Zoop it! Efficient Zero-Order Optimization with Output Perturbation

Anonymous ACL submission

Abstract

As language models continue to scale in size, the computational burden of first-order gradient methods (e.g., stochastic gradient descent variants) for pretraining or fine-tuning becomes a significant challenge. Zero-order optimization methods, which bypass backpropagation, reduce the need to store per-layer activations, thereby offering a more computationally feasible alternative. Recent advancements, such as the memory-efficient MeZO method, leverage simultaneous perturbation stochastic approximation (SPSA) to fine-tune large models via finite-difference estimation with two forward passes. However, SPSA suffers from high variance, as all neurons receive the same reward signal. In this work, we introduce Zeroth-order Optimization with Output Perturbation (Zoop), which perturbs per-layer activations instead of weights. We provide a memory-efficient, model-agnostic implementation of Zoop by introducing a lightweight noise layer between consecutive layers. Zoop not only improves memory efficiency compared to MeZO when batch sizes are small but also mitigates variance in large-batch or long-sequence contexts. The modular noise layer design integrates seamlessly into pretrained models, enabling broad applicability. We evaluate Zoop on autoregressive and masked language models up to 66 billion parameters across tasks such as classification, multiple-choice, and text generation. Results demonstrate that Zoop fine-tunes large language models more efficiently than current methods, achieving effective optimization with fewer updates.

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, 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 and 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

[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.

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 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 finetunes models more efficiently than prior methods. Our contributions are as follows:

- We introduce Zoop, a low-variance finetuning 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 pretrained model.
- We demonstrate Zoop's effectiveness across various model types, tasks, and scales, and demonstrated Zoop can achieve 10x faster finetuning of LLM compared to existing approaches.

2 Background

Zeroth-order methods estimate the gradient $\nabla L(\theta)$ of a loss function $L(\theta)$ without backpropagation. For example, Simultaneous Perturbation Stochastic Approximation (SPSA) estimates the gradient using only two function evaluations:

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

where $\{\xi_i\}$ are independent standard Gaussian vectors, and n balances computational cost and estimation accuracy.

Alternatively, forward-mode autodifferentiation approximates the inner product $\nabla L(\theta)^\top \xi_i$:

$$\hat{g}^{\text{SP}}_n(\theta) \approx \frac{1}{n}\sum_{i=1}^{n} \alpha_i \xi_i, \quad \text{with} \quad \alpha_i = \nabla L(\theta)^\top \xi_i.$$

This estimator projects $\nabla L(\theta)$ onto the space spanned by $\{\xi_i\}$:

$$\hat{g}^{\text{SP}}_n(\theta) = W_n \nabla L(\theta), \quad \text{where} \quad W_n = \frac{1}{n}\sum_{i=1}^{n} \xi_i \xi_i^\top,$$

with $\xi_i \sim \mathcal{N}(0, I_d)$, nW_n follows a Wishart distribution, allowing us to characterize $\hat{g}^{\text{SP}}_n(\theta)$:

Theorem 2.1. *Let $g = \nabla L(\theta)$. Write $g = \nabla L(\theta)$ and $\hat{g}_n = \hat{g}^{\text{SP}}_n(\theta)$ to simplify notation. Then:*

$$\mathbb{E}[\hat{g}_n] = g, \quad \mathbb{E}[\|\hat{g}_n - g\|^2] = \frac{d+1}{n}\|g\|^2.$$

The inner product satisfies:

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

The expected cosine similarity between $\hat{g}_n$ and g is:

$$\mathbb{E}\left[\cos^2(\hat{g}_n, g)\right] \geq \frac{n}{d+n+1}. \tag{2}$$

The proof uses the variance formula of the Wishart random variable nW_n. See Appendix A.3.

Despite reducing function evaluations, this estimator faces a high-variance problem: the mean squared error is $O(d/n)$, implying a curse of dimensionality. In practice, introducing independent perturbations for each data point in a batch is infeasible due to parallelization challenges.

3 Zero-Order Gradient Estimation with Activation Perturbation

By leveraging the structural properties of deep neural networks—specifically by perturbing activations instead of weights—we can significantly reduce the variance of gradient estimations and overcome the curse of dimensionality inherent in weight perturbation methods. This activation perturbation approach benefits from the lower dimensionality of activations and allows for introducing independent random noises for each data point, leading to more precise gradient estimates that scale favorably with batch size.

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.

This work investigates activation perturbation for LLM training and demonstrates that it offers a promising alternative to existing weight perturbation methods. However, a key practical challenge arises from the difficulty of implementing activation perturbations in the PyTorch framework, as

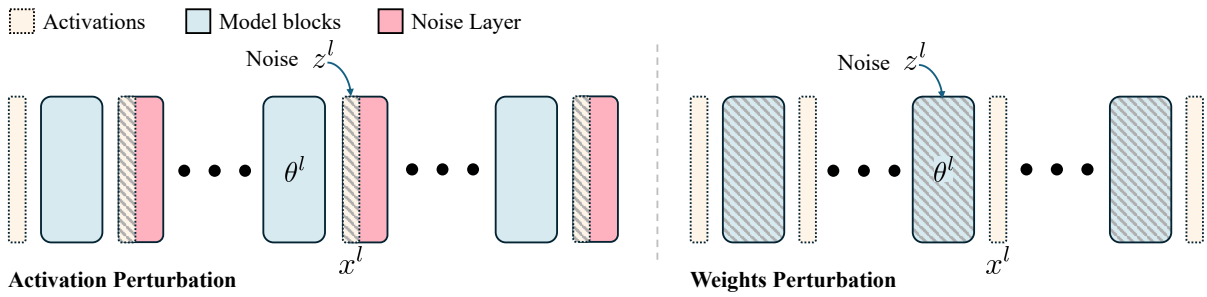

Figure 1: **Comparison of Activation Perturbation and Weight Perturbation Methods.** Left: We introduce the Noise Layers applies noise z^l directly to the activations x^l. The advantage of this method is that it operates in a smaller dimensional space (activations) compared to weight space, potentially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

it requires precise specification of where the perturbations should be applied. To address this, we propose a framework that introduces *Noise Layers*, facilitating the straightforward implementation of activation perturbations across various architectures.

Activation Perturbation for DNN. Given the input x, consider a specific layer in a deep neural network (DNN) with parameter θ and activation function ϕ, where the loss is defined as:

$$L(\theta, x) = \ell(\phi(\theta, x), x).$$

Here, ℓ is a loss function (e.g., the mean square error or cross-entropy loss). Then we can compute the gradient with respect to θ via the chain rule (denote $z = \phi(\theta, x) \in \mathbb{R}^{\mathtt{d}_{hid}}$): $\nabla_\theta L(\theta, x) = \frac{\mathrm{d}\phi(\theta,x)}{\mathrm{d}\theta}\frac{\mathrm{d}\ell(z,x)}{\mathrm{d}z}$. Let $\{x_i\}_{i=1}^{n_{\mathsf{batch}}}$ be a batch of data, then the mini-batch gradient is

$$\frac{1}{n_{\mathsf{batch}}}\sum_{i=1}^{n_{\mathsf{batch}}} \nabla L(\theta, x_i) = \frac{1}{n_{\mathsf{batch}}}\sum_{i=1}^{n_{\mathsf{batch}}} J(x_i)^\top h(x_i),$$

with $J(x)^\top = \dfrac{\mathrm{d}\phi(\theta, x)}{\mathrm{d}\theta}$, $h(x) = \dfrac{\mathrm{d}\ell(z, x)}{\mathrm{d}z}$.

Estimating each $h(x_i)$ with an independent perturbation ξ_i yields

$$\tilde{g}_n = \frac{1}{n_{\mathsf{batch}}}\sum_{i=1}^{n_{\mathsf{batch}}} J(x_i)^\top \hat{h}(x_i), \tag{3}$$

$$\text{with} \quad \hat{h}(x_i) = (h(x_i)^\top \xi_i)\xi_i \tag{4}$$

In Appendix A.4, we analyze the mean squared error and variance of $\hat{g}_n$ under Gaussian noise ξ, where we show the mean square error scales with $\mathcal{O}(\mathtt{d}_{hid})$. Because the hidden size $\mathtt{d}_{hid}$ is typically much smaller than the parameter size $\mathtt{d}_{para}$, the activation perturbation estimator $\tilde{g}_n$ yields a significant improvement on accuracy. Taking a transformer block of model dimension $\mathtt{d}_{model}$ as an example, we have $\mathtt{d}_{para} = \mathtt{d}_{model}^2$, and $\mathtt{d}_{hid} = \mathtt{d}_{model}$. Hence, we decrease the MSE from $O(\mathtt{d}_{model}^2)$ to $O(\mathtt{d}_{model})$. See the Appendix for the proof of Lemma A.3.

Activation Perturbation for LLM. Consider a LLM structured into L modular layers. Within this hierarchical framework, the l-th module processes its input $x^l \in \mathbb{R}^{d_l}$, resulting in the output:

$$x^{l+1} = \phi^l(x^l; \theta^l),$$

where $\phi\colon \mathbb{R}^{\mathtt{d}_{para}} \times \mathcal{X} \to \mathbb{R}^{\mathtt{d}_{hid}}$ maps the input x^l to its output x^{l+1}. To train the neural network, we minimize the following loss function:

$$\min_\theta \frac{1}{n_{\mathsf{batch}}}\sum_{i=1}^{n_{\mathsf{batch}}} \mathcal{L}(x_i^L),$$

where n_{batch} is the size of the batch and x_i^L is the output of the last layer L. For a given intermediate layer l, consider the function $\frac{1}{n_{\mathsf{batch}}}\sum_{i=1}^{n_{\mathsf{batch}}} \mathcal{L}(\phi(\theta^l, x_i^l))$, and $\mathcal{L}\colon \mathbb{R}^{\mathtt{d}_{hid}} \times \mathcal{X} \to \mathbb{R}$ evaluates the loss. The gradient for θ^l at any intermediate layer is given by:

$$\frac{1}{n_{\mathsf{batch}}}\sum_{i=1}^{n_{\mathsf{batch}}} h_i \nabla_{\theta^l}\phi(\theta^l, x_i^l)^\top, \tag{5}$$

$$h_i = \nabla_{x_i^{l+1}}\mathcal{L}(x_i^{l+1}), \tag{6}$$

Each data point contributes a rank-one component $h_i \nabla_{\theta^l}\phi(\theta^l, x_i^l)^\top$ to the gradient, where the gradient $\nabla_{\theta^l}\phi(\theta^l, x_i^l)$ can be computed using forward-mode AD or backpropagation. The main challenge is

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
  $\mathbf{y}_+ \leftarrow$ `PerturbActivations(`$\mathbf{x}$, ϵ, s, $\boldsymbol{\theta}$, Φ`)`
  $\mathbf{y}_- \leftarrow$ `PerturbActivations(`$\mathbf{x}$, $-\epsilon$, s, $\boldsymbol{\theta}$, Φ`)`
  $\boldsymbol{\alpha} \leftarrow \big(\mathcal{L}(\mathbf{y}_+, \mathbf{y}) - \mathcal{L}(\mathbf{y}_-, \mathbf{y})\big) \,/\, (2\epsilon)$ ▷ `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`
    $\boldsymbol{\theta}^l \leftarrow \boldsymbol{\theta}^l - \eta_t \times \mathbf{g}^l$
  end
end
Subroutine `PerturbActivations(`$\mathbf{x}^0$, ϵ, s, $\boldsymbol{\theta}$, Φ`)`
  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 $\mathbf{z}^l \sim \mathcal{N}(0, I)$
  end
  return $\mathbf{x}^L$

efficiently calculating h_i, the gradient signal passed backward to the output of layer l.

To efficiently leverage the structure of Equation (5), we employ a random perturbation estimator to compute h_i for each data point individually. Specifically, we use a unique random vector ξ_i for each estimation of h_i:

$$\hat{g} = \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} \hat{h}_i \nabla_{\theta^l} \phi(\theta^l, x_i^l)^\top,$$
$$\text{with} \quad \hat{h}_i = \alpha_i \xi_i, \quad \text{and} \quad \alpha_i = h_i^\top \xi_i.$$

To approximate α_i, we apply a finite difference method:

$$\begin{aligned}\alpha_i &= \nabla_\epsilon \mathcal{L}(x_i^{l+1} + \epsilon \xi_i) \big|_{\epsilon=0} \\ &= \frac{1}{2\epsilon}(\mathcal{L}(x_i^{l+1} + \epsilon \xi_i) - \mathcal{L}(x_i^{l+1} - \epsilon \xi_i)) + O(\epsilon^2).\end{aligned}$$

Efficient Implementation. We integrate activation perturbation with a novel implementation that minimizes memory overhead and maintains computational efficiency. In Algorithm 1, we present an efficient activation perturbation. The Implementation of Noise Layer in PyTorch-like pseudocode is provided in Algorithm 2. Our implementation of Zoop leverages localized backward propagation across modular layers of the network for automatic computation of $\nabla_{\theta^l} \phi(\theta^l, x_i^l)$. In practice, because the localized backpropagation is computed across small network modules, the computational overhead is very small (see Figure 5 for the memory breakdown). This is particularly beneficial for LLMs, where the size and complexity of models can lead to significant computational and memory challenges.

Additionally, similar to (Malladi et al., 2024), the random seeds are sampled and reused to generate noise vectors for activation perturbation. Note that in practice, we save much more memory by perturbing the activations instead of an entire weight matrix as shown in Figure 5. The latter incurs an additional memory cost as large as the largest weight matrix. Through this efficient implementation, our method maintains the balance between computational demand and memory usage, aligning the memory usage with the most advanced zeroth-order methods, such as MeZO.

4 Related Work

Parameter-Efficient Fine-Tuning for LLMs Recent work on fine-tuning large language models focuses on improving parameter efficiency to reduce computational and memory demands (Zaken et al., 2021; Lester et al., 2021). Approaches like LoRA introduce low-rank matrices to adjust pre-trained weights, minimizing memory usage while maintaining performance (Hu et al., 2021). Similarly, Adapter layers update only a small portion of parameters while achieving near full-model performance (Houlsby et al., 2019), and Prefix-tuning modifies the attention mechanism with a few trainable vectors (Li and Liang, 2021). These methods highlight the trend toward resource-efficient strategies that enhance the feasibility of using LLMs in constrained environments.

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. These approaches, while innovative, often require the manual design of local losses and are generally memory-intensive due to the use of forward-mode AD. In contrast, our work, Zoop, differs from these methods and eliminates the need for forward-mode AD and manual design of local losses. Instead, we use localized backpropagation to achieve efficient and fast training in LLM fine-tuning.

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. 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, 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 re-

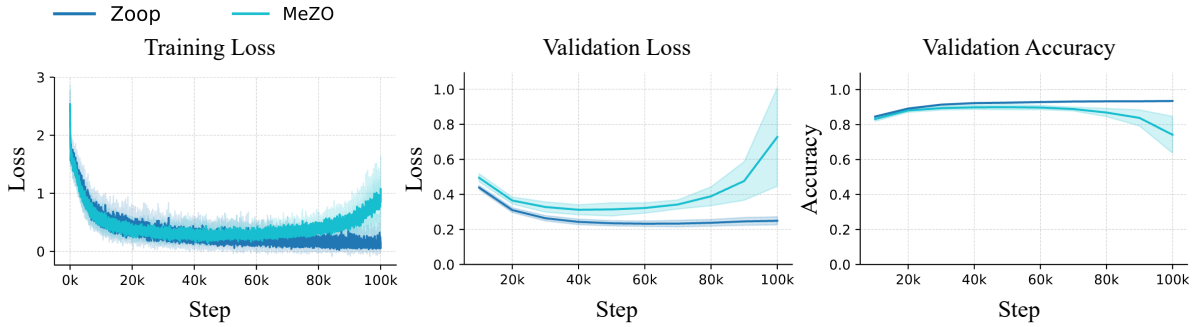

Figure 2: Comparison of training losses, validation loss, and validation accuracy for Zoop and MeZO over 100K update steps on TREC task. The figure illustrates the faster convergence rate of Zoop relative to MeZO, with Zoop reaching stable training loss and higher validation accuracy more quickly. Each 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.

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.

sults 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.

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	SST-2	RTE	CB	BoolQ	WSC	WIC	MultiRC	COPA	ReCoRD
Task type	———	———	———	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.

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 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.

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.

5.3 Memory Efficiency Analysis.

As detailed in Figure 4, 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 sin-

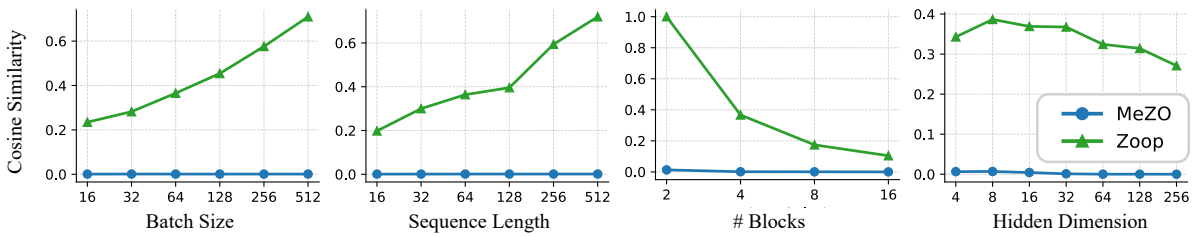

Figure 3: **Ablation study.** Cosine similarity between estimated and ground-truth gradients for Zoop and MeZO across varying batch sizes, sequence lengths, number of blocks, and hidden dimensions. Zoop generally outperforms MeZO, with improvements in gradient accuracy as batch size and sequence length increase, but higher noise or larger embedding dimensions slightly degrade performance.

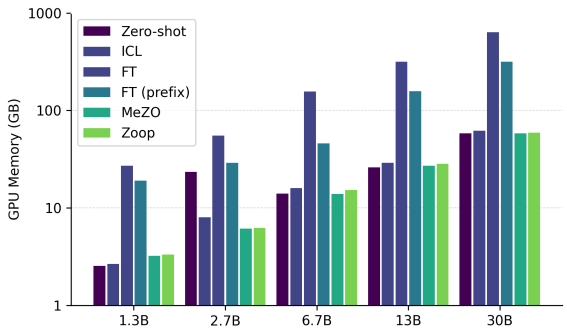

Figure 4: GPU memory consumption with different OPT models on SST-2.

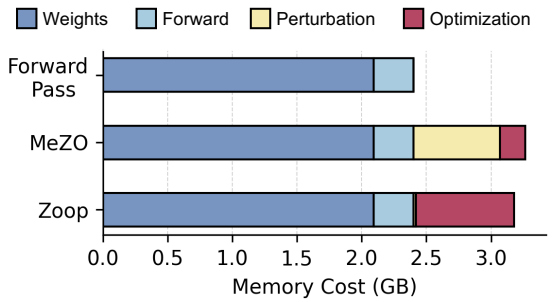

Figure 5: Memory consumption of fine-tuning a RoBERTa-large model on a single device.

gle device in Figure 5. 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 backpropagation, which remains minimal because of the small module size. These factors result in Zoop ultimately consume memory comparable to that of MeZO.

5.4 Ablation Study

To better understand how batch size, sequence length, number of blocks (# blocks), and the hidden dimension influences MeZO and Zoop's performance, we conduct a small-scale ablation studies by comparing MeZO and Zoop's estimated gradient to the ground-truth gradient, computed by a GPT-2 model. In particular, we vary the batch size from $\{16, 32, 64, 128, 256, 512\}$, sequence length from $\{16, 32, 64, 128, 256, 512\}$, # blocks from $\{2, 4, 8, 16\}$, and the hidden dimension from $\{4, 8, 16, 32, 64, 128, 256\}$. For each experiment, we use 1000 trials of gradient estimation and report the average cosine similarity for each method. The result is provided in Figure 3.

Our findings show that: 1) Zoop generally outperforms MeZO, aligning more closely with true gradients. 2) As batch size and sequence length increase, the accuracy of Zoop's gradient estimates improves. However, increasing the number of noise layers or embedding dimensions introduces more noise, which slightly reduces the similarity of estimated gradients to true gradients.

6 Conclusion

In this paper, we introduce a large language model fine-tuning method based on gradient estimation through activation perturbation. Our approach, Zoop, is designed to be memory-efficient, providing an alternative solution to zeroth-order methods that perturb weights. Zoop not only achieves faster convergence with estimated gradients that exhibit lower variance but also maintains minimal memory consumption. This efficiency and effectiveness open avenues for future research in developing scalable optimization techniques that enhance the practicality and sustainability of deploying advanced language models in resource-limited settings.

7 Limitation

Despite its simplicity and effectiveness, Zoop does have limitations, such as potentially increased memory usage for finetuning tasks with exceptionally large inputs. This is because Zoop introduces noise into the activations, which can consume more memory; in such cases, perturbing weights could be a more memory-efficient alternative.

References

Atılım Güneş Baydin, Barak A Pearlmutter, Don Syme, Frank Wood, and Philip Torr. 2022. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587*.

Louis Fournier, Stéphane Rivaud, Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. 2023. Can forward gradient match backpropagation? In *International Conference on Machine Learning*, pages 10249–10264. PMLR.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2020. Making pre-trained language models better few-shot learners. *arXiv preprint arXiv:2012.15723*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Jinyang Jiang, Zeliang Zhang, Chenliang Xu, Zhaofei Yu, and Yijie Peng. 2023. One forward is enough for neural network training via likelihood ratio method. In *The Twelfth International Conference on Learning Representations*.

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

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

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Sijia Liu, Pin-Yu Chen, Xiangyi Chen, and Mingyi Hong. 2018. signsgd via zeroth-order oracle. In *International Conference on Learning Representations*.

I Loshchilov. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. 2024. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36.

Sadhika Malladi, Alexander Wettig, Dingli Yu, Danqi Chen, and Sanjeev Arora. 2023. A kernel-based view of language model fine-tuning. In *International Conference on Machine Learning*, pages 23610–23641. PMLR.

Mengye Ren, Simon Kornblith, Renjie Liao, and Geoffrey Hinton. 2022. Scaling forward gradient with local losses. *arXiv preprint arXiv:2210.03310*.

Utkarsh Singhal, Brian Cheung, Kartik Chandra, Jonathan Ragan-Kelley, Joshua B Tenenbaum, Tomaso A Poggio, and Stella X Yu. 2023. How to guess a gradient. *arXiv preprint arXiv:2312.04709*.

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

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Yihua Zhang, Pingzhi Li, Junyuan Hong, Jiaxiang Li, Yimeng Zhang, Wenqing Zheng, Pin-Yu Chen, Jason D Lee, Wotao Yin, Mingyi Hong, et al. 2024. Revisiting zeroth-order optimization for memory-efficient llm fine-tuning: A benchmark. *arXiv preprint arXiv:2402.11592*.

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 5 and Table 6 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 5: 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 6: The hyperparameter grids for OPT experiments.

A.2 PyTorch Implementation of Noise Layer

We provide the PyTorch-like pseudocode for the noise layer implementation in Algorithm 2.

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
```

A.3 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:

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

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 $\hat{g}_n^{\mathrm{SP}}$.

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

$$\mathbb{E}[\hat{g}] = g, \qquad \mathbb{E}[\|\hat{g} - g\|^2] = \frac{\mathtt{d}_{para} + 1}{n}\|g\|^2. \tag{7}$$

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

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

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

$$\mathbb{E}\left[\cos(\hat{g}, g)^2\right] := \mathbb{E}\left[\left(\frac{\hat{g}^\top g}{\|g\|\,\|\hat{g}\|}\right)^2\right] \tag{8}$$

$$\geq \frac{n}{\mathtt{d}_{para} + n + 1}. \tag{9}$$

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

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

Hence, the MSE is

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

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

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

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

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

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

(7) 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.4 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 $\mathrm{var}(g) = \frac{\sigma_g^2}{n_{\mathtt{batch}}}$. Assume $\mathtt{d}_{eff} = \sup_x \frac{\|J(x)\|_F^2 \|h(x)\|^2}{\|g(x)\|^2}$. We have*

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

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

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

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

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

In this case, we have

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

□

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*

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

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

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

In addition,

$$\mathbb{E}\left[\cos(\tilde{g}(x), g(x))^2\right] := \mathbb{E}\left[\left(\frac{\tilde{g}(x)^\top g(x)}{\|g(x)\|\,\|\tilde{g}(x)\|}\right)^2\right] \tag{10}$$

$$\geq \frac{1}{\mathtt{d}_{eff} + 2}. \tag{11}$$

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 $\|g\|^2 = \left\|A^\top h\right\|^2 \asymp d$, and hence $\mathtt{d}_{eff} \asymp \mathtt{d}_{hid}$.

A.5 Proof of MSE Reduction Using Activation Perturbation Estimator in Transformer Models

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

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

Hence,

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

The result on the cosine similarity follows the same procedure as the proof of Theorem 2.1. □

A.6 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

$$
\nabla L(W) = \frac{1}{N} \sum_{i=1}^{N} h_i x_i^\top, \quad h_i = \nabla \ell(Wx_i), \tag{12}
$$

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 $h_i = \nabla \ell(Wx_i)$, which is the signal passed from the output in the backward pass.

To leverage the structure of $\nabla L(W)$ in Eq. (12), 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 (12), yielding

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

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

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

Compared with the standard method, the random perturbation is introduced on the intermediate state $z = Wx_i \in \mathbb{R}^{\mathtt{d}_{\mathtt{out}}}$, rather than the parameter space (which is $W \in \mathbb{R}^{\mathtt{d}_{\mathtt{out}} \times \mathtt{d}_{\mathtt{in}}}$ 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.*