \right] \quad \text{(RGE)}. \tag{4}$$

We can write the formulas for ΔA and ΔB:

$$\Delta A = \delta[\frac{f(\mathbf{x} + \mu \mathbf{u}) - f(\mathbf{x} - \mu \mathbf{u})}{2\mu} \mathbf{u}_A] = \delta[g(\mathbf{u}) \cdot \mathbf{u}_A] \tag{5}$$

$$\Delta B = \delta[\frac{f(\mathbf{x} + \mu \mathbf{u}) - f(\mathbf{x} - \mu \mathbf{u})}{2\mu} \mathbf{u}_B] = \delta[g(\mathbf{u}) \cdot \mathbf{u}_B] \tag{6}$$

Under the conditions of RGE, μ is an extremely small value, and in this case, the value of $g(u)$ depends on the properties of the function f. Therefore, $g(u)$ is independent of $\mathbf{u}_A$ and $\mathbf{u}_B$. Meanwhile, since $\mathbf{u}_A$ and $\mathbf{u}_B$ are independent, ΔA and ΔB are independent. We have:

$$\begin{aligned} \mathrm{Var}(\Delta W) &= \mathrm{Var}(\Delta A \cdot B + A \cdot \Delta B) \\ &= \mathrm{Var}(\Delta A \cdot B) + \mathrm{Var}(A \cdot \Delta B) \\ &= B^2 \cdot \mathrm{Var}(\Delta A) + A^2 \cdot \mathrm{Var}(\Delta B). \end{aligned}$$

When fine-tuning the SST2 task with the OPT-1.3B model, we empirically have:

$$\begin{cases} \mathbb{E}\left[f(\mathbf{x} + \mu \mathbf{u}) - f(\mathbf{x} - \mu \mathbf{u})\right] = 0 \\ \mathrm{Var}(f(\mathbf{x} + \mu \mathbf{u}) - f(\mathbf{x} - \mu \mathbf{u})) = 0.004 \\ \mu = 0.01 \\ \delta = 0.0001 \\ \mathbf{u} \sim \mathcal{N}(0, 1) \end{cases}$$

As a result, we get the Var(ΔA) and Var(ΔA):

$$\begin{cases} \text{Var}(\Delta A) = 1 \times 10^{-7} \\ \text{Var}(\Delta B) = 1 \times 10^{-7} \end{cases}$$

In other tasks, the value of Var($f(\mathbf{x} + \mu\mathbf{u}) - f(\mathbf{x} - \mu\mathbf{u})$) may undergo slight changes, but it does not affect our conclusion:

$$\text{Var}(\Delta W) \propto (A^2 + B^2). \quad (7)$$

B Experimental details

As in previous work, we use RGE as the zeroth-order gradient estimator with $q = 1$ (see Equation 4).

B.1 Datasets and hyperparameters

For RoBERTa-large, we evaluate the performance on five NLP tasks: SST-2, SST-5, SNLI, MNLI and RTE. We adopt two settings: $k = 16$ and $k = 512$, which require 16 and 512 examples per class, respectively, during both the training and validation stages. For the selection of training hyperparameters, please refer to Table 1.

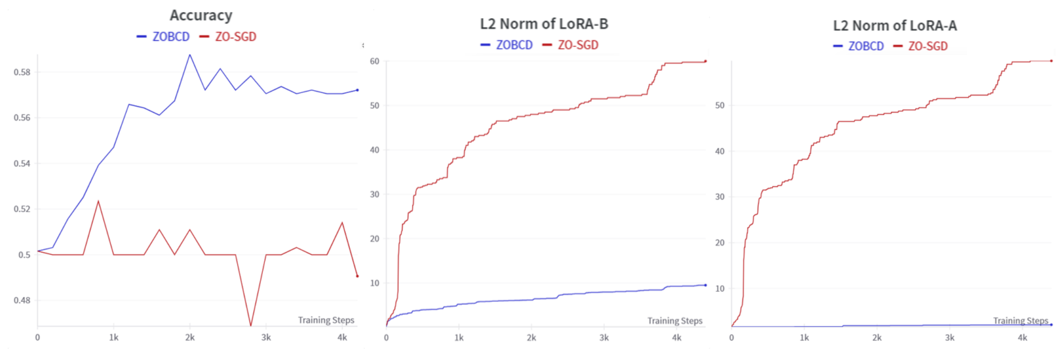
Table 1: Hyperparameters for different experiments of RoBERTa-large.

Experiment	Hyperparameters	Values
MeZO	Batch size	64
	Learning rate	{1e-7, 1e-6, 1e-5, 1e-4}
	ϵ	1e-3
MeZO-LoRA	Batch size	64
	Learning rate	{1e-5, 5e-5, 1e-4}
	r	8
	ϵ	{1e-3, 1e-2}
HiZOO-LoRA	Batch size	64
	Learning rate	{1e-5, 5e-5, 1e-4}
	r	8
	ϵ	{1e-3, 1e-2}
ZOBCD-LoRA	Batch size	64
	Learning rate	1e-4
	r	8
	ϵ	1e-2
FT-LoRA	Batch size	8
	Learning rate (r, α)	{1e-4, 3e-4, 5e-4, 1e-3}
	r	8

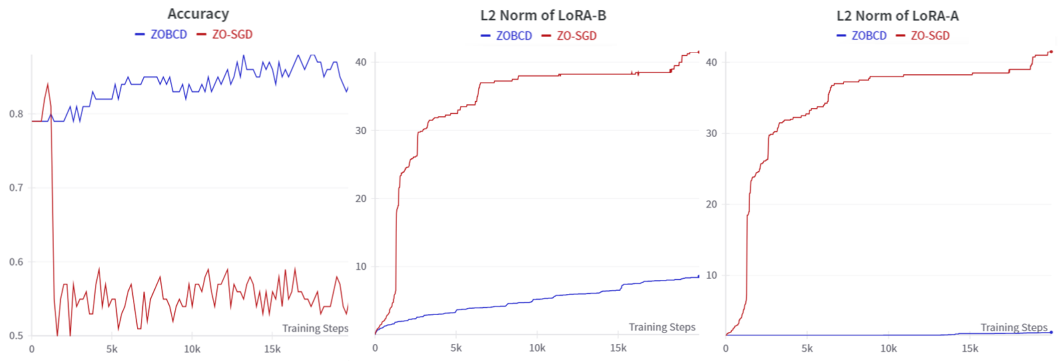
For large auto-regressive language models, we conduct experiments on the following datasets: SST-2, RTE, WSC, WIC, Copa, CB. We train using half precision (float16 or bfloat16 depending on the model). For the selection of training hyperparameters, please refer to Table 2.

B.2 More visualization comparisons of training curves.

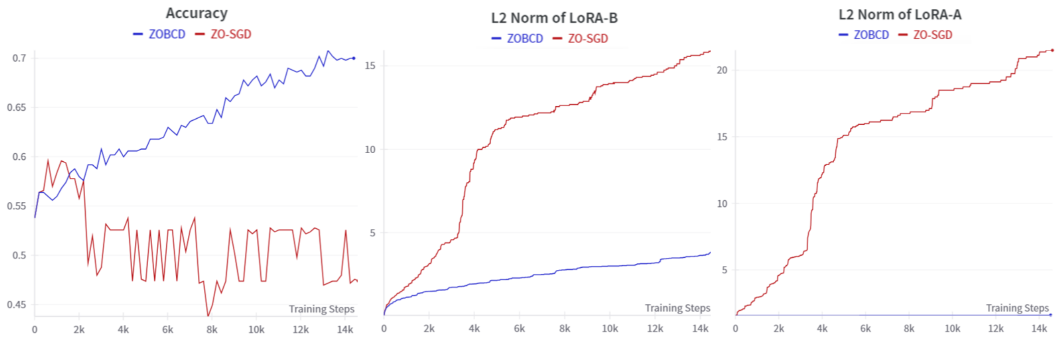
As shown in Figure 1, we provide additional visualizations of training data, most of which come from the relevant experiments in the main text's tables. We demonstrate the effectiveness of our method and its ability to effectively suppress abnormal surges in weight magnitude through a multitude of visualized training curves. It can be observed that when training LoRA with traditional ZO-SGD, the issue of training collapse is prevalent, accompanied by a surge in the magnitude of LoRA's A and B matrices. Our ZOBCD method mitigates the problem of the sharp increase in LoRA weight magnitude by stabilizing the training process and significantly enhances the performance of fine-tuning.



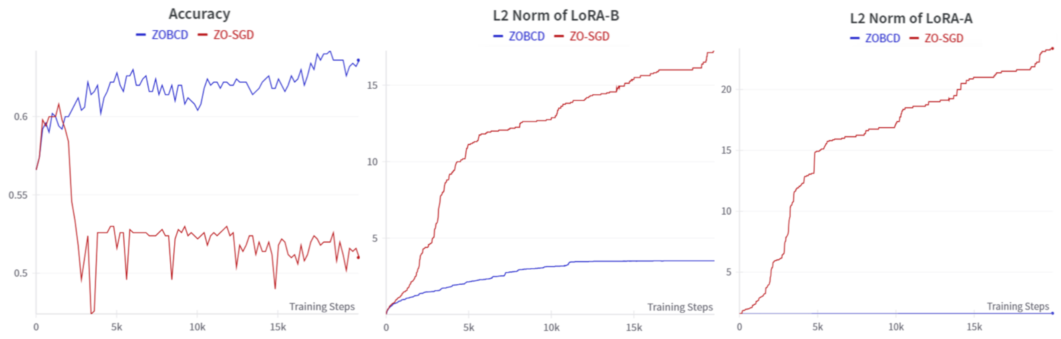
(a) Fine-tuning Llama2-7B model on WIC task with 1000 samples



(b) Fine-tuning Llama2-7B model on Copa task with 1000 samples



(c) Fine-tuning Llama3-8B model on WIC task with 1000 samples



(d) Fine-tuning Llama3-8B model on WSC task with 1000 samples

Figure 1: More visualization comparisons of training curves.

Table 2: Hyperparameters for different experiments of large auto-regressive language models.

Experiment	Hyperparameters	Values
MeZO	Batch size	16
	Learning rate	{1e-7, 1e-6, 1e-5, 1e-4}
	ϵ	1e-3
MeZO-LoRA	Batch size	16
	Learning rate	{1e-5, 5e-5, 1e-4}
	r	8
	ϵ	{1e-3, 1e-2}
HiZOO-LoRA	Batch size	16
	Learning rate	{1e-5, 5e-5, 1e-4}
	r	8
	ϵ	{1e-3, 1e-2}
ZOBCD-LoRA	Batch size	16
	Learning rate	1e-4
	r	8
	ϵ	1e-2
FT-LoRA	Batch size	8
	Learning rate (r, α)	{1e-4, 3e-4, 5e-4, 1e-3}
	r	8

B.3 Detailed numerical results of the Figure 5 in the main text

We verify the effectiveness of our proposed ZOBCD method's distinct training strategies for the LoRA matrices in Figure 5 in the main text, and the detailed numerical results is shown in Table 3.

Table 3: Detailed numerical results of the Figure 5 in the main text

Algorithm	$k = 512$			$k = 16$		
	SNLI	**MNLI**	**RTE**	**MNLI**	**SNLI**	**RTE**
MeZO-LoRA	73.4	65.3	72.5	67.1	57.5	59.3
ZOBCDs1-LoRA	77.7	67.5	74.7	68.9	59.3	63.2
ZOBCDs2-LoRA	79.7	66.4	74.7	69.6	60.8	65.1
ZOBCDs3-LoRA	80.5	68.1	75.1	70.8	60.1	64.5
ZOBCD-LoRA	82.1	69.7	76.3	71.8	62.2	66.0
LoRA	85.0	73.4	81.2	77.2	67.7	66.6

C Ablation study of BCD

In Section 4.4 of the main text, we mentioned that: We also conduct an ablation study on the block strategy of BCD as a whole. We compare our ZOBCD block strategy based on the LoRA matrices with the block strategy based on magnitude and the random subspace block strategy. In this section, we will provide a detailed explanation of our experiments.

The first step of the block coordinate descent (BCD) algorithm involves partitioning the existing parameters. We hope to reduce the scale of parameters optimized at a single time through the iterative optimization of the BCD method. At the same time, if the partitioned parameters have different properties, BCD can also conveniently provide different training parameters for different parameters.

C.1 Magnitude-based gradient-selective updating block coordinate descent (MGS-BCD)

We first introduce the magnitude-based gradient-selective updating block coordinate descent algorithm (MGS-BCD). Selecting parameters based on the output magnitude of neuron or block is a common

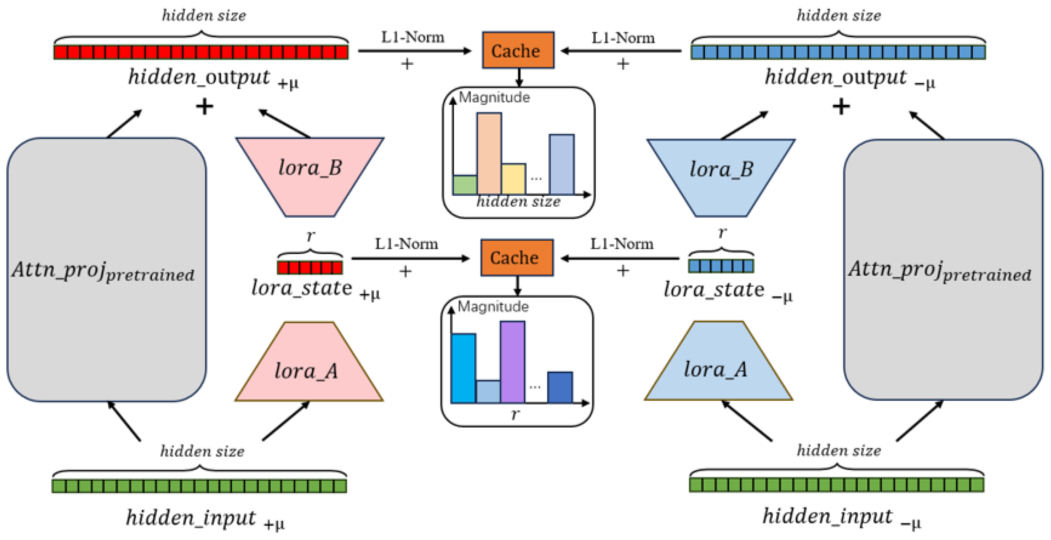


Figure 2: Magnitude-based gradient-selective updating.

Table 4: Comparative experiments on two different models training with LoRA.

Model	Method	Type	SST2	RTE	CB	WSC	WIC	COPA	Average
Llama2-7B	MeZO	ZO	85.7	59.2	72.1	52.6	52.3	84.0	67.7
	RGS-BCD	ZO	85.9	59.6	72.5	52.8	54.5	85.0	68.4
	MGS-BCD	ZO	90.4	59.7	72.9	53.2	54.7	85.0	69.3
	LoRA-BCD	ZO	92.0	60.3	73.6	53.8	56.1	87.0	70.5
Llama3-8B	MeZO-LoRA	ZO	88.4	77.6	75.0	61.2	59.4	91.0	75.4
	RGS-BCD	ZO	88.6	77.9	77.2	62.0	59.6	91.0	76.1
	MGS-BCD	ZO	90.4	82.5	82.8	62.6	62.7	91.0	78.7
	LoRA-BCD	ZO	92.6	86.2	87.5	64.2	65.2	91.0	81.1

strategy in model pruning. Those parameters with larger output magnitudes are more relevant to task prediction, demonstrating more significant importance during training. MGS-BCD records the output of every LoRA-A and LoRA-B, and selects parameters of trainable LoRA by magnitude of outputs and a hyperparameter threshold rate s. The selected parameters' gradient is to be update while the others masked. MGS-BCD applies alternating optimization on different parameters grouped with magnitude. The overview of magnitude-based selecting is shown in Figure 2. For parameters with larger output magnitudes, use a larger learning rate and more training steps during optimization; for parameters with smaller output magnitudes, use a smaller learning rate and fewer training steps. The complete algorithm process is presented in Algorithm 1.

C.2 Random subspace gradient-selective updating block coordinate descent (RGS-BCD)

Random subspace gradient-selective updating is another skill to reduce the scale of optimization, and it has been applied to ZO optimization by recent work as we mention in the Related Work of the main text. Different from MGS-BCD, RGS-BCD randomly groups the LoRA parameters and applies alternating optimization on each groups of parameters. The complete algorithm process is presented in Algorithm 2.

C.3 Performance comparison among variants of the BCD algorithm.

In this section, we compare our proposed ZOBCD-LoRA method (denoted here as LoRA-BCD) with MGS-BCD and RGS-BCD, as shown in Table 4. BCD methods effectively improve the fine-tuning performance, and our proposed ZOBCD-LoRA works best.

Algorithm 1 Magnitude-based gradient-selective updating block coordinate descent (MGS-BCD)

```
Require: parameters $\theta \in \mathbb{R}^d$, loss $\mathcal{L}$, step budget $T$, perturbation scale $\epsilon = 0.01$, batch size $B$,
    learning rate schedule $\{\eta_t^{(A)} = 1e-4, \eta_t^{(B)} = 2e-4\}$, threshold rate $s$
 1: Split $\theta$ into blocks $A$ and $B$
 2: Initialize output magnitudes $M_A$ and $M_B$ for blocks $A$ and $B$
 3: for $t = 1, ..., T$ do
 4:     Sample batch $\mathcal{B}$ and seed $s$
 5:     for each block $X \in \{A, B\}$ do
 6:         Compute output magnitudes $M_X$
 7:         Select parameters $P_X$ based on $M_X$ and threshold rate $s$
 8:         Set learning rate $\eta_t^{(X)}$ and number of steps $n_t^{(X)}$ based on $M_X$
 9:     end for
10:     for $i = 1, ..., n_t^{(B)}$ do
11:         B ← PerturbParameters(B, $\epsilon$, s, 1)
12:         $\ell_+ \leftarrow \mathcal{L}(A, B; \mathcal{B})$
13:         B ← PerturbParameters(B, $-2\epsilon$, s, 1)
14:         $\ell_- \leftarrow \mathcal{L}(A, B; \mathcal{B})$
15:         B ← PerturbParameters(B, $\epsilon$, s, 1)
16:         projected_gradB ← $(\ell_+ - \ell_-)/(2\epsilon)$
17:         Update B: $b_j \leftarrow b_j - \eta_t^{(B)} *$ projected_gradB $* z$, $z \sim \mathcal{N}(0, 1)$
18:     end for
19:     for $i = 1, ..., n_t^{(A)}$ do
20:         A ← PerturbParameters(A, $\epsilon$, s, 0.25)
21:         $\ell_+ \leftarrow \mathcal{L}(A, B; \mathcal{B})$
22:         A ← PerturbParameters(A, $-2\epsilon$, s, 0.25)
23:         $\ell_- \leftarrow \mathcal{L}(A, B; \mathcal{B})$
24:         A ← PerturbParameters(A, $\epsilon$, s, 0.25)
25:         projected_gradA ← $(\ell_+ - \ell_-)/(2\epsilon)$
26:         Update A: $a_i \leftarrow a_i - \eta_t^{(A)} *$ projected_gradA $* z$, $z \sim \mathcal{N}(0, 0.25)$
27:     end for
28: end for
29: return $\theta = \{A, B\}$
30: procedure PERTURBPARAMETERS($\theta, \epsilon, s, \delta$)
31:     Reset random seed $s$
32:     for $\theta_i \in \theta$ do
33:         $a_i \leftarrow a_i + \epsilon z$, $z \sim \mathcal{N}(0, \delta)$
34:     end for
35: end procedure
```

C.4 Discussion about the reason ZOBCD-LoRA performs the best

Recent works have shown through experiments and theoretical analysis that LoRA's A and B matrices exhibit different characteristics during training. LoRA+ has proved that during training (as we mention in the main text), using the same update step size for the A and B matrices cannot make the linear terms of contribution in the model output updates, which are caused by A or B, equal to $\Theta(1)$. This means that the A and B matrices of LoRA have unequal status during training and LoRA+ suggests using a training step size for matrix A that is much smaller than that for matrix B. The research findings of LoRA+ demonstrate that during the fine-tuning of LoRA, matrix A requires more fine-grained optimization behavior, and the B matrix requires more updates to adapt to the changes in the A matrix. In first-order optimization, even if this property is ignored, the backpropagation mechanism and stable optimizer algorithms can offset the instability from identical update steps, limiting the adverse impact to suboptimal convergence speed. However, when the training optimizer becomes a zeroth-order optimizer that relies solely on forward passes, the characteristics of the LoRA structure can greatly affect the stability of training.

Current ZO methods not only have the same issue as FO methods regarding the identical update step size for matrices A and B, but also have the risk of falling into the high variance of gradient direction

Algorithm 2 Random subspace gradient-selective updating block coordinate descent (RGS-BCD)

```
Require: parameters $\theta \in \mathbb{R}^d$, loss $\mathcal{L}$, step budget $T$, perturbation scale $\epsilon = 0.01$, batch size $B$
 1: Randomly split $\theta$ into two groups $G_1$ and $G_2$
 2: for $t = 1, ..., T$ do
 3:     Sample batch $\mathcal{B}$ and seed $s$
 4:     $G_1 \leftarrow \text{PerturbParameters}(G_1, \epsilon, s, 1)$
 5:     $\ell_+ \leftarrow \mathcal{L}(G_1, G_2; \mathcal{B})$
 6:     $G_1 \leftarrow \text{PerturbParameters}(G_1, -2\epsilon, s, 1)$
 7:     $\ell_- \leftarrow \mathcal{L}(G_1, G_2; \mathcal{B})$
 8:     $G_1 \leftarrow \text{PerturbParameters}(G_1, \epsilon, s, 1)$
 9:     projected_gradG1 $\leftarrow (\ell_+ - \ell_-)/(2\epsilon)$
10:     Update $G_1$: $g_{1j} \leftarrow g_{1j} - \eta_t *$ projected_gradG1 $* z$, $z \sim \mathcal{N}(0, 1)$
11:     $G_2 \leftarrow \text{PerturbParameters}(G_2, \epsilon, s, 0.25)$
12:     $\ell_+ \leftarrow \mathcal{L}(G_1, G_2; \mathcal{B})$
13:     $G_2 \leftarrow \text{PerturbParameters}(G_2, -2\epsilon, s, 0.25)$
14:     $\ell_- \leftarrow \mathcal{L}(G_1, G_2; \mathcal{B})$
15:     $G_2 \leftarrow \text{PerturbParameters}(G_2, \epsilon, s, 0.25)$
16:     projected_gradG2 $\leftarrow (\ell_+ - \ell_-)/(2\epsilon)$
17:     Update $G_2$: $g_{2k} \leftarrow g_{2k} - \eta_t *$ projected_gradG2 $* z$, $z \sim \mathcal{N}(0, 0.25)$
18: end for
19: return $\theta = \{G_1, G_2\}$
20: procedure PERTURBPARAMETERS($\theta, \epsilon, s, \delta$)
21:     Reset random seed $s$
22:     for $\theta_i \in \theta$ do
23:         $a_i \leftarrow a_i + \epsilon z$, $z \sim \mathcal{N}(0, \delta)$
24:     end for
25: end procedure
```

estimation. Specifically, traditional RGE estimates the gradient by applying a small perturbation $\mu \boldsymbol{Z}$ to the tunable parameters in both positive and negative directions, and this perturbation will be equally applied to LoRA's A and B matrices in every forward pass. In the initial phase of training, the adverse effects of uniform perturbations on LoRA's A and B matrices are relatively minor, because $B \approx 0$ (The B matrix is initialized to 0 to ensure that the output of the LoRA module is 0), and from Equation 6 in the main text it follows $\mathrm{Var}(\Delta W) = A^2 \cdot \mathrm{Var}(\Delta B)$, meaning the perturbation is mainly dominated by LoRA's B matrix. As training progresses, the values of B increase, causing the LoRA to take effect in the fine-tuning task. Correspondingly, the perturbations to LoRA's A matrix have an increasingly significant impact on the gradient of predictions, and since the A matrix is more sensitive than the B matrix, **it can easily trigger instability in the overall weight optimization direction**. Readers might wonder at this point whether, if the value of μ is particularly small, the greater sensitivity of matrix A might not necessarily lead to significant instability. In fact, theoretically in zeroth-order optimization, the smaller the value of μ, the more precise the gradient estimation. However, when applied to the gradient estimation of large models, the value of μ is often relatively large (empirically, taking 0.01 yields better results), otherwise it may fail to induce changes in the loss, leading to vanishing gradients or slower convergence and getting trapped in local optima.

This instability optimization direction is likely a significant cause of the surge in the $L2$ norms of LoRA's A and B matrices. The recent work DoRA found that LoRA matrices generate a certain synergistic effect during the training process (as we mention in the main text), which is specifically manifested in a positive correlation between the **amplification of weight magnitudes** and **changes in the training direction**. This is consistent with the phenomena we have observed.