Towards Efficient Low-order Hybrid Optimizer for Language Model Fine-tuning

Minping Chen[1], **You-Liang Huang**[1], **Zeyi Wen**[*1,2]

[1] The Hong Kong University of Science and Technology (Guangzhou)
[2] The Hong Kong University of Science and Technology
{mchen779, yhuang142}@connect.hkust-gz.edu.cn, wenzeyi@ust.hk

Abstract

As the size of language models notably grows, fine-tuning the models becomes more challenging: fine-tuning with first-order optimizers (e.g., SGD and Adam) requires high memory consumption, while fine-tuning with a memory-efficient zeroth-order optimizer (MeZO) has a significant accuracy drop and slower convergence rate. In this work, we propose a Low-order Hybrid Optimizer (LoHO) which merges zeroth-order (ZO) and first-order (FO) optimizers for fine-tuning. LoHO is empowered with inter-layer hybrid optimization and intra-layer hybrid optimization, which boosts the accuracy of MeZO while keeping memory usage within a budget. The inter-layer hybrid optimization exploits the FO optimizer in deep layers and the ZO optimizer in shallow ones, therefore avoiding unnecessary gradient propagation to improve memory efficiency. The intra-layer hybrid optimization updates a proportion of parameters in a layer by the ZO optimizer, and the rest by the FO optimizer, taking advantage of gradient sparsity for high efficiency implementation. Our experimental results across common datasets on different pre-trained backbones (i.e., RoBERTa-large, OPT-13B and OPT-30B) demonstrate that LoHO can significantly improve the predictive accuracy and convergence rate of MeZO, while controlling the memory footprint during fine-tuning. Moreover, LoHO can achieve comparable performance with first-order fine-tuning using substantially fewer memory resources.

Introduction

Fine-tuning with first-order (FO) optimizers, such as Adam (Kingma and Ba 2014) and AdamW (Loshchilov and Hutter 2018), has been the standard paradigm to adapt pre-trained language models (PLMs) to the specific downstream tasks (Liu et al. 2019; Raffel et al. 2020; Zhang et al. 2023). However, with the growth of the number of model parameters, the increasing memory requirement for fine-tuning with these first-order optimizers becomes a bottleneck, as the back-propagation process is memory-consuming due to the storage of gradients, optimizer states, and activations. For example, Malladi et al. (2024) demonstrate that full fine-tuning an OPT-13B model with Adam requires approximately 12 times the memory cost as the inference.

In response to this challenge, several approaches have been proposed. In-context learning (ICL) (Brown et al. 2020) makes an inference with only forward passes based on the input prompts (i.e., labeled samples). However, the context sizes of most PLMs are limited and the predictive accuracy of ICL is often much worse than fine-tuning (Brown et al. 2020). Another type of method, namely parameter-efficient fine-tuning (PEFT) (Houlsby et al. 2019; Li and Liang 2021; Hu et al. 2022), only updates a small number of model parameters while approaching similar predictive accuracy of full fine-tuning methods, but it still needs to store numerous activations (intermediate results of the forward pass) as the trainable parameters are dispersed across the entire model. To address these problems, MeZO (Malladi et al. 2024) proposes a memory-efficient zeroth-order (ZO) optimizer to estimate the gradients based on finite differences of function values with just two forward passes, thus saving a significant amount of memory. Despite its benefits in memory efficiency, MeZO suffers from a significant accuracy degradation compared with full fine-tuning using first-order optimizers on various tasks and needs substantially more training steps to converge (Malladi et al. 2024).

We observe that MeZO only requires the same memory as inference, and the free/unused memory is wasted when the GPUs are solely occupied to maximize the computation efficiency during fine-tuning. For example, we find that when using an A800 GPU to fine-tune OPT-13B with MeZO, it exhibits over 10GB of free memory. It would be beneficial to use this free memory to mitigate MeZO's accuracy drop and convergence issues. To this end, we propose Low-order Hybrid Optimizer (LoHO), which exploits the memory efficiency of ZO optimizers and the optimization quality of FO.[1] LoHO is equipped with two hybrid optimization strategies, i.e., inter-layer hybrid optimization and intra-layer hybrid optimization. The inter-layer hybrid optimization employs an FO optimizer for fine-tuning a PLM's several layers, while a ZO optimizer updates its rest layers, avoiding unnecessary gradient propagation with the suitable setting of the FO layers. Different from inter-layer hybrid optimization, intra-layer hybrid optimization selects some parame-

*Corresponding Author

[1] In this paper, we classify zeroth-order optimizers and first-order optimizers as low-order optimizers to distinguish them from high-order optimizers, i.e., second-order optimizers.

ters in a layer to be updated by the FO optimizer, while the rest are updated by the ZO optimizer. Furthermore, LoHO makes use of the gradient sparsity due to the mixture of FO and ZO to achieve more efficient fine-tuning, by storing the trainable parameters and their gradients in a sparse format. To enhance the convergence in fine-tuning, we customize the learning rates of both optimizers in LoHO, based on the Frobenius norm of gradients estimated by the ZO optimizer and that computed by the FO optimizer, thus keeping the progress of the two optimizers in a similar pace. A nice property of LoHO is that it enables practitioners to control the GPU memory usage (to the fullest extent if needed) by setting the number of layers or ratio of parameters to be handled by the FO optimizer and ZO optimizer.

To summarize, our key contributions are as follows.

- We propose a low-order hybrid optimizer (called LoHO) which integrates a zeroth-order optimizer with a first-order optimizer for fine-tuning language models. LoHO is equipped with inter-layer hybrid optimization and intra-layer hybrid optimization, which maximize the memory usage within a given budget while improving the model quality of the MeZO method.
- To further optimize LoHO, we design a customization strategy of the learning rate for FO and ZO optimizers, based on the Frobenius norm of gradients, thus achieving better generalization.
- Our experiments across common datasets on three pre-trained backbones (i.e., RoBERTa-large, OPT-13B, and OPT-30B) show the effectiveness of LoHO on predictive accuracy and convergence rate compared with MeZO. Furthermore, LoHO achieves comparable performance with first-order based full fine-tuning, enjoying less memory consumption (e.g., reducing the GPU resources from 4 GPUs to a single GPU).

Methodology

In this section, we introduce the technical details of the proposed Low-order Hybrid Optimizer (LoHO), which contains two hybrid optimization strategies, as shown in Figure 1. The left part of Figure 1 shows the inter-layer hybrid optimization and the right part shows the intra-layer hybrid optimization. For completeness, we first introduce the MeZO (Malladi et al. 2024) method and then elaborate on the details of our methods.

The MeZO Solution

A traditional ZO gradient estimator known as Simultaneous Perturbation Stochastic Approximation (SPSA) (Spall and J.C 2002), and the corresponding SGD algorithm, ZO-SGD are the basic ZO technique used in MeZO (Malladi et al. 2024). Suppose $\mathcal{D} = \{(x_i, y_i)\}_{i \in \|\mathcal{D}\|}$ is a labeled dataset and $\mathcal{B} \in \mathcal{D}$ is a minibatch of data. We denote the loss on the minibatch as $\mathcal{L}(\theta; \mathcal{B})$, where $\theta \in \mathbb{R}^d$ denotes the parameters of a PLM and d is the number of the model parameters. Then the gradient estimate of Simultaneous Perturbation Stochastic Approximation (SPSA) is defined as follows:

$$\hat{\nabla}\mathcal{L}(\theta; \mathcal{B}) = \frac{\mathcal{L}(\theta + \epsilon z; \mathcal{B}) - \mathcal{L}(\theta - \epsilon z; \mathcal{B})}{2\epsilon} z \approx z z^T \nabla \mathcal{L}(\theta; \mathcal{B}), \quad (1)$$

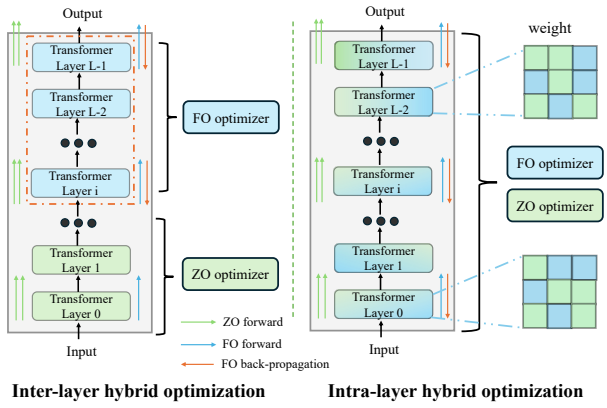


Figure 1: Low-order Hybrid Optimizer (LoHO).

where $z \in \mathbb{R}^d$ which is sampled from a Gaussian distribution, i.e, $z \sim \mathcal{N}(0, \mathbf{I}_d)$, and ϵ is the perturbation scale. SPSA only needs two forward passes of the model to estimate the gradients. Parameter perturbation of the model is performed before each forward pass, and both of the forward passes aim to compute a loss value. Then the gradients are estimated based on the difference between the two loss values.

Based on the SPSA, the zeroth-order SGD (ZO-SGD) optimizer updates the parameters as $\theta_{t+1} = \theta_t - \eta \hat{\nabla}\mathcal{L}(\theta; \mathcal{B}_t)$, where η is the learning rate, and $\mathcal{B}_t$ is the minibatch at the t-th iteration. However, the memory requirement for the vanilla ZO-SGD algorithm is twice as the inference because it necessitates the storage of $z \in \mathbb{R}^d$, whose size is the same as the model parameters. To address this issue, MeZO (Malladi et al. 2024) proposes a memory-efficient implementation for ZO-SGD based on the in-place operation, which only requires the same memory footprint as inference. Specifically, MeZO stores the seed of the random number generator used for sampling z, and resamples the same random noise z with the seed when z is needed. Besides, MeZO modifies the parameters in place when performing model perturbation and parameter updating.

Our Proposed LoHO Solution

Although MeZO offers benefits in memory efficiency, it has significant accuracy degradation compared with full fine-tuning with SGD or Adam. Moreover, it often exhibits "wasted" memory when the GPU is solely used for the fine-tuning task. In response, we propose a low-order hybrid optimizer (LoHO) to improve the performance of MeZO while keeping the memory usage within a budget, by integrating a ZO optimizer and a FO optimizer. There are two kinds of solutions for combining the ZO optimizer and the FO optimizer, i.e., the inter-layer solution and the intra-layer solution. The inter-layer solution has three variants: (1) Z+F which uses the ZO optimizer to train the shallow layers and the FO optimizer to train the deep layers; (2) F+Z which uses the FO optimizer to train the shallow layers and the ZO optimizer to train the deep layers; (3) F+Z+F which uses the FO optimizer to train the first several layers and the last sev-

eral layers, while using the ZO optimizer to train the middle layers. In comparison, the intra-layer solution updates some parameters in a layer with the ZO optimizer, and the rest are updated by the FO optimizer. Although these four solutions theoretically can be used for hybrid optimization, we explore the performance of inter-layer solution with Z+F and intra-layer solution in this paper considering the memory efficiency problem. Next, we present the details of where the peak memory footprint comes from during fine-tuning to explain why choosing these two solutions.

The memory consumption mainly comes from three parts: model parameters, gradients (may include the optimizer states if using Adam), and activations. We take a multi-layer perception (MLP) network as an example:

$$h_N = \mathbf{MLP}_N(\mathbf{MLP}_{N-1}(...(\mathbf{MLP}_2(\mathbf{MLP}_1(h_0)))...)), \quad (2)$$

where h_0 is the input, and N is the number of the MLP layers. The output of the i-th layer is $h_i = \mathbf{MLP}_i(h_{i-1}) = \sigma(W_i h_{i-1})$, where σ is the activation function, and W_i is the weight matrix with the bias term omitted here for simplicity. If we denote the output of the i-th layer before performing the activation function as $x_i = W_i h_{i-1}$, then in a back-propagation step with a loss function $\mathcal{L}$, the gradient of W_i is computed using the chain rule as follows.

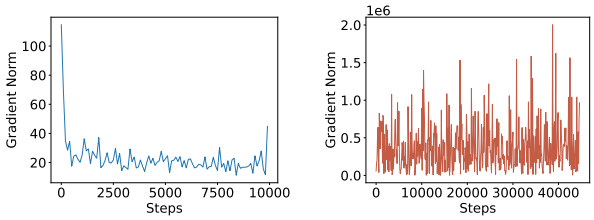
$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial W_i} &= \frac{\partial \mathcal{L}}{\partial h_i}\left(\prod_{j=i+1}^{N} \frac{\partial h_j}{\partial x_j}\frac{\partial x_j}{\partial h_{j-1}}\right)\frac{\partial h_i}{\partial x_i}\frac{\partial x_i}{\partial W_i} \\ &= \frac{\partial \mathcal{L}}{\partial h_i}\left(\prod_{j=i+1}^{N} \Delta\sigma_j W_j\right)\Delta\sigma_i h_{i-1}, \end{aligned} \quad (3)$$

where $\Delta\sigma$ is the derivative of σ. Based on this, activations $\{x_j\}_{j=i}^{N}$ are cached in order to compute the gradient of W_i even when $\{W_j\}_{j>i}$ are frozen.

Inter-layer Hybrid Optimization Upon analyzing the peak memory consumption during fine-tuning, it is evident that the positioning of the FO-optimized layers significantly impacts the overall memory footprint. The Z+F solution, which uses the ZO optimizer for the shallow layers and the FO optimizer for the deep layers, can reduce more memory consumption compared with F+Z and F+Z+F solutions, since they require caching more activations. Therefore, we adopt this solution, as shown in the left part of Figure 1. In this solution, the gradients computation of the ZO and FO optimizer are independent. For the deep layers that are optimized by the FO optimizer, we perform back-propagation to obtain their gradients. For the shallow layers that are optimized by the ZO optimizer, we perform two forward passes to estimate their gradients. Then we update the parameters for the shallow layers and the deep layers separately. In this paper, we integrate the recently proposed MeZO (Malladi et al. 2024) with FO optimizers such as SGD and Adam to demonstrate the effectiveness of inter-layer hybrid optimization, although our method can work with other zeroth-order optimizers (e.g. ZO-AdaMU (Jiang et al. 2024)).

Intra-layer Hybrid Optimization In addition to the inter-layer hybrid optimization, we explore another strategy, i.e., intra-layer hybrid optimization, as shown in the right part in Figure 1. A question raised here is how to determine the parameter subset in a layer to be updated by the FO optimizer. Although various strategies can be explored, we find that random selection can achieve promising performance. Specifically, for each weight matrix in a layer, we randomly select a ratio of parameters to be updated by the FO optimizer, and the ZO optimizer updates the remainder. Although this intra-layer hybrid optimization is simple, its efficient implementation is nontrivial. If we use the general implementation of training neural networks, we inherently necessitate the computation of gradients for the non-FO parameters as the gradient of each weight matrix is computed as shown in Equation (3). Such implementation results in degradation in memory efficiency, making the memory requirement similar to that of full fine-tuning.

To tackle this challenge, we employ a sparse training technique based on a sparse operations library (Nikdan, Tabesh, and Alistarh 2024), where the trainable parameters of the FO optimizer and their gradients are stored in a sparse format. Specifically, in the forward step, for a dense pre-trained weight W_i, we add it with a trainable sparse matrix Δ_i where the non-zero values indicate the FO trainable parameters of W_i. Then the output O is obtained by $O = X(W_i + \Delta_i)$, where X is the input. For the corresponding backward step, according to the chain rule, the gradients of X and Δ_i are $\frac{\partial \mathcal{L}}{\partial X} = \frac{\partial \mathcal{L}}{\partial O(W_i+\Delta_i)})^t$ and $\frac{\partial \mathcal{L}}{\partial \Delta_i} = X^t \frac{\partial \mathcal{L}}{\partial O}$ respectively. Note that $\frac{\partial \mathcal{L}}{\partial \Delta_i}$ is still sparse by using the Sampled Dense-Dense Matrix Multiplication (SDDMM) kernel. It multiplies two dense matrices where only the required elements of the output are computed. The gradient estimation for the ZO optimizer is similar to that in the inter-layer hybrid optimization. However, in the intra-layer hybrid optimization, we only estimate the gradients of the parameters that belong to the ZO optimizer. Through this implementation, we can control memory consumption by setting the proper ratio of parameters belonging to the FO optimizer.



(a) SGD (b) MeZO

Figure 2: Gradient Frobenius norm comparison during fine-tuning between SGD optimizer and MeZO optimizer.

Customized Learning Rates for Hybrid Optimization In LoHO, we can use the same learning rate for the FO optimizer and ZO optimizer. However, we find that the Frobenius norm of the gradients obtained by the ZO optimizer (e.g., MeZO) and the FO optimizer (e.g., SGD) are significantly different, as shown in Figure 2. We transform the whole model parameters into a vector and calculate the Frobenius norm of its gradient $G \in \mathbb{R}^d$ as $\|G\|_F =$

$\sqrt{\sum_{i=1}^{d} |G_i|^2}$, where $\|\|_F$ denotes the Frobenius norm. The Frobenius norm of the gradients of MeZO is significantly larger than that of SGD due to two main reasons. First, the absolute magnitudes of the gradients in SGD often decrease from the deep layers to the shallow layers, because most of the absolute gradient values are smaller than one, and performing the multiplication of the chain rule shown in Equation (3) makes the absolute gradient values smaller. In contrast, the calculation of the gradients of each layer in MeZO is independent, and thus the absolute magnitudes of the gradients in different layers maintain the same scales. Second, there is a perturbation scale ϵ in the gradient estimation function (cf. Equation 1) which is commonly set to a value much smaller than one (e.g., 0.01 or 0.001) (Malladi et al. 2024), making the estimated gradient norm relatively larger than that in back-propagation. Therefore, we need to set different learning rates for the FO optimizer and the ZO optimizer to balance the parameter update amplitude of the two kinds of optimizers. Furthermore, as the FO optimizer often converges faster than the ZO optimizer, we can speed up the convergence rate in the hybrid optimization if we set the learning rates suitably. Consequently, the slow convergence problem of MeZO can be alleviated. In practice, the learning rate of the ZO optimizer can be configured to be several orders of magnitude lower than that of the FO optimizer. This customized learning rate setting can be utilized in both inter-layer hybrid optimization and intra-layer optimization.

Experiments

Datasets and Baselines

Datasets Following the settings as the work of Malladi et al. (2024), we chose RoBERTa-large (Liu et al. 2019), OPT-13B and OPT-30B (Zhang et al. 2023) as the pre-trained backbones to conduct the experiments. Most of the datasets we selected exhibit performance gaps exceeding 4% between MeZO and full fine-tuning with Adam. Thus, for the RoBERTa-large experiments, we used the following datasets: SST-2 (Socher et al. 2013), RTE (Cer et al. 2017), MNLI (Williams, Nangia, and Bowman 2018) and SNLI (Bowman et al. 2015). We followed the settings of Malladi et al. (2024), which used 512 examples per class for both training and validation. Similarly, for the OPT experiments, we used the following datasets, including RTE (Cer et al. 2017), BoolQ (Clark et al. 2019), CB (De Marneffe, Simons, and Tonhauser 2019), MultiRC (Khashabi et al. 2018) and WIC (Pilehvar and Camacho-Collados 2018). We randomly sampled 1,000 examples for training, 500 examples for validation, and 1,000 examples for testing, which is the same as MeZO (Malladi et al. 2024).

Baselines The baselines in our experiments include:

- Zero-shot: it directly performs inference based on a single prompt without fine-tuning.
- ICL (In context learning): it performs inference with prompts based on several labeled samples as input.
- MeZO (Malladi et al. 2024): it updates all the model parameters with the memory-efficient ZO-SGD optimizer.

Table 1: Experiments on the RoBERTa-large model. † indicates results reported in MeZO (Malladi et al. 2024). Memory budget: a single RTX 4090 GPU with 24GB memory.

Dataset	SST-2	RTE	MNLI	SNLI	Average
Zero-shot†	79.0	51.4	48.8	50.2	57.4
MeZO†	$93.3_{0.7}$	$78.6_{2.0}$	$78.3_{0.5}$	$83.0_{1.0}$	83.3
MeZO-Adam†	$93.3_{0.6}$	$79.2_{1.2}$	$79.6_{0.4}$	$85.3_{0.8}$	84.4
LoHO-SGD$_{inter}$	$94.6_{0.5}$	$79.8_{1.8}$	$80.6_{1.9}$	$85.8_{0.6}$	85.2
LoHO-Adam$_{inter}$	$\mathbf{95.0}_{0.6}$	$\mathbf{81.1}_{1.1}$	$\mathbf{83.2}_{0.4}$	$\mathbf{87.6}_{0.9}$	**86.7**
LoHO-Adam$_{intra}$	$94.9_{0.5}$	$80.4_{0.7}$	$82.2_{0.9}$	$\mathbf{87.6}_{0.4}$	86.3

- MeZO-Adam (Malladi et al. 2024): it updates all the model parameters with a variant of MeZO which uses Adam optimizer instead of SGD. As the naive implementation of MeZO-Adam requires additional memory to store the gradient momentum estimates, Malladi et al. (2024) only investigate its performance on the RoBERTa-large experiments. We followed their setting in this work.

Our methods include (i) LoHO-SGD$_{inter}$ and LoHO-Adam$_{inter}$, which are inter-layer hybrid optimization methods using SGD and Adam as the FO optimizer respectively, and (ii) LoHO-Adam$_{intra}$, which is an intra-layer hybrid optimization method using Adam as the FO optimizer.

Experimental Setup

A natural question raised in inter-layer hybrid optimization is how to set the number of FO-optimized layers. Although one can perform experiments to find out the best setting of the number of FO-optimized layers, in this work, our main experiments investigate the performance with the maximum number of FO-optimized layers that can be used in a single GPU. There are two reasons for this setting. On the one hand, using more FO layers is more promising for achieving better performance. On the other hand, we can best utilize the GPU memory as much as possible, since the free/unused memory is wasted when the GPUs are solely used for the specific task to maximize the computation efficiency. Other settings about the number of FO layers are explored in the analysis experiments. Another question is how to set the ratio of parameters to be updated by the FO optimizer in each layer. To make a fair comparison between inter-layer and intra-layer optimization, the only difference between LoHO-Adam$_{inter}$ and LoHO-Adam$_{intra}$ is that a ratio of parameters of the FO layers in LoHO-Adam$_{intra}$ is optimized by the ZO optimizer. For more details about the settings, please refer to the Appendix A. Our code is available at https://github.com/Chan-1996/LoHO.

Main Results

Results on RoBERTa-large The performance comparison between different methods on RoBERTa-large is shown in Table 1. First, we can see that the performance of LoHO-SGD$_{inter}$ and LoHO-Adam$_{inter}$ substantially outperform MeZO and MeZO-Adam by 1.9% and 2.3% on average accuracy respectively, and LoHO-Adam$_{intra}$ also outperforms MeZO-Adam by 1.9% on average accuracy, showing the potential of making more efficient use of memory. Notably,

Table 2: Experiments on the OPT-13B model. Memory budget: a single A800 GPU with 80GB memory.

Dataset	RTE	CB	BoolQ	WIC	MultiRC	Average
Zero-shot†	59.6	46.4	59.0	55.0	46.9	53.4
ICL†	62.1	57.1	66.9	50.5	53.1	57.9
MeZO†	66.1	67.9	67.6	61.1	60.1	64.6
LoHO-SGD$_{inter}$	76.2	**71.4**	**67.8**	63.6	58.6	67.5
LoHO-Adam$_{inter}$	**77.6**	69.6	64.7	**65.4**	**67.2**	**68.9**
LoHO-Adam$_{intra}$	75.5	**71.4**	64.0	**65.4**	66.5	68.6

Table 3: Experiments on the OPT-30B model. Memory budget: a single A800 GPU with 80GB memory.

Dataset	RTE	BoolQ	WIC	Average
Zero-shot†	52.0	39.1	50.2	47.1
ICL†	66.8	66.2	51.3	61.4
MeZO†	66.4	67.2	56.3	63.3
LoHO-SGD$_{inter}$	68.0	**68.1**	61.0	65.7
LoHO-Adam$_{inter}$	**72.6**	67.2	60.7	**66.8**
LoHO-Adam$_{intra}$	71.5	65.8	**62.7**	66.7

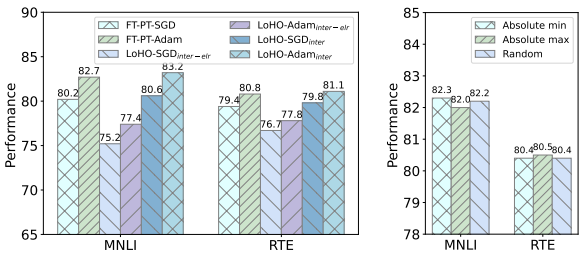
the performance difference between MeZO-Adam (resp. MeZO) and LoHO-Adam$_{inter}$ (resp. LoHO-SGD$_{inter}$) is statistically significant with a p-value=1.1e-2 (resp. 1.6e-2) < 0.05 by a two-tailed t-test. Second, LoHO-Adam$_{inter}$ outperforms LoHO-SGD$_{inter}$, indicating that using the momentum information can improve the quality of the estimated gradients. Finally, LoHO-Adam$_{intra}$ achieves comparable performance with LoHO-Adam$_{inter}$ while involving less trainable parameters for the FO optimizer.

Results on OPT-13B and OPT-30B The experimental results of different methods on OPT-13B and OPT-30B are shown in Table 2 and Table 3 respectively. Our proposed LoHO-SGD$_{inter}$ and LoHO-Adam$_{inter}$ outperform MeZO by 2.9% and 4.3% in average accuracy respectively on OPT-13B. Besides, our methods consistently outperform MeZO on the OPT-30B model. However, the predictive accuracies of our methods on OPT-30B are lower than that on OPT-13B, due to the fewer FO layers used in the OPT-30B model as we set the memory budget to a single GPU. In summary, our methods are effective both for medium-sized language models and large language models compared with MeZO, and LoHO-Adam$_{inter}$ achieve the best performance among different variants of the proposed LoHO.

Ablation Study

We performed the ablation study and presented the results on two representative datasets, i.e., the MNLI dataset and RTE dataset using RoBERTa-large as the backbone.

Inter-layer hybrid optimization The ablation baseline for the proposed inter-layer hybrid optimization include: FT-PT-SGD which freezes the layers optimized by MeZO in LoHO-SGD$_{inter}$ method and only updates the rest of the layers using SGD optimizer; FT-PT-Adam which freezes the layers optimized by MeZO in LoHO-Adam$_{inter}$ method and only updates the rest layers using Adam optimizer; LoHO-SGD$_{inter-elr}$ and LoHO-Adam$_{inter-elr}$ which uses



(a) Inter-layer (b) Intra-layer

Figure 3: Ablation study of hybrid optimization.

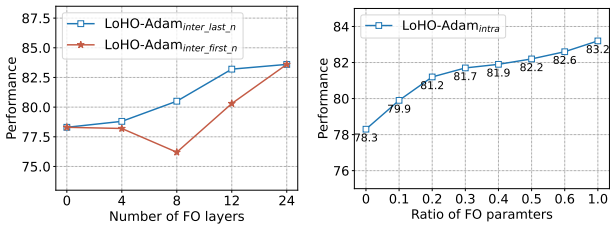
the equal learning rate for the ZO optimizer and FO optimizer. From Figure 3 (a), we can observe that methods based on hybrid optimization outperform their corresponding variant based on a single optimizer. This is reasonable as in hybrid optimization, the searching space of the model is larger and it is more promising to approach the global optimal during the training process. Another finding is that due to the noticeable difference in the gradient norm obtained by the ZO and the FO optimizer, the experimental results of LoHO-SGD$_{inte-elr}$ and LoHO-Adam$_{inter-elr}$ are unsatisfactory. In summary, the ablation study further indicates the effectiveness of the proposed method in this work.

Intra-layer hybrid optimization In LoHO, we used random selection to obtain a subset of parameters to be updated by the FO optimizer. Here, we investigate other strategies to select the FO-optimized parameters, including absolute-min and absolute-max. Concretely, the absolute-min (resp. absolute-max) strategy leverages a ratio of parameters with minimal (resp. maximal) absolute values to be updated by the FO optimizer. As shown from Figure 3 (b), the three strategies achieve comparable performance on both the MNLI and RTE datasets. Therefore, we adopt the random selection strategy, instead of other heuristic strategies.

Analysis on Different Settings for LoHO

Effect of the Number of FO Layers We study the effect of different settings of FO layers in the inter-layer hybrid optimization. In particular, we investigate utilizing different numbers of FO layers and different positions of FO layers for LoHO-Adam$_{inter}$ on the MNLI dataset (i.e., LoHO-Adam$_{inter_last_n}$ and LoHO-Adam$_{inter_first_n}$). As shown in Figure 4 (a), when the FO layers are the deep layers, the model accuracy consistently improves on the MNLI dataset as the number of FO layers increases, which is sensible. However, placing FO layers in shallow layers significantly reduces predictive accuracy compared to using the FO optimizer for deep layers in most cases. One possible reason is that the deep layers have a greater impact than the shallow layers for this task. We leave this in future work to investigate the importance and effect of each layer in the PLM.

Effect of the Ratio of FO-optimized Parameters We also study the effect of the ratio of FO-optimized parameters in the proposed intra-layer hybrid optimization. As presented in Figure 4 (b), we performed this analysis on the



(a) Effect of FO layers (b) Effect of FO parameter ratio

Figure 4: Effect of FO layers and the ratio of FO parameters.

Table 4: Memory usage comparison. The max sequence lengths of RTE and MultiRC datasets are 288 and 746 respectively. bz denotes batch size.

Model	Method	Memory (GB)	GPU	#FO layers
RoBERTa (RTE) (bz=64)	MeZO	4.4	RTX 4090	0
	MeZO-Adam	9.3	RTX 4090	0
	LoHO-SGD$_{inter}$	20.0	RTX 4090	12
	LoHO-Adam$_{inter}$	20.5	RTX 4090	12
	LoHO-Adam$_{intra}$	19.4	RTX 4090	12
OPT-13B (MultiRC) (bz=16)	MeZO	71.0	A800	0
	LoHO-SGD$_{inter}$	76.9	A800	2
	LoHO-Adam$_{inter}$	78.9	A800	2
	LoHO-Adam$_{intra}$	76.6	A800	2
OPT-30B (RTE) (bz=8)	MeZO	65.1	A800	0
	LoHO-Adam$_{inter}$	78.9	A800	4
	LoHO-Adam$_{intra}$	73.5	A800	4

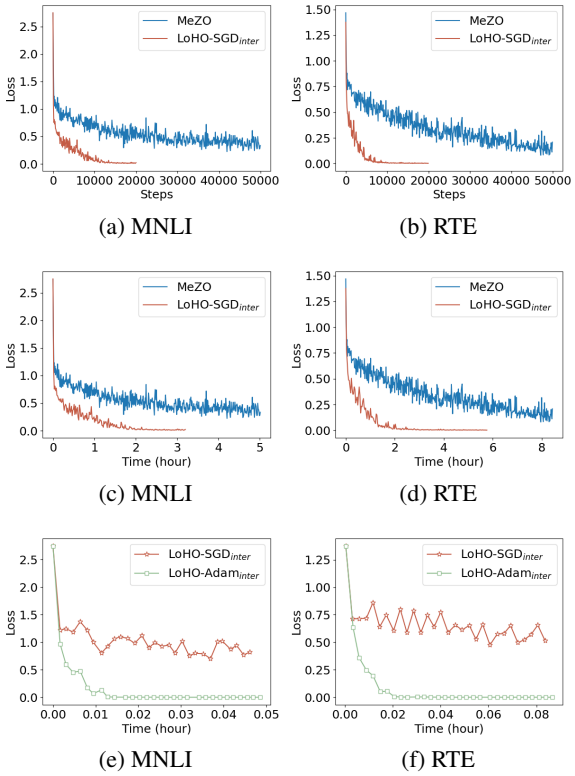
MNLI dataset to demonstrate the model quality across various ratios of FO-optimized parameters in each FO layer. With the ratio of FO-optimized parameters increasing, the model performance improves consistently. Moreover, using 60% of FO-optimized parameters can achieve comparable performance with that of using 100% FO-optimized parameters, which is akin to inter-layer hybrid optimization. However, improving the ratio of FO-optimized parameters also requires more memory consumption. Therefore, it is a trade-off between memory requirement and model performance.

Analysis on Memory Usage

Here, we show the memory profiling of fine-tuning using different backbones in Table 4. Specifically, we selected the RTE and MultiRC datasets to perform the memory profiling. The number of the FO layers depends on the sequence length of the specific dataset. For example, for the OPT-30B model, the maximum number of FO layers is four using a single A800 GPU. Following Malladi et al. (2024), we used Nvidia's *nvidia-smi* command to monitor the peak GPU memory usage. From the results, we can see that although requiring additional memory compared with MeZO, our hybrid optimization method still can be fine-tuned in a single GPU. The experimental results and the memory profiling results demonstrate that trading off between performance and memory is effective.

Analysis on Convergence Rate

In this section, we demonstrate the convergence rate comparison between MeZO and our hybrid optimizer (here we use inter-layer hybrid optimization). For a fair comparison, we compare MeZO with LoHO-SGD$_{inter}$ since both of them use SGD optimizer. We show this comparison from two perspectives (loss $vs.$ training step and loss $vs.$ training time) on two datasets in Figure 5 (more results can be found in Appendix B). It can be observed that the convergence rate of LoHO-SGD$_{inter}$ is noticeably faster than that of MeZO on these two datasets. Although LoHO-SGD$_{inter}$ needs to perform three forward passes and a backward pass in a training step, while MeZO only needs to perform two forward passes, LoHO-SGD$_{inter}$ can reduce substantial training steps compared with MeZO. As a result, the overall training time of LoHO-SGD$_{inter}$ is shorter than that of MeZO. This comparison demonstrates that LoHO-SGD$_{inter}$ has its advantages in terms of accuracy and time efficiency. We also compare the convergence rate between LoHO-SGD$_{inter}$ and LoHO-Adam$_{inter}$, as shown in the lower part of Figure 5. The convergence rate of LoHO-Adam$_{inter}$ is faster than that of LoHO-SGD$_{inter}$, which meets our expectations as Adam also converges faster than SGD.



(a) MNLI (b) RTE

(c) MNLI (d) RTE

(e) MNLI (f) RTE

Figure 5: Convergence rate comparison on RoBERTa-large.

Comparison with First-order Fine-tuning

Here, we compare LoHO with full fine-tuning using first-order optimizers. We present the performance comparison and memory usage in Table 5 and Table 6 respectively. Full FT (SGD) and Full FT (Adam) are full fine-tuning methods using the SGD optimizer and the Adam optimizer respectively. LoHO achieves comparable performance with

Table 5: Performance comparison between LoHO and fine-tuning with first-order optimizers on RoBERTa-large.

Dataset	SST-2	RTE	MNLI	SNLI	Average
Full FT (SGD)	95.1	81.1	80.5	85.5	85.6
Full FT (Adam)	94.9	83.0	83.6	87.2	87.2
LoHO-SGD$_{inter}$	94.6	79.8	80.6	85.8	85.2
LoHO-Adam$_{inter}$	95.0	81.1	83.2	87.6	86.7

Table 6: Memory usage comparison.

Model	Method	Mem.(GB)	GPU	#FO layers
RoBERTa	Full FT (SGD)	43.1	A800	24
(RTE)	Full FT (Adam)	45.8	A800	24
(bz=64)	LoHO-SGD$_{inter}$	20.0	RTX 4090	12
	LoHO-Adam$_{inter}$	20.5	RTX 4090	12
OPT-13B	Full FT (Adam) (bz=1)	316.0	4 $\times$ A800	40
(MultiRC)	LoHO-Adam$_{inter}$(bz=16)	78.9	1 $\times$ A800	2

Full FT (Adam), with over 50% less memory usage on RoBERTa-large. On OPT-13B, it needs four A800 GPUs to run Full FT (Adam) with a batch size of one, while LoHO can be run with a single A800 GPU and a batch size of 16. Besides, LoHO can achieve comparable predictive accuracy or even outperform Full FT (Adam) on some tasks with OPT-13B backbone. For example, on the RTE dataset whose sequence length is shorter than 300 (so we can use more FO layers in an A800 GPU), LoHO achieves 77.6% while Full FT (Adam) achieves 70.8% in accuracy. However, on datasets with longer sequence lengths, such as MultiRC, the accuracy of LoHO and Full FT (Adam) is 67.2% and 71.1%, respectively, as we can only use two FO layers within the memory budget. Nevertheless, if improving the memory budget, LoHO can approach the performance of Full FT (Adam) on datasets with longer sequence lengths.

Related Work

Here, we discuss the existing work on zeroth-order and first-order optimization and hybrid optimizers.

Zeroth-order Optimization

ZO optimization has been extensively studied in the realm of convex and strongly convex objectives (Jamieson, Nowak, and Recht 2012; Agarwal et al. 2012; Raginsky and Rakhlin 2011). As a kind of backpropagation-free optimization method, ZO optimization approximates gradients based on finite differences. It has long been studied and achieved remarkable success in solving various machine learning problems, such as adversarial attack and defense (Tu et al. 2019; Ilyas et al. 2018; Shu et al. 2022; Zhao et al. 2019), automated machine learning (Gu et al. 2021; Wang et al. 2022), policy search in reinforcement learning (Vemula, Sun, and Bagnell 2019), visual prompting for transfer learning (Tsai, Chen, and Ho 2020) and so on.

Recently, Malladi et al. (2024) propose a memory-efficient zeroth-order optimizer (MeZO) that adapts the classical ZO-SGD method to operate in place, making large language model fine-tuning as efficient as inference. Despite its memory efficiency, MeZO still exhibits a significant performance drop in terms of predictive accuracy and slower convergence speed compared to full fine-tuning on various tasks. Several techniques have been developed to address these problems, including Sparse MeZO (Liu et al. 2024) which leverages gradient sparsity to improve both predictive accuracy and convergence speed of MeZO, ZO-AdaMU (Jiang et al. 2024) which adapts the simulated perturbation with momentum in the stochastic approximation to improve convergence stability and rate in ZO-SGD, and HiZOO (Zhao et al. 2024) which leverages the diagonal Hessian to enhance the convergence and predictive accuracy of MeZO. Although efforts have been invested to improve the MeZO, there is still a performance gap between these methods and first-order based methods on complex tasks.

First-order Optimization

There are many classical first-order optimization methods, such as SGD, Momentum, Adagrad (Duchi, Hazan, and Singer 2011) and ADADELTA (Zeiler 2012), which are fundamental in many research areas such as computer vision and natural language processing. However, with the emergence of large-scale models, the effectiveness of such conventional first-order optimization methods has been challenged. This is because their convergence rate needs to be improved to accelerate the training of large-scale models. Adam (Kingma and Ba 2014) has emerged and become a dominant optimizer for training and fine-tuning large-scale models due to its fast convergence speed. To alleviate the over-fitting problem, AdamW (Loshchilov and Hutter 2018) proposes to add a weight decay coefficient in Adam. To accelerate the training of large-scale models with large batch sizes, LAMB (You et al. 2019) proposes to employ the principled inter-layer adaptation strategy.

Hybrid Optimization

Hybrid optimization has not been extensively studied. Landro, Gallo, and La Grassa (2020) combine SGD and Adam by using constant weights to balance the contributions of gradient estimates from each optimizer. Similar to the standard fine-tuning with FO optimizers, this method requires significant memory. Ansaripour et al. (2022) propose hybrid optimization at the model level under the decentralized optimization setting in a distributed system (some agents are optimized by ZO; others are optimized by FO), which is significantly different from ours. The hybrid optimization proposed in our work is layer level (i.e., inter-layer strategy) or weight level (i.e., intra-layer strategy).

Conclusion

In this work, we have proposed a Low-order Hybrid Optimizer (LoHO) for language model fine-tuning. In LoHO, we have proposed two hybrid optimization strategies namely inter-layer hybrid optimization and intra-layer hybrid optimization. The two proposed strategies are designed to maximize memory usage to the fullest extent, by considering the gradient back-propagation and leveraging the gradient sparsity respectively. We have experimentally demonstrated the effectiveness of LoHO both in predictive accuracy and convergence rate on various datasets and different pre-trained backbones within a memory budget.

Acknowledgments

This work is supported by the Guangzhou Industrial Information and Intelligent Key Laboratory Project (No. 2024A03J0628). It is also funded by the Guangzhou Science and Technology Development Projects (No. 2023A03J0143 and No. 2024A04J4458) and the NSFC Project (No. 62306256).

References

Agarwal, A.; Bartlett, P. L.; Ravikumar, P.; and Wainwright, M. J. 2012. Information-Theoretic Lower Bounds on the Oracle Complexity of Stochastic Convex Optimization. *IEEE Transactions on Information Theory*, 5(58): 3235–3249.

Ansaripour, M.; Talaei, S.; Nadiradze, G.; and Alistarh, D. 2022. Hybrid Decentralized Optimization: Leveraging Both First-and Zeroth-Order Optimizers for Faster Convergence. *arXiv e-prints*, arXiv–2210.

Bowman, S. R.; Angeli, G.; Potts, C.; and Manning, C. D. 2015. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.

Cer, D. M.; Diab, M. T.; Agirre, E.; Lopez-Gazpio, I.; and Specia, L. 2017. SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation, SemEval@ACL 2017*, 1–14.

Clark, C.; Lee, K.; Chang, M.-W.; Kwiatkowski, T.; Collins, M.; and Toutanova, K. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

De Marneffe, M.-C.; Simons, M.; and Tonhauser, J. 2019. The commitmentbank: Investigating projection in naturally occurring discourse. In *proceedings of Sinn und Bedeutung*, volume 23, 107–124.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).

Gu, B.; Liu, G.; Zhang, Y.; Geng, X.; and Huang, H. 2021. Optimizing large-scale hyperparameters via automated learning algorithm. *arXiv preprint arXiv:2102.09026*.

Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; de Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; and Gelly, S. 2019. Parameter-Efficient Transfer Learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, volume 97, 2790–2799.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *The Tenth International Conference on Learning Representations, ICLR 2022*.

Ilyas, A.; Engstrom, L.; Athalye, A.; and Lin, J. 2018. Black-box adversarial attacks with limited queries and information. In *International conference on machine learning*, 2137–2146. PMLR.

Jamieson, K. G.; Nowak, R.; and Recht, B. 2012. Query complexity of derivative-free optimization. *Advances in Neural Information Processing Systems*, 25.

Jiang, S.; Chen, Q.; Pan, Y.; Xiang, Y.; Lin, Y.; Wu, X.; Liu, C.; and Song, X. 2024. ZO-AdaMU Optimizer: Adapting Perturbation by the Momentum and Uncertainty in Zeroth-Order Optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 18363–18371.

Khashabi, D.; Chaturvedi, S.; Roth, M.; Upadhyay, S.; and Roth, D. 2018. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 252–262.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Landro, N.; Gallo, I.; and La Grassa, R. 2020. Mixing Adam and SGD: A combined optimization method. *arXiv preprint arXiv:2011.08042*.

Li, X. L.; and Liang, P. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers)*, 4582–4597.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, abs/1907.11692.

Liu, Y.; Zhu, Z.; Gong, C.; Cheng, M.; Hsieh, C.-J.; and You, Y. 2024. Sparse MeZO: Less Parameters for Better Performance in Zeroth-Order LLM Fine-Tuning. *arXiv preprint arXiv:2402.15751*.

Loshchilov, I.; and Hutter, F. 2018. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

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

Nikdan, M.; Tabesh, S.; and Alistarh, D. 2024. Rosa: Accurate parameter-efficient fine-tuning via robust adaptation. *arXiv preprint arXiv:2401.04679*.

Pilehvar, M. T.; and Camacho-Collados, J. 2018. WiC: the word-in-context dataset for evaluating context-sensitive meaning representations. *arXiv preprint arXiv:1808.09121*.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.*, 21: 140:1–140:67.

Raginsky, M.; and Rakhlin, A. 2011. Information-based complexity, feedback and dynamics in convex programming. *IEEE Transactions on Information Theory*, 57(10): 7036–7056.

Shu, Y.; Dai, Z.; Sng, W.; Verma, A.; Jaillet, P.; and Low, B. K. H. 2022. Zeroth-order optimization with trajectory-informed derivative estimation. In *The Eleventh International Conference on Learning Representations*.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A. Y.; and Potts, C. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013*, 1631–1642.

Spall; and J.C. 2002. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3): 332–341.

Tsai, Y.-Y.; Chen, P.-Y.; and Ho, T.-Y. 2020. Transfer learning without knowing: Reprogramming black-box machine learning models with scarce data and limited resources. In *International Conference on Machine Learning*, 9614–9624. PMLR.

Tu, C.-C.; Ting, P.; Chen, P.-Y.; Liu, S.; Zhang, H.; Yi, J.; Hsieh, C.-J.; and Cheng, S.-M. 2019. Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 742–749.

Vemula, A.; Sun, W.; and Bagnell, J. 2019. Contrasting exploration in parameter and action space: A zeroth-order optimization perspective. In *The 22nd International Conference on Artificial Intelligence and Statistics*, 2926–2935. PMLR.

Wang, X.; Guo, W.; Su, J.; Yang, X.; and Yan, J. 2022. Zarts: On zero-order optimization for neural architecture search. *Advances in Neural Information Processing Systems*, 35: 12868–12880.

Williams, A.; Nangia, N.; and Bowman, S. R. 2018. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018*, 1112–1122.

You, Y.; Li, J.; Reddi, S.; Hseu, J.; Kumar, S.; Bhojanapalli, S.; Song, X.; Demmel, J.; Keutzer, K.; and Hsieh, C.-J. 2019. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*.

Zeiler, M. D. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhang, S.; Roller, S.; Goyal, N.; Artetxe, M.; Chen, M.; Chen, S.; Dewan, C.; Diab, M.; Li, X.; Lin, X. V.; et al. 2023. Opt: Open pre-trained transformer language models, 2022. *URL https://arxiv. org/abs/2205.01068*, 3: 19–0.

Zhao, P.; Liu, S.; Chen, P.-Y.; Hoang, N.; Xu, K.; Kailkhura, B.; and Lin, X. 2019. On the design of black-box adversarial examples by leveraging gradient-free optimization and operator splitting method. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 121–130.

Zhao, Y.; Dang, S.; Ye, H.; Dai, G.; Qian, Y.; and Tsang, I. W. 2024. Second-Order Fine-Tuning without Pain for LLMs: A Hessian Informed Zeroth-Order Optimizer. *arXiv preprint arXiv:2402.15173*.