Zoop it! Efficient Zero-Order Optimization with Output Perturbation

Anonymous Authors[1]

Abstract

Zero-order gradient methods offer a promising alternative for finetuning large models with constrained computational resources by estimating the gradient without full backpropagation. Recent approaches employ an efficient finite difference method that uses random noise perturbations across all model weights for gradient estimation. However, this approach suffers from high variance and slow convergence, as the number of parameters can be enormous. In a deep network that consists of a cascade of layers, instead of perturbing the entire model weights with random noises, one can alternatively only perturb the activation to compute a gradient estimation only for that activation layer, and backprop from there on to get a gradient estimation for the entire layer. This method reduces variance with the increase of batch sizes. Despite the potential advantages, efficient implementation of this technique for large model training remains unexplored. We introduce a new method, called Zero-order Optimization with Output Perturbation (Zoop), which can be easily integrated into existing network architectures. Zoop enables selective perturbation control to balance memory efficiency and gradient accuracy. We evaluated Zoop on autoregressive and masked language models up to 66 billion parameters across various tasks such as classification, multiple-choice, and text generation. Our findings show that Zoop effectively fine-tunes large language models with fewer updates than current methods, demonstrating its efficiency and effectiveness in model optimization.

1. Introduction

Efficiently fine-tuning large language models (LLMs) poses substantial challenges due to the high memory requirements associated with their size. For instance, applying gradient descent with AdamW (Loshchilov & Hutter, 2017) demands GPU memory approximately three times the model's weight parameters plus the layer-wise activations necessary for backpropagation.[1]

To address the memory constraints, Parameter-Efficient Fine-Tuning Methods (PEFT), such as prefix-tuning (Li & Liang, 2021) and LoRA (Hu et al., 2021), freeze most parameters of a pretrained model and fine-tune only a subset to reduce GPU memory usage. However, this constraint on parameter updates limits their fine-tuning performance. Zeroth-order gradient methods offer an alternative by enabling full model fine-tuning through estimating gradients without backpropagation (Malladi et al., 2024; Zhang et al., 2024), thus bypassing the need to store layer activations. Recent advancements in zeroth-order methods have reduced memory requirements to allow fine-tuning of models with up to 30 billion parameters on a single Nvidia A100 GPU (Malladi et al., 2024; Zhang et al., 2024).

Despite these benefits, zeroth-order methods face high variance in gradient estimation, which leads to slow convergence. These methods are often based on the Simultaneous Perturbation Stochastic Approximation (SPSA) (Spall, 1992), which estimates the true gradients by applying random noise perturbations to weights, resulting in gradient variance that scales with the number of parameters.

This work introduces an alternative zeroth-order method, Zeroth-order Optimization with Output Perturbation (Zoop), which perturbs the activation instead of the model weights (Ren et al., 2022; Le Cun et al., 1988). The general intuition is that the gradient of any linear layer is structured, as it is the outer product of the output gradient and the input activation. Hence, by perturbing only the activations, Zoop can reduces the variance of the gradient estimate.

In practice, we implement Zoop in a minimal, memory-efficient, and model-agnostic way, by introducing a novel noise layer (the Zoop layer) between every consecutive layer (See Algorithm 2). As a result, the Zoop layer acts as a plug-in module for any pretrained large models. We evaluate

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

[1]While gradient checkpointing can alleviate some of this memory burden by reducing the stored activations to $\mathcal{O}(\sqrt{L})$, where L is the number of layers, this approach remains resource-intensive as L grows.

Algorithm 1: Zoop: Efficient Activation Perturbation

Require: Model parameters $\boldsymbol{\theta} = \{\theta^l\}_{l=1}^L$, Layer-wise function $\Phi = \{\phi^l\}_{l=1}^L$, batch size B, loss function $\mathcal{L}$, number of training steps T, perturbation scale ϵ, step size $\{\eta_t\}_{t=1}^T$.

for $t = 1$ **to** T **do**

  Sample a batch $(\mathbf{x}, \mathbf{y}) \subset \mathcal{D}$ of size B and random seed s

  $\mathbf{y}_+ \leftarrow$ `PerturbActivation`$(\mathbf{x}, \epsilon, s, \boldsymbol{\theta}, \Phi)$

  $\mathbf{y}_- \leftarrow$ `PerturbActivation`$(\mathbf{x}, -\epsilon, s, \boldsymbol{\theta}, \Phi)$

  $\boldsymbol{\alpha} \leftarrow \big(\mathcal{L}(\mathbf{y}_+, \mathbf{y}) - \mathcal{L}(\mathbf{y}_-, \mathbf{y})\big) \,/\, (2\epsilon)$ ▷ `Gradient scaling factor`

  Reset random number generator with seed s.

  for $l = 1$ **to** L **do**

    $\mathbf{x}^l \leftarrow \phi^l(\mathbf{x}^{l-1}; \boldsymbol{\theta}^l)$

    $\boldsymbol{u}^l \leftarrow \boldsymbol{\alpha} \odot (\mathbf{x}^l \odot \mathbf{z}^l)$, where $\mathbf{z}^l \sim \mathcal{N}(0, I)$

    $\boldsymbol{u}^l$`.sum().backward()` ▷ `Local backpropagation`

    $\mathbf{g}^l \leftarrow$ `get_grad`$(\boldsymbol{\theta}^l)$ ▷ `Retrieve gradient for parameter`

    $\mathbf{g}^l \leftarrow \mathbf{g}^l / \|\mathbf{g}^l\|$ ▷ `Normalize gradient`

    $\boldsymbol{\theta}^l \leftarrow \boldsymbol{\theta}^l - \eta_t \times \mathbf{g}^l$

Subroutine `PerturbActivation`$(\mathbf{x}^0, \epsilon, s, \boldsymbol{\theta}, \Phi)$

  Reset random number generator with seed s.

  for $l = 1$ **to** L **do**

    $\mathbf{x}^l \leftarrow \phi^l(\mathbf{x}^{l-1}; \boldsymbol{\theta}^l)$

    $\mathbf{x}^l \leftarrow \mathbf{x}^l + \epsilon \mathbf{z}^l$, where $\mathbf{z}^l \sim \mathcal{N}(0, I)$

  return $\mathbf{x}^L$

Algorithm 2: Noise Layer Implementation in PyTorch-like Pseudocode.

```
# Activation perturbation
def perturb_activations(self, seed, epsilon):
    # seed: int
    # epsilon: float
     x + epsilon * torch.randn(x.size())
     return x
# Update with estimated gradient
def update_gradients(self, seed, epsilon, alpha,
model, optimizer):
    # seed: int
    # epsilon: float
    # alpha: (batch,)
    # 1.  Local backward propagation
     loss = ((x * torch.randn(x.size()) *
alpha).sum()
     loss.backward()
    # 1.  Update parameters with estimated gradients
     for param in model.parameters():
         param.grad = param.grad /
torch.norm(param.grad)
     return x
```

Zoop on both autoregressive and masked language models with up to 66 billion parameters across a range of tasks, including classification, multiple-choice, and text generation, demonstrating that Zoop fine-tunes models more efficiently than prior methods. Our contributions are as follows:

- We introduce Zoop, a low-variance fine-tuning approach with activation perturbations.
- We provide a minimal and efficient implementation of Zoop as a modular, memory-efficient optimization layer, which can be seamlessly integrated into any pre-trained model.
- We demonstrate Zoop's effectiveness across various model types, tasks, and scales, and demonstrated Zoop can achieve faster finetuning of LLM compared to existing approaches.

2. Background

When training models parameterized by θ, we need to compute the gradients $\nabla L(\theta)$ of a loss function $L(\theta)$ to perform optimization. While backpropagation computes this gradient exactly, we sometimes need methods that work without access to the exact derivatives, relying only on evaluations of the loss function $L(\theta)$. These are known as *zeroth-order optimization methods*.

A prominent example is the *Simultaneous Perturbation Stochastic Approximation (SPSA)* method. It estimates the gradient using random directional smoothing, which is defined as:

$$\hat{g}^{\text{SP}}_{\epsilon,m}(\theta) := \frac{1}{m}\sum_{i=1}^{m} \frac{L(\theta + \epsilon\xi_i) - L(\theta - \epsilon\xi_i)}{2\epsilon}\xi_i,$$

where $\{\xi_i\}_{i=1}^n$ are independent random direction vectors that has the same size as the model parameter θ (e.g., standard Gaussian $\mathcal{N}(0, I_d)$), $\epsilon > 0$ is a parameter controlling the perturbation strength, and m is the number of noises sampled. The loss L is typically evaluated on a mini-batch n_{batch} of data. This method requires $2m$ evaluations of the loss function L to compute a single gradient estimate. Increasing m generally improves accuracy but increases computational cost. Many applications use $m = 1$ for efficiency (Malladi et al., 2024; Zhang et al., 2024).

A significant challenge with SPSA, especially for models with a large number of parameters (high dimension d), is the *high variance of its gradient estimate*. We provide the following theorem formally quantifies the Mean Squared Error (MSE) between the SPSA estimate $\hat{g}^{\text{SP}}_{\epsilon,m}(\theta)$ and the true gradient $\nabla L(\theta)$:

Theorem 2.1. *Let $g = \nabla L(\theta)$ be the true gradient of dimension d. Then:*

$$\mathbb{E}[||\hat{g}^{SP}_{\epsilon,m} - g||^2] = \frac{d+1}{m}||g||^2,$$

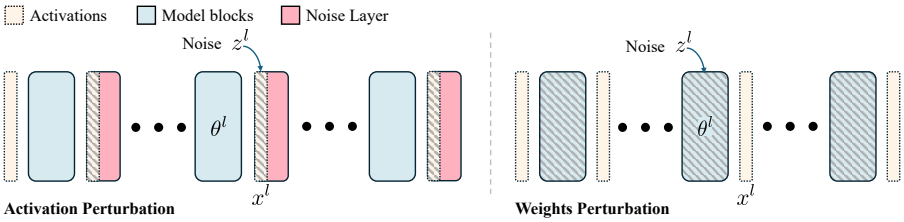

Figure 1: **Comparison of Activation Perturbation and Weight Perturbation Methods.** Left: We introduce the Noise Layers applies noise z^l directly to the activations x^l. The advantage of this method is that it operates in a smaller dimensional space (activations) compared to weight space, potentially leading to lower variance and higher accuracy in gradient estimation. Right: Weight perturbation methods applies noise to the weights θ^l.

$$\textit{and} \quad \mathbb{E}[\cos^2(\hat{g}^{SP}_{\epsilon,m}, g)] \geq \frac{m}{d+m+1}$$

Please refer to Appendix A.2 for the proof. This shows that the error scales proportionally to the model's dimension d and inversely with m, the number of random directions sampled. Such scaling leads to a curse of dimensionality: as d grows, achieving a desired accuracy requires a prohibitively large m (and thus $2m$ function evaluations per estimate), rendering the method inefficient for very large models. The problem is particularly acute when $m = 1$ is used for efficiency, as the error then scales linearly with d.

3. Zero-Order Gradient Estimation via Activation Perturbation for LLMs

Perturbing activations instead of weights presents a compelling strategy for zero-order (ZO) gradient estimation in deep neural networks. This idea, first explored by LeCun et al. (LeCun, 1985), can reduce gradient estimation variance. Our method builds on this by introducing independent random perturbations to activations (instead of weights) for each data point within a mini-batch , yielding gradient estimates whose precision scales favorably with batch size, a crucial aspect for training LLMs.

Layer-wise Gradient Estimation with Activation Perturbation. Consider an LLM composed of L transformer blocks. For the l-th block, parameterized by θ^l, the forward computation is $x^{l+1} = \phi^l(x^l; \theta^l)$, where $x^l \in \mathbb{R}^{d^l_{\text{in}}}$ is the input and $x^{l+1} \in \mathbb{R}^{d^l_{\text{out}}}$ is the output. The overall objective is to minimize a loss function $\mathcal{J}(\theta) = \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} \mathcal{L}(x_i^L)$, where x_i^L is the final output for the i-th data sample.

The gradient of the total loss with respect to the parameters θ^l of an intermediate layer l can be expressed using the chain rule:

$$\nabla_{\theta^l} \mathcal{J}(\theta) = \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} \nabla_{x_i^{l+1}} \mathcal{L}(x_i^{l+1}) \left(\frac{\partial \phi^l(x_i^l; \theta^l)}{\partial \theta^l} \right)^\top \tag{1}$$

Let $J_i^l = \frac{\partial \phi^l(x_i^l; \theta^l)}{\partial \theta^l}$ denote the Jacobian of the l-th layer's output x_i^{l+1} with respect to its parameters θ^l. This J_i^l (a $d^l_{\text{out}} \times \mathtt{d}(\theta^l)$ matrix) can be computed efficiently within the layer module, for example, using forward-mode automatic differentiation or local backpropagation. Let $h_i^l = \nabla_{x_i^{l+1}} \mathcal{L}(x_i^{l+1})$ be the gradient of the downstream loss with respect to the l-th layer's output x_i^{l+1} (a $d^l_{\text{out}} \times 1$ column vector). The primary challenge in ZO optimization is to estimate h_i^l without explicit backpropagation from subsequent layers.

We estimate h_i^l using a random perturbation approach, similar to those used in SPSA approaches (Malladi et al., 2024; Zhang et al., 2024). For each data sample x_i and its corresponding layer output x_i^{l+1}, we generate an independent random perturbation vector $\xi_i \in \mathbb{R}^{d^l_{\text{out}}}$, typically drawn from $\mathcal{N}(0, I)$. The ZO estimator for h_i^l is then:

$$\hat{h}_i^l = \alpha_i \xi_i, \tag{2}$$

where α_i is an estimate of the projection of h_i^l onto ξ_i. Specifically, $\alpha_i \approx (h_i^l)^\top \xi_i$, and is computed using a finite difference scheme:

$$\alpha_i = \frac{\mathcal{L}(x_i^{l+1} + \epsilon \xi_i) - \mathcal{L}(x_i^{l+1} - \epsilon \xi_i)}{2\epsilon}, \tag{3}$$

where $\epsilon > 0$ is a small smoothing parameter. This requires two forward passes from layer $l+1$ using the perturbed activations. Substituting $\hat{h}_i^l$ into Eq. (1), the ZO gradient estimator for θ^l is:

$$\hat{g}_{\theta^l} = \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} (J_i^l)^\top \hat{h}_i^l. \tag{4}$$

Efficient Activation Perturbation with a Noise Layer. In practice, for LLMs composed of stacked transformer blocks, our method perturbs intermediate activations x_i^l at any chosen layer l (for $l \in \{1, \ldots, L\}$) to estimate the corresponding block's gradient $\nabla_{\theta^l} \mathcal{J}(\theta)$. For each data sample x_i in a mini-batch and each targeted layer l, an independent noise vector ξ_i^l is applied to its output activation x_i^{l+1}.

The coefficient $\alpha_i^l \approx ((h_i^l)^\top \xi_i^l)$, required for scaling the noise vector (as per Eq. (2) and Eq. (3)), is estimated using a standard finite difference scheme; this involves two forward passes from layer $l+1$ using the perturbed activations $x_i^{l+1} \pm \epsilon \xi_i^l$. A key aspect of this approach is that by employing $n_{\texttt{batch}}$ independent perturbations across the samples in a mini-batch, we effectively leverage $n_{\texttt{batch}}$ diverse directional queries for constructing the gradient estimate. As will be discussed in the theoretical analysis, this significantly contributes to variance reduction.

Our activation perturbation method, integrated as a conceptual "Noise Layer," is designed for minimal memory overhead and computational efficiency, detailed in Algorithm 1 and Algorithm 2 (which provides PyTorch-like pseudocode for the noise injection and ZO estimation step). The local Jacobians $J_i^l = \frac{\partial \phi^l(x_i^l;\theta^l)}{\partial \theta^l}$ are computed via localized automatic differentiation (forward or reverse mode) confined to each module. This is computationally inexpensive as individual modules are typically much smaller than the entire network for a memory usage analysis). This modularity is particularly advantageous for LLMs. Following practices like those in (Malladi et al., 2024), random seeds for generating perturbations ξ_i^l can be managed to ensure reproducibility or minimize generation overhead. Our approach thus aims for memory efficiency comparable to leading ZO methods like MeZO, while benefiting from the variance reduction inherent in activation perturbation.

Theoretical Analysis of Variance Reduction. A key theoretical advantage of activation perturbation is the reduced dimensionality dependence of the gradient estimator's variance. As detailed in Appendix A.3 (and summarized in Lemma A.3), the Mean Squared Error (MSE) of the estimated layer gradient $\hat{g}_{\theta^l}$ scales as $\mathcal{O}(d_{\text{out}}^l / n_{\texttt{batch}})$, where d_{out}^l is the dimension of the perturbed activation at the output of layer l. This contrasts favorably with conventional weight perturbation methods, where the MSE typically scales with $\texttt{d}(\theta^l)$, the number of parameters in layer l. This MSE scaling highlights a significant benefit: the variance of our gradient estimator decreases proportionally to $1/n_{\texttt{batch}}$, i.e., as the batch size $n_{\texttt{batch}}$ increases. This property is particularly valuable for training large models on large datasets, allowing for more stable and reliable gradient estimates with larger batch sizes. We provide detailed analysis in Appendix.

4. Related Work

Zero-Order Optimization for LLMs Fine-Tuning Zero-order optimization bypasses backpropagation, significantly reducing memory costs, which is beneficial for fine-tuning LLMs (Malladi et al., 2024; Zhang et al., 2024). Methods like MeZO (Malladi et al., 2024) use SPSA to introduce noise and improve efficiency, while Zero sign-based SGD estimates gradients without full backpropagation, demonstrating the adaptability of zero-order methods in various contexts (Zhang et al., 2024).

Forward Gradient and Perturbation Learning Forward gradient methods estimate gradients without backpropagation, offering memory efficiency and biological plausibility (Ren et al., 2022; Baydin et al., 2022).

Recent research has extended these ideas by combining activity-perturbed forward gradient methods with forward-mode AD (Ren et al., 2022; Singhal et al., 2023; Fournier et al., 2023). For example, Ren et al. (Ren et al., 2022) employs multiple local greedy loss functions to stabilize the forward gradient, demonstrating effectiveness in image classification tasks. Similarly, Singhal et al. (Singhal et al., 2023) reduces gradient variance using the sparsity of ReLU activations in image classification contexts. Jiang et al. (Jiang et al., 2023) introduces the Unified Likelihood Ratio Method, which injects noise across different layers to facilitate more parallelizable backpropagation processes.

Activation perturbation methods for training neural networks were first explored by LeCun (1985), with more recent advancements studied in Ren et al. (2022); Singhal et al. (2023). While these methods have been successfully applied to image classification tasks (Ren et al., 2022; Singhal et al., 2023), their application to the training of large language models (LLMs) remains largely unexplored.

5. Experiments

We conducted experiments on several language models, including RoBERTa-Large, OPT (specifically, OPT-13B, 30B, and 66B) and GPT-2 following the experimental settings outlined in MeZO (Malladi et al., 2024) and ZO-LLM benchmarks (Liu et al., 2018). The downstream tasks involved fine-tuning with prompts. Detailed experimental settings are provided in the Appendix A.1. Our experiments include BERT-style models and autoregressive models, and evaluated various tuning strategies, including full-parameter tuning, LoRA, and prefix-tuning.

The results show that Zoop achieves superior performance with fewer training steps compared to MeZO, the most state-of-the-art zero-order tuning methods. Additionally, Zoop's memory usage is comparable to MeZO, with less computational overhead compared to PEFT methods such as LoRA or prefix-tuning.

5.1. BERT-style Language Model Finetuning

We performed experiments on RoBERTa-Large, following protocols used in previous works (Malladi et al., 2024; Gao et al., 2020; Malladi et al., 2023). Our experiments focused on tasks such as sentiment classification, natural language inference, and topic classification. We used a sampling strategy where $k = 512$ examples per class were selected.

Task	SST-2	SST-5	SNLI	MNLI	RTE	TREC
Type	—— sentiment ——		—— natural language inference ——			— topic —
Zero-shot	79.0	35.5	50.2	48.8	51.4	32.0
LP	91.3 (0.5)	51.7 (0.5)	80.9 (1.0)	71.5 (1.1)	73.1 (1.5)	89.4 (0.5)
Gradient estimation methods						
MeZO	90.7 (0.6)	43.5 (0.8)	65.2 (1.2)	54.8 (0.9)	62.6 (2.5)	47.4 (1.8)
MeZO (LoRA)	85.0 (0.5)	41.2 (0.7)	55.4 (2.3)	53.7 (2.0)	64.0 (1.4)	43.4 (2.7)
MeZO (Prefix)	86.3 (0.5)	41.4 (1.3)	60.9 (0.6)	53.5 (1.1)	56.5 (1.8)	49.3 (3.2)
MeZO-Adam	90.6 (0.4)	44.0 (1.7)	64.2 (1.4)	55.2 (0.7)	63.9 (1.2)	47.1 (2.8)
Zoop	91.4 (0.5)	**51.5 (2.3)**	**80.8 (1.7)**	70.2 (2.1)	72.8 (1.4)	**96.0 (0.5)**
Zoop (Prefix)	90.6 (1.1)	47.4 (2.1)	75.4 (0.8)	68.9 (0.7)	71.8 (1.3)	89.2 (1.5)
Zoop (LoRA)	89.9 (1.9)	49.7 (2.8)	71.0 (4.7)	65.0 (2.2)	66.6 (4.6)	90.6 (4.6)
Zoop-Adam	**91.5 (0.9)**	49.7 (1.1)	**80.8 (0.9)**	**70.4 (1.2)**	**74.6 (1.3)**	95.6 (0.7)
Exact gradient method						
FT	93.9 (0.7)	55.9 (0.9)	88.7 (0.8)	84.4 (0.8)	82.7 (1.4)	97.3 (0.2)
FT (LoRA)	94.2 (0.2)	55.3 (0.7)	88.3 (0.5)	83.9 (0.6)	83.2 (1.3)	97.0 (0.3)
FT (prefix)	93.7 (0.3)	54.6 (0.7)	88.3 (0.7)	83.3 (0.5)	82.5 (0.8)	97.4 (0.2)

Table 1: Table shows the average accuracy (with standard deviation) for RoBERTa-Large (350M parameters) across various tuning methods: MeZO, MeZO using LoRA and prefix-tuning, Zoop, Zoop with LoRA, and Zoop with prefix-tuning, alongside traditional fine-tuning with Adam (FT) and linear probing (LP). All methods used the same number of update steps (1,000). Zoop outperforms MeZO and closely approaches FT performance with significantly lower memory usage.

We compared the performance of MeZO, Zoop and fine-tuning with backpropagation over 1,000 steps. The results are summarized in Table 1.

Zoop outperforms MeZO in language model tuning with comparable memory usage. Our results indicate that Zoop, especially the versions employing Adam (Loshchilov & Hutter, 2017) optimizer, achieves competitive performance close to that of full model fine-tuning while requiring significantly less memory than fine-tuning with exact gradients. This demonstrates the potential of gradient estimation optimization methods like Zoop in effectively tuning large-scale language models with reduced resource requirements. Compared to previous state-of-the-art method MeZO, Zoop and its variants generally outperform MeZO across all tasks. The results underscore that while MeZO provides a solid baseline, Zoop presents a more robust perfromance within both limited computational resources and training time.

Zoop's lower variance in gradient estimation leads to faster convergence. As discussed earlier, Zoop's gradient estimation has lower variance, which contributes to its faster convergence rate compared to MeZO. In Figure 2, we illustrate this by comparing the training losses and validation accuracies of both methods. Zoop converges significantly faster than MeZO, achieving a stable training loss and accuracy in fewer update steps. Further, Table 2 provides a comprehensive comparison of these methods over extended training periods of 10K update steps. Zoop maintains competitive performance at 10K steps, closely approaching the results of full model fine-tuning.

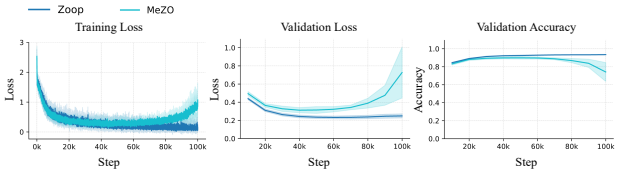

Figure 2: Comparison of training losses, validation loss, and validation accuracy for Zoop and MeZO over 100K update steps on TREC task. The figure illustrates the faster convergence rate of Zoop relative to MeZO, with Zoop reaching stable training loss and higher validation accuracy more quickly. Each curve represents the mean performance across multiple runs, with shaded areas indicating the standard deviation, demonstrating Zoop's consistent performance advantage in both training and validation phases.

5.2. Autoregressive Language Model Finetuning

We present the experimental results on the OPT-13B (Table 3), focusing on a range of natural language processing tasks. For each task, we sample 1000 examples for fine-tuning. In Table 3, we compares several fine-tuning approaches including Zero-shot, In-context Learning (ICL), Linear Probing (LP), various configurations of MeZO, Zoop, and full fine-tuning (FT). In Table 4, we compare the performance on OPT-30B and OPT-66B model with MeZO and Zoop in their variant with prefix fine-tuning. Notably, Zoop displayed better performance than that of MeZO.

Our results aligns with previous finding in masked language

Task Type	Update Steps	SST-2	SST-5	SNLI	MNLI	RTE	TREC
		—— sentiment ——		—— natural language inference ——			— topic —
Zero-shot	-	79.0	35.5	50.2	48.8	51.4	32.0
MeZO	10K	**92.5 (0.6)**	49.8 (1.9)	80.4 (0.8)	69.1 (1.5)	**75.1 (1.1)**	89.0 (0.6)
Zoop	10K	91.7 (0.6)	**51.7 (1.7)**	**81.2 (0.8)**	70.5 (1.7)	72.7 (1.2)	**96.8 (0.4)**

Table 2: Table shows average accuracy (with standard deviation) for RoBERTa-Large (350M parameters) using different tuning methods and update steps (**10K**). Results indicate that both Zoop and MeZO achieve similar levels of performance once a sufficient number of update steps is reached.

Task / Task type	SST-2	RTE	CB	BoolQ	WSC	WIC	MultiRC	COPA	ReCoRD
	—— classification ——							– multiple choice –	
Zero-shot	58.8	59.6	46.4	59.0	38.5	55.0	46.9	80.0	81.2
ICL	87.0	62.1	57.1	**66.9**	39.4	50.5	53.1	**87.0**	**82.5**
MeZO	85.8	61.0	**67.9**	64.4	60.6	53.8	55.5	84.0	81.3
MeZO (prefix)	75.7	56.3	62.5	63.6	62.5	53.6	52.4	80.0	80.6
MeZO (LoRA)	73.7	60.3	53.4	62.5	43.2	54.5	47.1	81.0	81.2
Zoop	**87.8**	**62.5**	**67.9**	64.8	59.6	**61.3**	**60.3**	80.0	81.7
Zoop (prefix)	83.9	59.2	64.3	64.2	**65.4**	58.3	58.2	78.0	80.6
Zoop (LoRA)	84.6	60.3	67.8	64.7	61.5	54.4	55.4	83.0	81.6
LP	93.4	68.6	67.9	59.3	63.5	60.2	63.5	55.0	27.1
FT (12x memory)	92.0	70.8	83.9	77.1	63.5	70.1	71.1	79.0	74.1

Table 3: Experiments on OPT-13B (with 1000 examples). ICL: in-context learning; LP: linear probing; FT: full fine-tuning with Adam.

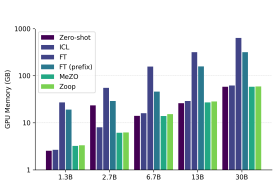

Figure 3: GPU memory consumption with different OPT models on SST-2.

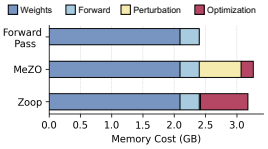

Figure 4: Memory consumption of fine-tuning a RoBERTa-large model on a single device.

Task	SST-2	RTE	BoolQ	WSC	WIC
30B zero-shot	56.7	52.0	39.1	38.5	50.2
30B MeZO (prefix)	76.9	59.6	49.6	47.1	55.6
30B Zoop (prefix)	**80.7**	**64.3**	**63.4**	**59.2**	**58.2**
66B zero-shot	57.5	**67.2**	**66.8**	43.3	50.6
66B MeZO (prefix)	79.4	60.3	50.5	49.1	55.1
66B Zoop (prefix)	**84.7**	62.3	61.8	**62.4**	**58.6**

Table 4: Experiments on OPT-30B and OPT-66B (with 1000 examples). We see that on most tasks Zoop effectively optimizes up to 66B models and outperforms zero-shot, MeZO and ICL.

models, show the effectiveness of Zoop in fine-tuning large language models. Compared to traditional methods such as full fine-tuning and MeZO, Zoop provides a balanced approach, offering substantial reductions in memory usage and faster convergence speed without compromising on the task performance. This makes it particularly suited for environments where computational resources are limited.

5.3. Memory Efficiency Analysis.

As detailed in Figure 3, we profiled the memory usage across different methods. The results show that Zoop achieved comparable memory efficiency with MeZO. Compared to full fine-tuning with exact gradient, Zoop requires approximately 12 times less memory. This efficiency highlights Zoop's capability to optimize large-scale models effectively under constrained resource settings.

In addition, we present the memory breakdown of fine-tuning a RoBERTa-large model on a single device in Figure 4. Compared to a pure forward pass, MeZO perturbs weight matrix, incurring an additional memory cost as large as the largest weight matrix (the word embedding matrix). In contrast, Zoop consumes additional space only as large as the activations, which is significantly smaller—by approximately 20x times—than that used by MeZO. Furthermore, Zoop consume additional memory usage due to local back-propagation, which remains minimal because of the small module size. These factors result in Zoop ultimately consume memory comparable to that of MeZO.

6. Limitation

One limitation of Zoop concerns the universality of its accelerated convergence. While Zoop is engineered to converge faster than methods like MeZO, leveraging activation perturbations for improved variance reduction especially with larger batch sizes or sequence lengths, this enhanced convergence speed does not uniformly translate to superior performance across all tasks within a fixed iteration budget (e.g., 100,000 update steps). In some instances, Zoop may offer only marginal improvements or not surpass MeZO, indicating that its convergence benefits and resulting efficacy can be task-dependent. This shows that while Zoop presents a promising avenue for faster zeroth-order optimization, further exploration may be needed to fully delineate the specific conditions or task characteristics under which its convergence advantages are more important.

References

Baydin, A. G., Pearlmutter, B. A., Syme, D., Wood, F., and Torr, P. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587*, 2022.

Fournier, L., Rivaud, S., Belilovsky, E., Eickenberg, M., and Oyallon, E. Can forward gradient match backpropagation? In *International Conference on Machine Learning*, pp. 10249–10264. PMLR, 2023.

Gao, T., Fisch, A., and Chen, D. Making pre-trained language models better few-shot learners. *arXiv preprint arXiv:2012.15723*, 2020.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Jiang, J., Zhang, Z., Xu, C., Yu, Z., and Peng, Y. One forward is enough for neural network training via likelihood ratio method. In *The Twelfth International Conference on Learning Representations*, 2023.

Le Cun, Y., Galland, C., and Hinton, G. E. Gemini: gradient estimation through matrix inversion after noise injection. *Advances in neural information processing systems*, 1, 1988.

LeCun, Y. A learning scheme for asymmetric threshold networks. *Proceedings of Cognitiva*, 85(537):599–604, 1985.

Li, X. L. and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Liu, S., Chen, P.-Y., Chen, X., and Hong, M. signsgd via zeroth-order oracle. In *International Conference on Learning Representations*, 2018.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Malladi, S., Wettig, A., Yu, D., Chen, D., and Arora, S. A kernel-based view of language model fine-tuning. In *International Conference on Machine Learning*, pp. 23610–23641. PMLR, 2023.

Malladi, S., Gao, T., Nichani, E., Damian, A., Lee, J. D., Chen, D., and Arora, S. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36, 2024.

Ren, M., Kornblith, S., Liao, R., and Hinton, G. Scaling forward gradient with local losses. *arXiv preprint arXiv:2210.03310*, 2022.

Singhal, U., Cheung, B., Chandra, K., Ragan-Kelley, J., Tenenbaum, J. B., Poggio, T. A., and Yu, S. X. How to guess a gradient. *arXiv preprint arXiv:2312.04709*, 2023.

Spall, J. C. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control*, 37(3):332–341, 1992.

Zhang, Y., Li, P., Hong, J., Li, J., Zhang, Y., Zheng, W., Chen, P.-Y., Lee, J. D., Yin, W., Hong, M., et al. Revisiting zeroth-order optimization for memory-efficient llm fine-tuning: A benchmark. *arXiv preprint arXiv:2402.11592*, 2024.

A. Appendix

A.1. Experiment details and hyperparameters setting.

We provide the hyperparameters used for the experiments on the RoBERTa and OPT models in Table 1.1 and Table 1.2 respectively.

Experiment	Hyperparameters	Zoop	MeZO	FT
Standard	Batch size	32	64	$\{2, 4, 8\}$
	Learning rate	$\{1e{-}3, 3e{-}3, 5e{-}3\}$	$\{1e{-}7, 1e{-}6, 1e{-}5\}$	$\{1e{-}4, 5e{-}4, 1e{-}3, 5e{-}3, 1e{-}2\}$
	ϵ	$1e{-}3$	$1e{-}3$	-
	Weight Decay	0	0	0
Prefix	Batch size	64	64	$\{8, 16, 32\}$
	Learning rate	$\{1e{-}1, 3e{-}1, 5e{-}1\}$	$\{1e{-}2, 5e{-}3, 1e{-}3\}$	$\{1e{-}2, 3e{-}2, 5e{-}2\}$
	ϵ	$1e{-}3$	$1e{-}1$	-
	Weight Decay	0	0	0
	# prefix tokens	5	5	5
LoRA	Batch size	64	64	$\{4, 8, 16\}$
	Learning rate	$\{5e{-}3, 8e{-}3, 1e{-}2\}$	$\{1e{-}5, 5e{-}5, 1e{-}4\}$	$\{1e{-}4, 3e{-}4, 5e{-}4\}$
	ϵ	$1e{-}3$	$1e{-}3$	-
	Weight Decay	0.1	0.1	0
	(r, α)	$(8, 16)$	$(8, 16)$	$(8, 16)$
Adam	Batch size	64	64	$\{8, 16, 32\}$
	Learning rate	$\{5e{-}7, 1e{-}6, 3e{-}6, 5e{-}6, 1e{-}5\}$	$\{1e{-}6, 5e{-}5, 1e{-}4, 5e{-}4, 1e{-}3\}$	$\{1e{-}5, 3e{-}5, 5e{-}5\}$
	ϵ	$1e{-}3$	$1e{-}3$	-
	Weight Decay	0	0	0

Table 1.1: The hyperparameter grids for RoBERTa-Large experiments.

Experiment	Hyperparameters	Zoop	MeZO	FT
Standard	Batch size	8	16	8
	Learning rate	$\{3e{-}3, 5e{-}3, 1e{-}2\}$	$\{1e{-}6, 5e{-}7, 1e{-}7\}$	$\{1e{-}5, 5e{-}5, 8e{-}5\}$
	ϵ	$1e{-}3$	$1e{-}3$	-
	Weight Decay	0	0	0
Prefix	Batch size	8	16	8
	Learning rate	$\{3e{-}1, 5e{-}1, 5e{-}2\}$	$\{5e{-}2, 1e{-}2, 5e{-}3\}$	$\{1e{-}5, 5e{-}5, 8e{-}5\}$
	ϵ	$1e{-}3$	$1e{-}1$	-
	Weight Decay	0	0	0
	# prefix tokens	5	5	5
LoRA	Batch size	8	16	8
	Learning rate	$\{3e{-}3, 5e{-}3, 1e{-}2\}$	$\{1e{-}4, 5e{-}5, 1e{-}5\}$	$\{1e{-}5, 5e{-}5, 8e{-}5\}$
	ϵ	$1e{-}3$	$1e{-}2$	-
	Weight Decay	0.1	0.1	0
	(r, α)	$(8, 16)$	$(8, 16)$	$(8, 16)$

Table 1.2: The hyperparameter grids for OPT experiments.

A.2. Statistical Properties and Limitations of the Weight Perturbation Gradient Estimator

Analysis of the SPSA estimator $\hat{g}$. To better understand the behavior of $\hat{g}$, consider its reformulation:

$$\hat{g}_n^{\text{SP}} = \frac{1}{n}\sum_{i=1}^{n}(\xi_i\xi_i^\top)g = \frac{1}{n}Wg, \qquad W \coloneqq \sum_{i=1}^{n}(\xi_i\xi_i^\top),$$

where scatter matrix W is known to follow the Wishart distribution $W \sim \mathcal{W}(n, I_{\mathtt{d}_{para}})$, with well-known analytical properties. Then we have the following proposition. In the following, for simplicity, we will use $\hat{g}$ to denote $\hat{g}_n^{\text{SP}}$.

Proposition A.1. *$\hat{g}$ is an unbiased estimator of $g \in \mathbb{R}^{\mathtt{d}_{para}}$, with*

$$\mathbb{E}[\hat{g}] = g, \qquad \mathbb{E}[\|\hat{g} - g\|^2] = \frac{\mathtt{d}_{para}+1}{n}\|g\|^2. \tag{5}$$

Moreover, the inner product $\hat{g}^\top g$ satisfies,

$$\mathbb{E}[\hat{g}^\top g] = \|g\|^2, \qquad \operatorname{var}(\hat{g}^\top g) = \frac{2}{n}\|g\|^4.$$

The cosine similarity between $\hat{g}_n$ and g satisfies

$$\mathbb{E}\left[\cos(\hat{g}, g)^2\right] \coloneqq \mathbb{E}\left[\left(\frac{\hat{g}^\top g}{\|g\|\,\|\hat{g}\|}\right)^2\right] \tag{6}$$

$$\geq \frac{n}{\mathtt{d}_{para}+n+1}. \tag{7}$$

Proof. 1) Note that $\hat{g}_n = \frac{1}{n}Wg$. Let $\hat{g}_n^{(i)}$ be the i-th element of $\hat{g}_n$. We can write $\hat{g}_n^{(i)} = \frac{1}{n}e_i^\top Wg$. We get

$$\operatorname{var}(\hat{g}_n^{(i)}) = \frac{1}{n}\left(\|e_i\|^2\|g\|^2 + (e_i^\top g)^2\right) = \frac{1}{n}\left(\|g\|^2 + (g^{(i)})^2\right).$$

Hence, the MSE is

$$\mathbb{E}[\|\hat{g}_n - g\|^2] = \sum_{i=1}^{\mathtt{d}_{para}}\operatorname{var}(\hat{g}_n^{(i)}) = \frac{1}{n}(d+1)\|g\|^2.$$

2) Write $\hat{g}_n^\top g = \frac{1}{n^2}g^\top Wg = \operatorname{vec}(g^\top Wg)$. We get

$$\operatorname{var}(\hat{g}_n^\top g) = \frac{1}{n^2}\operatorname{var}(g^\top Wg) = \frac{1}{n}(\|g\|^4 + \|g\|^4) = \frac{2}{n}\|g\|^4.$$

3) For any scalar $\alpha \in \mathbb{R}$, consider

$$\phi(\alpha, \hat{g}_n, g) = \frac{\|\alpha\hat{g}_n - g\|^2}{\|g\|^2}$$

Minimizing α yields $\alpha^* = \hat{g}_n^\top g / \|\hat{g}_n\|^2$, and

$$\inf_\alpha \phi(\alpha, \hat{g}_n, g) = 1 - \frac{(g_n^\top g)^2}{\|\hat{g}\|^2\|g\|^2} = 1 - \cos(\hat{g}_n, g)^2.$$

Therefore, for any $\alpha \in \mathbb{R}$, we have

$$\begin{aligned}
\mathbb{E}[1-\cos(\hat{g}_n, g)^2] &= \mathbb{E}[\inf_\alpha \phi(\alpha, \hat{g}_n, g)] \\
&\leq \inf_\alpha \mathbb{E}[\phi(\alpha, \hat{g}_n, g)] \\
&= \inf_\alpha \frac{1}{\|g\|^2}\left(\mathrm{var}(\alpha\hat{g}_n) + \mathbb{E}[(\alpha\hat{g}_n - g)]^2\right) \qquad \text{//Bias variance decomposition} \\
&= \inf_\alpha \frac{1}{\|g\|^2}\left(\alpha^2 \mathrm{var}(\hat{g}_n) + (1-\alpha)^2\|g\|^2\right) \\
&= \frac{\mathrm{var}(\hat{g}_n)}{\mathrm{var}(\hat{g}_n) + \|g\|^2} \\
&= \frac{\mathtt{d}_{para}+1}{\mathtt{d}_{para}+1+n},
\end{aligned}$$

where we used, for $x, y \geq 0$, $\inf_\alpha \alpha^2 x + (1-\alpha)^2 y = \frac{y^2x+x^2y}{(x+y)^2} = \frac{xy}{x+y}$. □

(5) illustrates that the mean squared error of $\hat{g}$ increases linearly with the dimension $\mathtt{d}_{para}$ of the parameter vector, and the cosine similarity decreases as $\mathtt{d}_{para}$ increases. Practically, to achieve estimation accuracy comparable to first-order methods, we would need $n \gtrsim \mathtt{d}_{para}$, which is often impractical. This underscores the limitations of using $\hat{g}$ for training large models.

A.3. Influence of Effective Dimension on Gradient Estimator Accuracy under Gaussian Noise

In this section, we explore the estimation accuracy of gradient estimators under Gaussian noise, emphasizing the influence of the effective dimension $\mathtt{d}_{eff}$ on their performance. We present theoretical results that quantify the mean squared error (MSE) and variance of mini-batch gradient estimators

Theorem A.2. *Assume ξ_i are iid Gaussian noise, and g is a mini-batch gradient estimation of a population gradient g^*, with $\mathbb{E}[g] = g_*$ and $\mathrm{var}(g) = \frac{\sigma_g^2}{n_{batch}}$. Assume $\mathtt{d}_{eff} = \sup_x \frac{\|J(x)\|_F^2\|h(x)\|^2}{\|g(x)\|^2}$. We have*

$$\begin{aligned}
&MSE(\hat{g}_{n,n_{batch}}, g) = \\
&\frac{1}{n_{batch}}\left(\sigma_g^2 + \frac{1}{n}(\mathtt{d}_{eff}+1)(\sigma_g^2 + \|g_*\|^2)\right)
\end{aligned}$$

Proof. Assume g is a mini-batch gradient estimation of a population gradient g^*, with $\mathbb{E}[g] = g_*$ and $\mathrm{var}(g) = \frac{\sigma_g^2}{n_{\mathrm{batch}}}$,

$$\begin{aligned}
&\mathbb{E}[\|\hat{g}_n(x) - g_*\|^2] = \mathrm{var}(\hat{g}_n(x)) \\
&= \frac{1}{n}\mathrm{var}(\hat{g}_n(x)) \\
&= \mathrm{var}(\mathbb{E}[\hat{g}_n(x) \mid x]) + \mathbb{E}[\mathrm{var}(\hat{g}_n(x) \mid x)] \\
&= \mathrm{var}(g(x)) + \mathbb{E}[\mathrm{var}(\hat{g}_n \mid g)] \\
&= \sigma_g^2 + \frac{1}{n}\mathbb{E}[(\mathtt{d}_{eff}(\theta)+1)\|g(x)\|^2] \\
&\leq \sigma_g^2 + \frac{1}{n}(\mathtt{d}_{eff}{}^* + 1)(\sigma_g^2 + \|g_*\|^2),
\end{aligned}$$

where we assume that $\mathtt{d}_{eff} = \sup_{alldata} \frac{\|J\|_F^2\|h\|^2}{\|J^\top h\|^2}$. Hence, consider the mini batch estimation,

$$\hat{g}_{n,n_{\mathrm{batch}}} = \frac{1}{n_{\mathrm{batch}}}\sum_{k=1}^{n_{\mathrm{batch}}} \hat{g}_n(x_k).$$

In this case, we have

$$MSE(\hat{g}_{n,n_{\texttt{batch}}}, g) = \frac{1}{n_{\texttt{batch}}}\left(\sigma_g^2 + \frac{1}{n}(\texttt{d}_{eff}{}^* + 1)(\sigma_g^2 + \|g_*\|^2)\right)$$

□

Lemma A.3. *Let $\tilde{g}(x) = J(x)^\top \xi\xi h(x)$ be the estimator of $g(x) = \nabla L(\theta, x)$ of each data point. We have*

$$\mathbb{E}[\tilde{g}(x)] = g(x),$$
$$\mathbb{E}\left[\|\tilde{g}(x) - g(x)\|^2\right] = (\texttt{d}_{eff}(x) + 1)\|g(x)\|^2,$$

where the $\texttt{d}_{eff}(x)$ is an effective dimension defined as

$$\texttt{d}_{eff}(x) = \frac{\|J(x)\|_F^2 \|h(x)\|^2}{\|J(x)^\top h(x)\|^2}.$$

In addition,

$$\mathbb{E}\left[\cos(\tilde{g}(x), g(x))^2\right] \coloneqq \mathbb{E}\left[\left(\frac{\tilde{g}(x)^\top g(x)}{\|g(x)\|\,\|\tilde{g}(x)\|}\right)^2\right] \tag{8}$$
$$\geq \frac{1}{\texttt{d}_{eff} + 2}. \tag{9}$$

Remark Assume each element of J, h, and $g = J^\top h$ is in the order of 1, we have $\|A\|_F^2 \asymp d \times d_h$, $\|h\|^2 \asymp d_h$ and $\|g\|^2 = \|A^\top h\|^2 \asymp d$, and hence $\texttt{d}_{eff} \asymp \texttt{d}_{hid}$.

A.4. Proof of MSE Reduction Using Activation Perturbation Estimator in Transformer Models

Proof. Write $J = [a_1, \ldots, a_{\texttt{d}_{para}}]$, we have $\tilde{g}_n^{(i)} = \frac{1}{n} a_i^\top W g$.

$$\text{var}(\tilde{g}_n^{(i)}) = \frac{1}{n}(\|a_i\|^2 \|h\|^2 + (a_i^\top h)^2), \quad \forall i \in [\texttt{d}_{para}].$$

Hence,

$$\begin{aligned}
\mathbb{E}[\|\tilde{g}_n - g\|^2] &= \sum_{i=1}^{\texttt{d}_{para}} \text{var}(\tilde{g}_n^{(i)}) \\
&= \sum_{i=1}^{\texttt{d}_{para}} \frac{1}{n}(\|a_i\|^2 \|h\|^2 + (a_i^\top h)^2) \\
&= \frac{1}{n}(\|J\|_F^2 \|h\|^2 + \|J^\top h\|^2) \\
&= \frac{1}{n}(\texttt{d}_{eff} + 1)\|g\|^2 \qquad //g = J^\top h.
\end{aligned}$$

The result on the cosine similarity follows the same procedure as the proof of Theorem **??**. □

A.5. Activation Perturbation for Linear Layers

Let us start with considering a linear layer, where the loss function can be written into a form of

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} \ell(Wx_i),$$

where W is the weight matrix, and $\{x_i\}_{i=1}^N$ is the dataset. The gradient of $L(W)$ is

$$\nabla L(W) = \frac{1}{N}\sum_{i=1}^{N} h_i x_i^\top, \quad h_i = \nabla \ell(W x_i), \tag{10}$$

where each data point contributes a *rank-one* component $h_i x_i^\top$ to the gradient. Since the input x_i is given or calculated during the forward pass, the main difficult reduces to calculating $h_i = \nabla \ell(W x_i)$, which is the signal passed from the output in the backward pass.

To leverage the structure of $\nabla L(W)$ in Eq. (10), we can use a random perturbation estimator to estimate h_i for each data point, rather than directly estimating the overall gradient. In particular, let us use a single random vector ξ_i to estimate each h_i for data point x_i in (10), yielding

$$\hat{g} = \frac{1}{n_{\texttt{batch}}}\sum_{i=1}^{n_{\texttt{batch}}} \hat{h}_i x_i^\top,$$
$$\text{with} \quad \hat{h}_i = \alpha_i \xi_i, \quad \text{and} \quad \alpha_i = h_i^\top \xi_i,$$

where α_i can be calculated with either forward mode AD or finite difference:

$$\begin{aligned}\alpha_i &= \nabla_\epsilon L(W x_i + \xi_i)\,\big|_{\epsilon=0} \\ &= \frac{L(W x_i + \epsilon \xi_i) - L(W x_i - \epsilon \xi_i)}{2\epsilon} + O\big(\epsilon^2\big).\end{aligned}$$

Compared with the standard method, the random perturbation is introduced on the intermediate state $z = W x_i \in \mathbb{R}^{\texttt{d}_{\texttt{out}}}$, rather than the parameter space (which is $W \in \mathbb{R}^{\texttt{d}_{\texttt{out}} \times \texttt{d}_{\texttt{in}}}$ in this case). This yields two key advantages that remediates the challenges of weight perturbation. The general intuition is that activation perturbation allows us to *1) introducing lower dimensional random perturbation, and 2) a larger number of independent perturbations. These two aspects together improves the performance.*