Reducing Fine-Tuning Memory Overhead by Approximate and Memory-Sharing Backpropagation

Yuchen Yang [1] **Yingdong Shi** [2] **Cheems Wang** [3] **Xiantong Zhen** [4] **Yuxuan Shi** [1] **Jun Xu** [1]

Abstract

Fine-tuning pretrained large models to downstream tasks is an important problem, which however suffers from huge memory overhead due to large-scale parameters. This work strives to reduce memory overhead in fine-tuning from perspectives of activation function and layer normalization. To this end, we propose the Approximate Backpropagation (Approx-BP) theory, which provides the theoretical feasibility of decoupling the forward and backward passes. We apply our Approx-BP theory to backpropagation training and derive memory-efficient alternatives of GELU and SiLU activation functions, which use derivative functions of ReLUs in the backward pass while keeping their forward pass unchanged. In addition, we introduce a Memory-Sharing Backpropagation strategy, which enables the activation memory to be shared by two adjacent layers, thereby removing activation memory usage redundancy. Our method neither induces extra computation nor reduces training efficiency. We conduct extensive experiments with pretrained vision and language models, and the results demonstrate that our proposal can reduce up to $\sim 30\%$ of the peak memory usage. Our code is released at github.

1. Introduction

Ever since the emergence of large models like GPTs (Radford et al., 2019), how to fine-tune them efficiently on downstream tasks has become an important problem (Hu et al., 2022). However, the unaffordable activation memory overhead largely limits their applications to memory-constrained hardware like edge devices. For this, it is essential to investigate memory reduction strategies for parameter-efficient fine-tuning (PEFT). A common strategy of PEFT (Houlsby et al., 2019; Liu et al., 2021a; Hu et al., 2022; Jia et al., 2022) is parameter freezing, which mainly reduces activation memory usage brought by linear projection layers. However, the activation memory overhead from non-linear modules in transformers still occupies a large part of the total usage, *e.g.*, $\sim 63\%$ in ViT (Dosovitskiy et al., 2021) and $\sim 74\%$ in LLaMA (Touvron et al., 2023) (Figure 2).

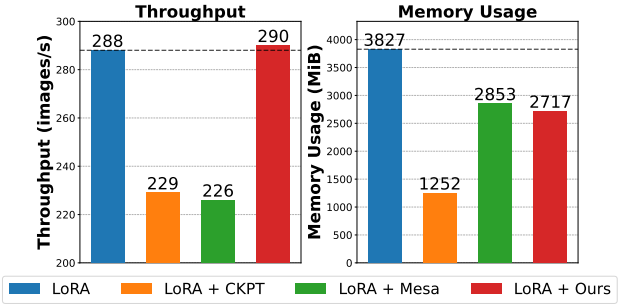

Figure 1. **Throughput (images/s) and memory usage (MiB) with LoRA (Hu et al., 2022) (rank $= 4$, batch size $= 64$) on fine-tuning pretrained ViT-B (Dosovitskiy et al., 2021) with CIFAR10/100 (Krizhevsky et al., 2009) and FGVC (Jia et al., 2022).** "LoRA + CKPT": LoRA with gradient-checkpointing (Chen et al., 2016) on every block. "LoRA + Mesa": LoRA with 8-bit activation quantization on GELU and LayerNorm (Pan et al., 2021). "LoRA + Ours": LoRA with our ReGELU2 and MS-LN. More details are provided in Section 6.

There are three main non-linear modules in a typical transformer: self-attention, activation function, and layer normalization. Previous efforts have been mainly devoted to reducing the memory complexity of vanilla self-attention (Child et al., 2019; Kitaev et al., 2020; Beltagy et al., 2020; Dao et al., 2022). Among them, FlashAttention (Dao et al., 2022) is a highly optimized implementation with linear memory complexity. Hereafter, transformers put a large part of activation memory usage in activation function and layer normalization. Nevertheless, these two modules draw little attention on activation memory reduction, though they are widely used in transformers (Liu et al., 2019; Touvron et al., 2022) and others (Tolstikhin et al., 2021; Yu et al., 2022).

[1]School of Statistics and Data Science, Nankai University, Tianjin, China [2]School of Information Science and Technology, ShanghaiTech University, Shanghai, China [3]Department of Automation, Tsinghua University, Peking, China [4]Central Research Institute, United Imaging Healthcare, Co., Ltd.. Correspondence to: Jun Xu <nankaimathxujun@gmail.com>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

Non-linear activation functions like GELU (Hendrycks & Gimpel, 2023) and SiLU (Hendrycks & Gimpel, 2023; Elfwing et al., 2017; Ramachandran et al., 2017) need the whole input tensor to compute the gradients in regular backpropagation (BP), and suffer from huge activation memory usage. To avoid performance degradation, one may prefer to initialize the large model with the pretrained weights before fine-tuning it. To this end, it is safe to avoid changing the forward pass of activation functions. A natural yet crucial question arises: ***is it possible to reduce the activation memory usage by only changing the backward pass?***

This paper provides positive feedback to the above question by developing an approximate backward pass as an alternative to the exact BP process. To achieve this goal, we propose safely decoupling the forward and backward passes with a new Approximate BackPropagation (Approx-BP) theory. Our Approx-BP theory reveals that if primitive functions are close in functional space, then derivatives can be substituted for each other in the training. Based on our Approx-BP theory, the pretrained models using a highly non-linear activation function could replace their non-linear derivatives with a moderately linear derivative that requires less activation memory. We apply this theory to GELU and SiLU, and derive our ReGELU2 and ReSiLU2 in which the activation memory usage is only 2 bits per element.

As for layer normalization (Ba et al., 2016), we observe redundancy in the activation memory within it and the subsequent linear layers. To avoid this redundancy, we introduce a Memory-Sharing BP (MS-BP) strategy and establish a sufficient condition under which a layer can share its activation memory with the following layer. By merging the affine parameters of LayerNorm and RMSNorm (Zhang & Sennrich, 2019) into the following linear layers with an adapted derivative calculation manner, we propose memory-sharing LayerNorm (MS-LN) and RMSNorm (MS-RMSNorm) to satisfy the condition of our MS-BP strategy and share activation memory usage with the following linear layers.

Without any extra computation cost, our method will not affect the training throughput of full fine-tuning or PEFT methods like LoRA while further reducing their activation memory usage (Figure 1). Experiments on ViT (Dosovitskiy et al., 2021) and LLaMA (Touvron et al., 2023) show that our method can reduce their peak GPU memory usage in fine-tuning by $\sim 30\%$, with comparable performance to those by full fine-tuning, LoRA (Hu et al., 2022), LoRA-FA (Zhang et al., 2023a), or QLoRA (Dettmers et al., 2023).

In summary, the contributions of this work are three-fold:

- We propose the Approximate Backpropagation (Approx-BP) theory, which supports the feasibility of decoupling the forward and backward passes in backpropagation training. Under our Approx-BP, we derive our ReGELU2 and ReSiLU2 as alternatives of GELU and SiLU, respectively, to share their primitives while possessing a 2-bit step function as the derivative.
- We provide a Memory-Sharing BP (MS-BP) strategy and apply it to layer normalization. The resulting MS-LN and MS-RMSNorm remove the redundant activation memory with the following linear layers.
- Our method has no extra computational cost and does not affect the training throughput or the fine-tuning networks' inference accuracy.

2. Related Work

Here, we briefly introduce the related research on reducing the activation memory usage in network training.

2.1. Activation Recomputation

The activation recomputation (Chen et al., 2016) (also called gradient checkpointing) avoids saving the intermediate activation in the forward pass of a network layer by recomputing it in the backward pass. It is widely used on self-attention (Korthikanti et al., 2023) to reduce the activation memory usage but at the cost of extra computation.

Later, FlashAttention (Dao et al., 2022) optimizes the complexity of activation recomputation in self-attention, which is implemented by an efficient CUDA kernel. Due to preserving the training process while effectively reducing the activation memory, gradient checkpointing is widely used in fine-tuning large models with GPU constraints. However, it suffers from a remarkable side affect of additional training duration, *e.g.*, $\sim 20\%$ in LoRA fine-tuning (Figure 1) when used at every block of the fine-tuning network.

Our method also changes the regular BP process, but avoids recomputation to preserve the training efficiency.

2.2. Activation Quantization

Network training in mixed precision (Micikevicius et al., 2017) is feasible to execute most of computations in half-precision floats (16-bit) in forward and backward passes. Besides reducing memory usage, this can also accelerate the training speed since half-precision computation is supported inherently in modern GPUs. 8-bit training is allowed in CNNs with tolerant performance loss (Banner et al., 2018).

To avoid global quantization in both forward and backward passes, activation compression training (ACT) (Chakrabarti & Moseley, 2019) executes the forward pass in the originally high precision, then stores activation tensors by low precision quantization, and finally dequantizes these tensors back to the original precision in the backward pass. Later, ActNN (Chen et al., 2021) stores activation tensors in 2-bit precision for training CNNs, greatly reducing activation memory usage by $\sim 12\times$. Mesa (Pan et al., 2021)

uses a customized 8-bit activation quantization strategy for training transformers. AC-GC (Evans & Aamodt, 2021) established a direct relationship between quantization error and training convergence by automatically selecting the compression ratios. GACT (Liu et al., 2022) introduced an adaptive compression strategy for general network architectures, which utilizes the empirical variance of the gradients to estimate the sensitivity of quantized activation tensors. ALAM (Woo et al., 2023) quantizes the group mean estimator and calculates the sensitivity by the empirical variance of the gradients' norm to allocate adaptive compression bits. When applied to transformers, these ACT methods generally reduce more activation memory usage than gradient checkpointing. However, frequent quantization and dequantization in training adversely affect the training throughput of transformers (Wang et al., 2023).

Our method avoids quantization and dequantization during training, and thus keeps the training throughput.

2.3. Parameter-Efficient Fine-Tuning

Parameter-Efficient Fine-Tuning (PEFT) is widely used for transformers due to little memory usage in storing the gradients of trainable parameters or the optimizer states, *e.g.*, AdamW (Loshchilov & Hutter, 2017). Adapter (Houlsby et al., 2019) inserts a two-layer MLP with residual connection after each FFN block. Later, BitFit (Zaken et al., 2021) only fine-tunes the bias and freezes other parameters in the transformers. Prompt Tuning is also studied in (Lester et al., 2021; Li & Liang, 2021; Liu et al., 2021a; Jia et al., 2022) to prepend extra learnable prompt tokens in self-attention. Recently, LoRA (Hu et al., 2022) and its variants (Zhang et al., 2023b;b; Jie & Deng, 2023; Zhang et al., 2023a; Kopiczko et al., 2023) are widely used for scalable fine-tuning power with no extra inference overhead. These methods mainly use low-rank matrices to fine-tune linear layers. By freezing "LoRA-A" parameters, the variant LoRA-FA (Zhang et al., 2023a) can eliminate most of the activation memory costs from linear layers in fine-tuning.

Though using few trainable parameters, these PEFT methods still consume the same order of magnitude of activation memory usage as those used in full fine-tuning. An exception LST (Sung et al., 2022) uses a ladder side model to avoid backward passes through the pretrained modules. However, it performs inferior to LoRA and brings extra memory overhead and latency in the inference stage.

Unlike LoRAs, our work aims to reduce the activation memory usage from non-linear layers in transformers.

2.4. Activation Approximation

The work of AAL (Woo & Jeon, 2022) introduced auxiliary activation to participate the backward pass instead of the original input activation in linear layers. The auxiliary activation is typically the activation from the previous block or the sign of the original activation. The work of (Jiang et al., 2022) introduced an asymmetric sparsifying strategy to obtain sparse activation features for back-propagation, while keeping dense forward activation features. These two works both showed the compatibility with Mesa (Pan et al., 2021) in their papers. Since our method is functionally similar to Mesa, they are also compatible with our method.

3. Preliminary

3.1. Fine-Tuning

Denote $\boldsymbol{x} \in \mathbb{R}^{p_0}$ as the network input vector under the data distribution $\mathcal{D}$, *i.e.*, $\boldsymbol{x} \sim \mathcal{D}$. For an L-layer neural network $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})$, the output feature vector of i-th hidden layer $\boldsymbol{h}^i$ is denoted as $\boldsymbol{z}^i = \boldsymbol{h}^i(\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i) = \boldsymbol{h}^i_{\boldsymbol{\theta}}(\boldsymbol{z}^{i-1}) \in \mathbb{R}^{p_i}$, where $\boldsymbol{z}^0 = \boldsymbol{x}$, $\boldsymbol{\theta} = [{\boldsymbol{\theta}^1}^\top, ..., {\boldsymbol{\theta}^L}^\top]^\top \in \mathbb{R}^M$ and $\boldsymbol{\theta}^i$ is the straightened vector of network parameters of the i-th hidden layer $\boldsymbol{h}^i$. The network can be formulated as

$$\boldsymbol{z}^L = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{h}^L_{\boldsymbol{\theta}} \circ \boldsymbol{h}^{L-1}_{\boldsymbol{\theta}} \circ ... \circ \boldsymbol{h}^1_{\boldsymbol{\theta}}(\boldsymbol{x}), \tag{1}$$

where "$\circ$" denotes layer composition. For simplicity, we define the set of all feature vectors by $\boldsymbol{z} = [{\boldsymbol{z}^1}^\top, ..., {\boldsymbol{z}^L}^\top]^\top$ and express the backward pass of network training as:

$$\boldsymbol{g} \triangleq \boldsymbol{g}(\ell(\boldsymbol{z}^L), \boldsymbol{z}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{z}^L), \tag{2}$$

where $\ell(\boldsymbol{z}^L) = \ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}))$ is the loss function, $\boldsymbol{g}$ is a composite function of the derivatives of ℓ and $\boldsymbol{h}$, *i.e.*, $\mathrm{d}\ell$ and $\{\mathrm{d}\boldsymbol{h}^i\}_{i=1}^L$, respectively. Then the parameter update at t-th iteration in regular BP can be expressed as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \boldsymbol{g}_t = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \ell(\boldsymbol{z}^L_t). \tag{3}$$

The main characteristic of fine-tuning is that the model parameters are initialized as the pretrained weights, *i.e.*, $\boldsymbol{\theta}_0 = \boldsymbol{\theta}_{\text{pretrained}}$. Different from the "training from scratch", the initial model $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_0)$ to be fine-tuned usually already has potential capability on the downstream tasks.

3.2. Activation Memory Usage in Fine-Tuning

In general, all intermediate feature vectors $\{\boldsymbol{z}^i\}_{i=1}^L$ may participate in calculating gradients in the backward pass (2). However, according to the specific layers in the network, we do not need to store all $\{\boldsymbol{z}^i\}_{i=1}^L$ into activation memory in practice. For example, freezing partial parameters is a widely used fine-tuning technique by PEFT methods. A frozen linear layer can be expressed as:

$$\boldsymbol{z}^i = \boldsymbol{h}^i(\boldsymbol{z}^{i-1}) = \boldsymbol{W}_{\text{frozen}} \boldsymbol{z}^{i-1} + \boldsymbol{b}_{\text{frozen}}, \tag{4}$$

where the frozen weight and bias need no gradient, avoiding storing the input feature $\boldsymbol{z}^{i-1}$ into activation memory.

As a representative PEFT method, LoRA (Hu et al., 2022) is briefly analyzed here and we express its adapting layer as

$$\boldsymbol{z}^i = \boldsymbol{h}^i(\boldsymbol{z}^{i-1}) = \boldsymbol{W}_{\text{frozen}}\boldsymbol{z}^{i-1} + \boldsymbol{B}\boldsymbol{A}\boldsymbol{z}^{i-1} + \boldsymbol{b}_{\text{frozen}}, \quad (5)$$

where $\boldsymbol{A} \in \mathbb{R}^{r\times p_{i-1}}$ and $\boldsymbol{B} \in \mathbb{R}^{p_i\times r}$ are trainable parameters. The stored features in activation memory are $\boldsymbol{z}^{i-1} \in \mathbb{R}^{p_{i-1}}$ and $\boldsymbol{A}\boldsymbol{z}^{i-1} \in \mathbb{R}^r$. Since $r \ll p_{in}$, the activation memory usage in a LoRA adapting layer is only slightly larger than that storing $\boldsymbol{z}^{i-1}$ in the linear layer. Besides, LoRA-FA (Zhang et al., 2023a) further freezes the projection matrix $\boldsymbol{A}$ in the LoRA adapting layers (5), and only stores the r-dimensional $\boldsymbol{A}\boldsymbol{z}^{i-1}$ in activation memory.

Although freezing techniques can reduce the activation memory usage in linear layers (including LoRA adapting layers), the activation memory overhead in non-linear layers is still expensive. Among these non-linear layers, activation function and layer normalization bring a large part of activation memory usage. In Figure 2, we illustrate the memory usage ratios of different modules in ViT (Dosovitskiy et al., 2021) and LLaMA (Touvron et al., 2023). One can see that in ViT both GELU and LayerNorm occupy 21.05% of the total activation memory usage, while in LLaMA 12.39% and 18.35% memory usage are from SiLU and RMSNorm, respectively (please refer to Appendix B for more details).

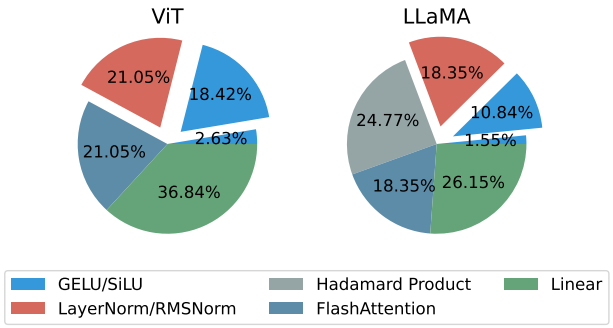

Figure 2. **Composition of activation memory usage in ViT and LLaMA**. For LLaMA, we use LLaMA-13B as an example. Our method is feasible to reduce the activation memory usage of GELU/SiLU and LayerNorm/RMSNorm (the split parts).

4. Approximate Backpropagation

A large model like LLaMA exhibits strong representation capability with its pretrained weights, which are usually more crucial than the fine-tuning itself to its performance on downstream tasks. Therefore, it is reasonable to fine-tune the large model with its architecture the same as the original design (see Appendix C for our empirical investigation).

To provide flexible fine-tuning scheme, in this section, we show the possibility of substituting the backward pass while remaining the forward pass of the pretrained model. In Section 4.1, we present our Approximate Backpropagation (Approx-BP) theory to demonstrate the theoretical feasibility of decoupling the forward and backward passes. In Section 4.2, under the guidance of our Approx-BP theory, we derive ReGELU2 and ReSiLU2 as memory-efficient alternatives of GELU and SiLU, respectively in transformers.

4.1. Approx-BP Theory

We introduce an approximate network $\widetilde{\boldsymbol{f}}$ that shares the same parameters $\boldsymbol{\theta}$ with $\boldsymbol{f}$ in Eqn. (1), *i.e.*,

$$\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}) = \widetilde{\boldsymbol{h}}^L_{\boldsymbol{\theta}} \circ \widetilde{\boldsymbol{h}}^{L-1}_{\boldsymbol{\theta}} \circ ... \circ \widetilde{\boldsymbol{h}}^1_{\boldsymbol{\theta}}(\boldsymbol{x}). \quad (6)$$

The loss function of $\widetilde{\boldsymbol{f}}$ is similarly denoted as $\ell(\widetilde{\boldsymbol{z}}^L) = \ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}))$, and its backward pass is denoted as

$$\widetilde{\boldsymbol{g}} \triangleq \widetilde{\boldsymbol{g}}(\ell(\widetilde{\boldsymbol{z}}^L), \widetilde{\boldsymbol{z}}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{z}}^L). \quad (7)$$

Here, the definitions of $\widetilde{\boldsymbol{z}}$ and $\widetilde{\boldsymbol{g}}$ are the counterparts of $\boldsymbol{z}$ and $\boldsymbol{g}$ in Section 3.1, respectively.

In order to approximate the backward pass of network training in Eqn. (2), we formulate our Approx-BP as

$$\widehat{\boldsymbol{g}} \triangleq \widetilde{\boldsymbol{g}}(\ell(\boldsymbol{z}^L), \boldsymbol{z}, \boldsymbol{\theta}) \approx \boldsymbol{g}(\ell(\boldsymbol{z}^L), \boldsymbol{z}, \boldsymbol{\theta}). \quad (8)$$

Then we replace the gradient update in regular BP (3) by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\widehat{\boldsymbol{g}}_t. \quad (9)$$

By decoupling the forward and backward passes, our Approx-BP is feasible to flexibly fine-tune large models.

By Triangle Inequality, we can derive an insightful property about our Approx-BP as follows:

$$\begin{aligned}\|\widehat{\boldsymbol{g}} - \boldsymbol{g}\| \le& \|\widehat{\boldsymbol{g}} - \widetilde{\boldsymbol{g}}\| + \|\widetilde{\boldsymbol{g}} - \boldsymbol{g}\| \\ =& \|\widetilde{\boldsymbol{g}}(\ell(\boldsymbol{z}^L), \boldsymbol{z}, \boldsymbol{\theta}) - \widetilde{\boldsymbol{g}}(\ell(\widetilde{\boldsymbol{z}}^L), \widetilde{\boldsymbol{z}}, \boldsymbol{\theta})\| + \\ & \|\nabla_{\boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{z}}^L) - \nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{z}^L)\|.\end{aligned} \quad (10)$$

The inequality (10) indicates that approximate BP $\widehat{\boldsymbol{g}}$ and the regular BP $\boldsymbol{g}$ differs in the intermediate outputs of forward pass $\|\boldsymbol{z} - \widetilde{\boldsymbol{z}}\|$, if functions $\widetilde{\boldsymbol{g}}$ and ℓ are in proper continuity. This observation motivates us to design proper alternatives to replace the derivatives of (non-linear) modules in a neural network, as long as their primitive functions are close enough in the functional space. We describe the degree of approximation in our Approx-BP by the following theorem.

Theorem 4.1. *Under the definitions in Section 4.1, assume that:*
A1. *$\widetilde{\boldsymbol{g}}(\ell(\boldsymbol{z}^L), \boldsymbol{z}, \boldsymbol{\theta})$ is uniformly Lipschitz continuous w.r.t. $\ell(\boldsymbol{z}^L)$ and $\boldsymbol{z}$.*
A2. *$\ell(\boldsymbol{z}^L)$ is Lipschitz continuous. $\boldsymbol{h}^i(\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i)$ is uniformly Lipschitz continuous w.r.t. $\boldsymbol{z}^{i-1}$ for $i = 2, ..., L$.*
A3. *$\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}))$ and $\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}))$ are twice differentiable w.r.t. $\boldsymbol{\theta}$ with uniformly bounded induced norm of their Hessian matrices. Then, $\exists\ \alpha > 0, \forall\ \boldsymbol{x}, \boldsymbol{\theta}$, we have*

$$\begin{aligned}&\|\widehat{\boldsymbol{g}}-\boldsymbol{g}\|_2\\&\leq\alpha\Big(\sum_{i=1}^{L}\sup_{\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i}\|\boldsymbol{h}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)-\widetilde{\boldsymbol{h}}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)\|_2+\\&\sqrt{\sum_{i=1}^{L}\sup_{\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i}\|\boldsymbol{h}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)-\widetilde{\boldsymbol{h}}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)\|_2}\Big).\end{aligned}\tag{11}$$

Although the networks containing ReLUs (Nair & Hinton, 2010) do not strictly satisfy the assumptions in Theorem 4.1, the violations only happen in a zero measure set. In the practical training, we can safely conceive a smoothing curve at the neighborhood of zero point in ReLU. Next, we demonstrate the convergence of our Approx-BP theory by another theorem described as follows.

Theorem 4.2. *Suppose data $\boldsymbol{x}$ follows the distribution $\mathcal{D}$. Denote T as the total iteration number. Assume that:*
A1. *$\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}))$ is continuously differentiable w.r.t. $\boldsymbol{\theta}$, and $\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}))$ is β-Lipschitz continuous w.r.t. $\boldsymbol{\theta}$.*
A2. *$\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}))$ is bounded below by a constant ℓ^*.*
A3. *$\exists\,\sigma>0$, for $\forall\boldsymbol{\theta}$, $\mathbb{E}_{\mathcal{D}}\|\widehat{\boldsymbol{g}}-\boldsymbol{g}\|_2^2<\sigma^2$.*
Then, for all $\eta<\frac{1}{2\beta}$, if we run Approx-BP training defined in (9), we have

$$\begin{aligned}&\min_{t\in\{0,\dots,T-1\}}\mathbb{E}_{\mathcal{D}}\|\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}_t))\|_2^2\\&\leq\frac{4(\mathbb{E}_{\mathcal{D}}\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}_0))-\ell^*)}{\eta T}+6\sigma^2.\end{aligned}\tag{12}$$

From Theorem 4.1 and Theorem 4.2, we conclude that the learning capability of the network $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta})$ with our Approx-BP theory mainly correlates to the functional closeness between the original layers $\boldsymbol{h}$ and the approximate layers $\widetilde{\boldsymbol{h}}$.

The theoretical analysis reveals that our Approx-BP theory can work as a feasible framework to decouple the forward and backward passes, with guaranteed training convergence. In contrast, the regular BP in network training links the two opposite passes in a balanced scale of memory overhead. Instead, our Approx-BP can potentially break the scale balance, and is feasible to reduce the activation memory.

4.2. Approx-BP on Activation Functions

Transformers (Radford et al., 2019; Dosovitskiy et al., 2021; Touvron et al., 2022; 2023) usually use GELU or SiLU (Hendrycks & Gimpel, 2023) as the non-linear activation function in MLP blocks. GELU and SiLU (Hendrycks & Gimpel, 2023) usually boost the network performance against ReLU (Nair & Hinton, 2010) in various vision and language tasks. However, GELU and SiLU need to store the whole 16-bit input tensor for backward pass, while ReLU only needs to store the 1-bit signs of the input tensor elements. Therefore, for consideration of memory efficiency, we propose to combine multiple ReLUs to approximate the regular BP process of GELU and SiLU. Denote h as the activation function of GELU or SiLU, we have

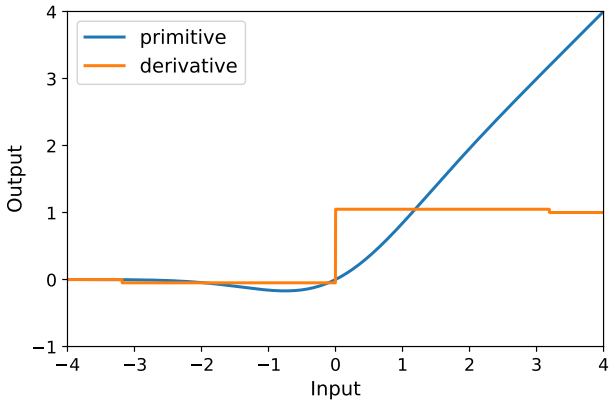

Figure 3. **Plot of our ReGELU2**. The primitive function is still GELU, while the derivative function is a 4-segment step function that need 2 bits of activation memory for derivative calculation.

$$\begin{aligned}h(x)=\mathrm{GELU}(x)&=\frac{x}{2}(1+\mathrm{erf}(\frac{x}{\sqrt{2}}))\\\text{or}\quad h(x)=\mathrm{SiLU}(x)&=\frac{x}{1+e^{-x}}.\end{aligned}$$

We define a combination of multiple ReLUs as

$$\begin{aligned}\widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x)=&\sum_{i=1}^{2^k-2}a_i\mathrm{ReLU}(x-c_i)+\\&(1-\sum_{i=1}^{2^k-2}a_i)\mathrm{ReLU}(x-c_{2^k-1}),\\\text{s.t.}\ \sum_{i=1}^{2^k-2}a_ic_i+&(1-\sum_{i=1}^{2^k-2}a_i)c_{2^k-1}=0,\end{aligned}\tag{13}$$

where the i-th element a_i (or c_i) of $\boldsymbol{a}$ (or $\boldsymbol{c}$) indicates the weight (or bias) of the i-th ReLU in our combined ReLUs. Here we use 2^k-1 ReLUs in $\widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}$ and k is the required bit number of activation memory for derivative calculation.

Proposition 4.3. *The combination function $\widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}$ of multiple ReLUs in Eqn. (13) has the following two properties:*

1. *It has the same limiting behavior with the the activation function $h(x)$, i.e., $\lim_{x\to\infty}h(x)-\widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x)=0$.*

2. *Its derivative is a 2^k-segment step function that need k bits of activation memory for derivative calculation.*

Here, we set $k=2$ to reduce the activation memory usage in activation functions. According to our Approx-BP theory, we should set the parameters in (13), so that $\widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x)$ could be close to $h(x)$ in the function space. To implicitly fulfill the constraint in (13) and put uniform importance to the

define domain, we solve the following feasible problem:

$$\min_{\boldsymbol{a},\boldsymbol{c}} \int_{-\infty}^{\infty} (h(x) - \widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x))^2 \mathrm{d}x. \tag{14}$$

We use the simulated annealing algorithm (Kirkpatrick et al., 1983) (more details in Appendix E) to find a quasi-optimal weight $\boldsymbol{a}^*$ and bias $\boldsymbol{c}^*$. That is, for GELU we have

$$\boldsymbol{a}^*_{gelu} = [-0.04922, 1.098]^\top,$$
$$\boldsymbol{c}^*_{gelu} = [-3.186, -0.001179, 3.191]^\top.$$

And for SiLU we have

$$\boldsymbol{a}^*_{silu} = [-0.04060, 1.081]^\top,$$
$$\boldsymbol{c}^*_{silu} = [-6.305, -0.0008685, 6.326]^\top.$$

We denote the combination of GELU and $\mathrm{d}\widetilde{h}_{\boldsymbol{a}^*_{gelu},\boldsymbol{c}^*_{gelu}}$ (or SiLU and $\mathrm{d}\widetilde{h}_{\boldsymbol{a}^*_{silu},\boldsymbol{c}^*_{silu}}$) by ReGELU2 (or ReSiLU2). Since ReGELU2 (or ReSiLU2) keeps the same primitive function as GELU (or SiLU), the initialization of the fine-tuning model is the exact pretrained model with GELU (or SiLU) activation function. The main advantage of ReGELU2 and ReSiLU2 over GELU and SiLU, respectively, is that ReGELU2 and ReSiLU2 only need to store 2-bit activation for backward pass. Our ReGELU2 and ReSiLU2 do not degrade the training efficiency, since they do not need extra computation for data range estimation (Pan et al., 2021). In addition, while setting a larger k in (13) is also feasible for solving (14) using SGD, this will result in more memory and computational overhead. Since ReGELU2 and ReSiLU2 achieve comparable performance to GELU and SiLU in Section 6, we recommend $k = 2$ to be a universal choice.

5. Memory-Sharing Backpropagation

An insight on regular BP (2) is that there exists redundancy when we store all $\{\boldsymbol{z}^i\}_{i=1}^L$ into activation memory. To show this, we give a more detailed analysis on backward pass at the i-th layer $\boldsymbol{z}^i = \boldsymbol{h}^i(\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i)$. In general, the purpose of backward pass at this layer is to calculate the gradient of the feature input $\frac{\partial \ell}{\partial \boldsymbol{z}^{i-1}}$ and the gradient of the parameter input $\frac{\partial \ell}{\partial \boldsymbol{\theta}^i}$ from the gradient of the feature output $\frac{\partial \ell}{\partial \boldsymbol{z}^i}$. These calculations can be expressed in a general form as

$$\begin{aligned} \frac{\partial \ell}{\partial \boldsymbol{\theta}^i} &= \frac{\partial \boldsymbol{h}^i(\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i)}{\partial \boldsymbol{\theta}^i} \frac{\partial \ell}{\partial \boldsymbol{z}^i}, \\ \frac{\partial \ell}{\partial \boldsymbol{z}^{i-1}} &= \frac{\partial \boldsymbol{h}^i(\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i)}{\partial \boldsymbol{z}^{i-1}} \frac{\partial \ell}{\partial \boldsymbol{z}^i}, \end{aligned} \tag{15}$$

where $\frac{\partial \boldsymbol{h}^i(\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i)}{\partial \boldsymbol{\theta}^i}$ and $\frac{\partial \boldsymbol{h}^i(\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i)}{\partial \boldsymbol{z}^{i-1}}$ are the Jacobian matrices of $\boldsymbol{h}^i$ *w.r.t.* $\boldsymbol{\theta}^i$ and $\boldsymbol{z}^{i-1}$, respectively. The reason for storing $\boldsymbol{z}^{i-1}$ into activation memory is that $\frac{\partial \boldsymbol{h}^i(\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i)}{\partial \boldsymbol{\theta}^i}$ and $\frac{\partial \boldsymbol{h}^i(\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i)}{\partial \boldsymbol{z}^{i-1}}$ involve the term $\boldsymbol{z}^{i-1}$. However, this involvement is not always necessary. In this section, we discuss about the situation in which the Jacobian matrices do not involve the term $\boldsymbol{z}^{i-1}$, and show how to use this property to achieve memory-sharing backpropagation (MS-BP) for avoiding the activation memory redundancy.

Algorithm 1 Memory-Sharing Layer Normalization

Denote ℓ as the loss function.
Input: $\boldsymbol{z}^{i-1} \in \mathbb{R}^{p_{i-1}}$
Forward Pass:
$\sigma = \sqrt{p_{i-1}^{-1} \boldsymbol{z}^{i-1\top} \boldsymbol{H}^\top \boldsymbol{H} \boldsymbol{z}^{i-1} + \varepsilon}$
$\boldsymbol{z}^i = \sigma^{-1} \boldsymbol{H} \boldsymbol{z}^{i-1}$
Save $\boldsymbol{z}^i$, σ *for backward pass*
Return Output: $\boldsymbol{z}^i$
Backward Pass:
Receive gradient : $\frac{\partial \ell}{\partial \boldsymbol{z}^i}$
$\frac{\partial \ell}{\partial \boldsymbol{z}^{i-1}} = \sigma^{-1} \boldsymbol{H}^\top (\mathbb{I} - p_{i-1}^{-1} \boldsymbol{z}^i \boldsymbol{z}^{i\top}) \frac{\partial \ell}{\partial \boldsymbol{z}^i}$
Return Gradient: $\frac{\partial \ell}{\partial \boldsymbol{z}^{i-1}}$

5.1. Sufficient Condition of MS-BP

We begin with a proposition about when the layer $\boldsymbol{h}^{i-1}$ can share the activation memory with the following layer $\boldsymbol{h}^i$.

Proposition 5.1. *If the layer $\boldsymbol{h}^i$ satisfies the following conditions, we can reduce the activation memory in $\boldsymbol{h}^i$ by sharing its activation memory with $\boldsymbol{h}^{i+1}$:*

1. *$\boldsymbol{h}^i$ does not involve parameters $\boldsymbol{\theta}^i$, i.e., $\boldsymbol{z}^i = \boldsymbol{h}^i(\boldsymbol{z}^{i-1})$.*
2. *The Jacobian matrix $\frac{\partial \boldsymbol{h}^i(\boldsymbol{z}^{i-1})}{\partial \boldsymbol{z}^{i-1}}$ can be reformulated as $\boldsymbol{J}(\boldsymbol{z}^i, \boldsymbol{\phi}^i)$, where $\boldsymbol{\phi}^i \in \mathbb{R}^{q_i}$ is an auxiliary variable with dimension $q_i \ll p_{i-1}$.*
3. *The backward pass at $\boldsymbol{h}^{i+1}$ involves $\boldsymbol{z}^i$.*

Under the conditions in Proposition 5.1, the calculation of $\frac{\partial \ell}{\partial \boldsymbol{\theta}^i}$ is not required any more, and the calculation of $\frac{\partial \ell}{\partial \boldsymbol{z}^{i-1}}$ no longer needs $\boldsymbol{z}^{i-1}$. Therefore, the intermediate feature $\boldsymbol{z}^{i-1}$ can be removed from the activation memory, and both $\boldsymbol{h}^i$ and $\boldsymbol{h}^{i+1}$ utilize $\boldsymbol{z}^i$ for gradient calculation. Then the activation memory usage in $\boldsymbol{h}^i$ and $\boldsymbol{h}^{i+1}$ can be reduced from $\{\boldsymbol{z}^{i-1}, \boldsymbol{z}^i\}$ to $\{\boldsymbol{\phi}^i, \boldsymbol{z}^i\}$. The first two conditions in Proposition 5.1 are loose enough to cover simple element-wise activation functions and normalization layers. But the third condition is not often met in fine-tuning networks when $\boldsymbol{h}^{i+1}$ is a frozen linear layer. Unfortunately, the widely used SiLU does not satisfy the second condition (please refer to Appendix F for details). Thus, we mainly consider how to apply MS-BP to the layer normalization in Section 5.2.

5.2. Memory-Sharing Normalization

In this section, we describe the detailed technique for applying our MS-BP to LayerNorm (Ba et al., 2016) and its variant RMSNorm (Zhang & Sennrich, 2019).

The forward pass at the LayerNorm or RMSNorm and the following linear layer can be expressed as:

$$\begin{aligned} \sigma &= \sqrt{p_{i-1}^{-1}\boldsymbol{z}^{i-1\top}\boldsymbol{H}^\top\boldsymbol{H}\boldsymbol{z}^{i-1}+\varepsilon}, \\ \widetilde{\boldsymbol{z}}^{i-1} &= \sigma^{-1}\boldsymbol{H}\boldsymbol{z}^{i-1}, \\ \boldsymbol{z}^{i} &= \mathrm{diag}(\boldsymbol{\alpha})\widetilde{\boldsymbol{z}}^{i-1}+\boldsymbol{\beta}, \\ \boldsymbol{z}^{i+1} &= \boldsymbol{W}\boldsymbol{z}^{i}+\boldsymbol{b}, \end{aligned} \tag{16}$$

where $\boldsymbol{H}$ is a general matrix. For LayerNorm we have $\boldsymbol{H} = \mathbb{I} - p_{i-1}^{-1}\mathbb{1}\mathbb{1}^\top$, while for RMSNorm we have $\boldsymbol{H} = \mathbb{I}$, $\boldsymbol{\beta} = \boldsymbol{0}$. Here, $\mathbb{I}$ is the identity matrix and $\mathbb{1}$ is a vector of all ones. ε is a small positive scalar of 10^{-6} or 10^{-8}. $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the affine weight and bias, respectively, in LayerNorm.

To satisfy the conditions in Proposition 5.1, we merge the affine parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ into the linear layer in (16) as

$$\widetilde{\boldsymbol{W}} = \boldsymbol{W}\mathrm{diag}(\boldsymbol{\alpha}),\ \widetilde{\boldsymbol{b}} = \boldsymbol{W}\boldsymbol{\beta}+\boldsymbol{b}. \tag{17}$$

Then the forward pass is simplified as:

$$\begin{aligned} \sigma &= \sqrt{p_{i-1}^{-1}\boldsymbol{z}^{i-1\top}\boldsymbol{H}^\top\boldsymbol{H}\boldsymbol{z}^{i-1}+\varepsilon}, \\ \boldsymbol{z}^{i} &= \sigma^{-1}\boldsymbol{H}\boldsymbol{z}^{i-1}, \\ \boldsymbol{z}^{i+1} &= \widetilde{\boldsymbol{W}}\boldsymbol{z}^{i}+\widetilde{\boldsymbol{b}}. \end{aligned} \tag{18}$$

Now, we check the conditions in Proposition 5.1. The first condition is met since there is no parameter in layer normalization after merging affine parameters. The third condition is met at least in full tuning and LoRA, where the query and value projections are always adapted. To show the second condition is also met, we demonstrate how to reformulate the Jacobian matrix of the layer normalization in Algorithm 1. By this way, the total activation memory usage of a memory-sharing layer normalization and the following linear layer becomes the memory size of one vector in $\mathbb{R}^{p_{i-1}}$ and one scalar in $\mathbb{R}$. We denote the memory-sharing LayerNorm as MS-LN and the memory-sharing RMSNorm as MS-RMSNorm (please refer to Appendix G for details).

6. Experiments

In this section, we conduct experiments by deploying our ReGELU2, ReSiLU2, MS-LN, and MS-RMSNorm into the representative ViT (Dosovitskiy et al., 2021) for vision tasks, as well as LLaMA (Touvron et al., 2023) and RoBERTa (Liu et al., 2019) for natural language understanding tasks. Specifically, we deploy our ReGELU2 (or ReSiLU2) into ViT, RoBERTa (or LLaMA) to replace the GELU (or SiLU) function. MS-LN (or MS-RMSNorm) is also used to replace LayerNorm (or RMSNorm) with merged weights of pretrained ViT, RoBERTa (or LLaMA). Our method needs no extra operation in practical implementation. We implement compatible CUDA kernels for our ReGELU2, ReSiLU2, MS-LN, and MS-RMSNorm. FlashAttention (Dao et al., 2022) is used in the ViT and LLaMA experiments. More experiments are put in Appendix J.

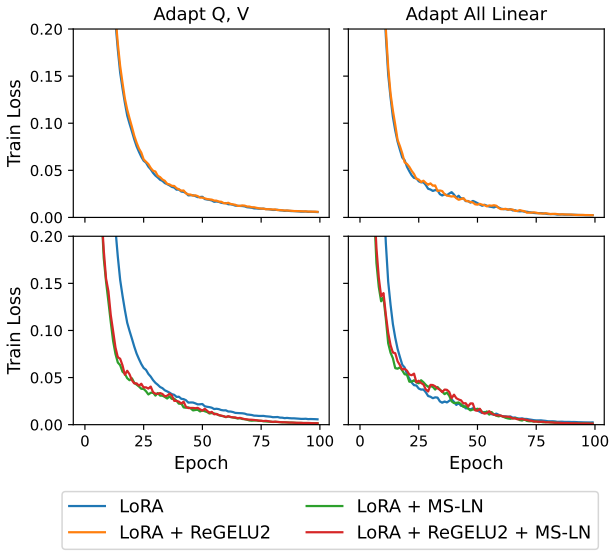

Figure 4. **Convergence of ReGELU2 and MS-LN when using LoRA (rank = 4) on ViT-base (Dosovitskiy et al., 2021).** The training loss is the average over the training loss on CIFAR10/100 (Krizhevsky et al., 2009) and FGVC (Jia et al., 2022).

6.1. Fine-Tuning ViT on Image Classification

Benchmark. Here, we employ the transformer models ViT-base and ViT-large pretrained on ImageNet-22k (Deng et al., 2009; Dosovitskiy et al., 2021) as the backbones, which are fine-tuned on the CIFAR10/100 (Krizhevsky et al., 2009) and FGVC (Jia et al., 2022) datasets. GELU and LayerNorm are the default modules in ViT-base and ViT-large.

Fine-tuning. We implement our method with LoRA (Hu et al., 2022), LoRA-FA (Zhang et al., 2023a), and full fine-tuning (Full-Tuning). For LoRA, we adapt the weights of query and value projection or all linear layers. Since the linear layers in LoRA-FA only store $\boldsymbol{A}\boldsymbol{x}$ instead of $\boldsymbol{x}$ in backward pass, our MS-LN can not reduce the activation memory usage to the following linear layers. Therefore, we only use ReGELU2 for LoRA-FA in our experiments. Please refer to Appendix H for more implementation details.

Comparison methods. We compare our method with Mesa (Pan et al., 2021), a activation quantization method providing 8-bit GELU and LN. We do not evaluate ActNN (Chen et al., 2021) since it is designed for CNNs and not usable to GELU and LN in ViTs. We also do not evaluate GACT (Liu et al., 2022) due to training collapse in our experiments.

Results. In Figure 4, we plot the average loss curves of fine-tuning ViT-base with LoRAs on CIFAR10/100 (Krizhevsky et al., 2009) and FGVC (Jia et al., 2022). We observe that the convergence tendency of our ReGELU2 is almost identical to that of GELU, while the training loss of ViT-base with our MS-LN decreases more rapidly than that without it. This indicates that ReGELU2 preserves the learning capability of GELU while MS-LN accelerates the convergence speed.

Table 1. **Average results on CIFAR10/100 and FGVC by fine-tuning ViT-base**. The best results are highlighted in **bold**.

Method	Activation	Norm	Adapt Q, V Top-1(%)	Adapt Q, V Mem.(MiB)	Adapt Q, V Thr.(images/s)	Adapt All Linear Top-1(%)	Adapt All Linear Mem.(MiB)	Adapt All Linear Thr.(images/s)
LoRA $r=4$	GELU	LN	**90.3**	3827	288	90.7	5128	**207**
	Mesa-GELU	LN	**90.3**	3453(-10%)	245(-15%)	**90.8**	4721(-8%)	186(-10%)
	ReGELU2	LN	**90.3**	**3087**(-19%)	**289**(+0%)	**90.8**	**4380**(-15%)	**207**(+0%)
	GELU	Mesa-LN	90.2	**3249**(-15%)	257(-11%)	90.8	4530(-12%)	189(-9%)
	GELU	MS-LN	**90.7**	3441(-10%)	**288**(+0%)	**91.2**	**4316**(-16%)	**207**(+0%)
	Mesa-GELU	Mesa-LN	90.4	2853(-25%)	226(-22%)	90.8	4209(-18%)	173(-17%)
	ReGELU2	MS-LN	**90.5**	**2717**(-29%)	**290**(+1%)	**91.2**	**3601**(-30%)	**208**(+0%)
LoRA-FA $r=4$	GELU	LN	**90.0**	3386	304	**90.2**	3430	249
	Mesa-GELU	LN	89.9	3012(-11%)	261(-14%)	**90.2**	3021(-12%)	218(-12%)
	Mesa-GELU	Mesa-LN	89.9	**2411**(-29%)	236(-22%)	90.1	**2457**(-28%)	200(-20%)
	ReGELU2	LN	89.8	2597(-23%)	**306**(+1%)	**90.2**	2717(-21%)	**251**(+0%)

Table 2. **Average results on CIFAR10/100 and FGVC by fine-tuning ViT-base and ViT-large**. The best results are highlighted in **bold**.

Method	Activation	Norm	ViT-base Top-1(%)	ViT-base Mem.(GiB)	ViT-base Thr.(images/s)	ViT-large Top-1(%)	ViT-large Mem.(GiB)	ViT-large Thr.(images/s)
Full Tuning	GELU	LN	89.23	5.6	235	90.99	15.7	175
	ReGELU2	LN	**89.31**	4.9(-13%)	232(-1%)	**91.15**	13.7(-13%)	176(1%)
	GELU	MS-LN	88.69	4.9(-14%)	238(+1%)	90.62	13.5(-14%)	182(4%)
	ReGELU2	MS-LN	88.75	**4.1**(-27%)	**241**(+2%)	90.96	**11.5**(-27%)	**183**(4%)

Table 3. **Main results on fine-tuning LLaMA-7B and LLaMA-13B using QLoRA on Alpaca**. "*" indicates that the values are reported in QLoRA paper. The best results are highlighted in **bold**. $1\text{GiB} = 1024\text{MiB} = 1024^3\text{Bytes}$.

Method	Activation	Norm	LLaMA-7B Accuracy(%)	LLaMA-7B Mem.(GiB)	LLaMA-7B Thr.(samples/s)	LLaMA-13B Accuracy(%)	LLaMA-13B Mem.(GiB)	LLaMA-13B Thr.(samples/s)
No Tuning	SiLU	RMSNorm	35.65(35.1*)			45.26(46.9*)		
QLoRA $r=64$ All Linear	SiLU	RMSNorm	**40.75**(39.0*)	20.6	7.9	**46.68**(47.5*)	31.4	5.8
	ReSiLU2	RMSNorm	39.86	19.0(-8%)	7.9(+0%)	46.59	29.0(-8%)	5.7(-2%)
	SiLU	MS-RMSNorm	40.13	18.0(-12%)	8.2(+3%)	46.34	27.5(-12%)	5.8(+0%)
	ReSiLU2	MS-RMSNorm	40.35	**14.6**(-29%)	**8.6**(+9%)	46.54	**22.3**(-29%)	**6.5**(+13%)

Table 4. **Main results on fine-tuning RoBERTa-base using LoRA on GLUE**. The best results are highlighted in **bold**.

Method	Activation	Norm	Tasks CoLA	SST-2	MRPC	STS-B	RTE	Mean Accuracy(%)	Mean Mem.(MiB)	Mean Thr.(samples/s)
LoRA $r=64$ Q, K	GELU	LN	61.08	93.81	86.52	89.18	71.48	80.41	6517	**202**
	ReGELU2	LN	58.03	93.46	87.75	**89.73**	69.31	79.66	5438(-17%)	202(-0%)
	GELU	MS-LN	57.52	94.04	86.52	89.18	**75.45**	80.54	6253(-4%)	196(-3%)
	ReGELU2	MS-LN	**61.60**	**94.27**	**87.99**	89.71	75.09	**81.73**	**5173**(-21%)	198(-2%)

In Table 1 and Table 2, we compare the results of inference accuracy, activation memory usage, and training throughput, on fine-tuning ViT-base by LoRA, LoRA-FA, and Full-Tuning, respectively. We observe that the LoRA with our method (ReGELU2 + MS-LN) reduces the peak GPU memory usage by $\sim$1.1 GiB and $\sim$1.5 GiB when adapting query/value projection and all linear layers, respectively, both occupying $\sim 30\%$ of peak GPU memory usage by vanilla LoRA. Similarly, our method reduces $\sim 27\%$ of the peak GPU memory usage in Full-Tuning. In LoRA-FA fine-tuning, our ReGELU2 reduces the peak GPU memory usage by $\sim 20\%$. Besides, our method (ReGELU2 + MS-LN) does not degrade the training throughput and inference accuracy, while Mesa degrades training throughput clearly.

Since its activation memory behavior is independent of the fine-tuning methods, our ReGELU2 achieves consistent activation memory reduction in all cases of our experiments. Due to frozen FFN modules in LoRA, the memory usage reduction by our MS-LN on adapting query and value projections is less than those on adapting all linear layers in LoRA and full tuning. Here, the third condition of Proposition 5.1 is not satisfied for the LN in the FFN modules.

6.2. Fine-Tuning LLaMA on Language Understanding

Benchmark. We fine-tune LLaMA-7B and LLaMA-13B (Touvron et al., 2023) using Alpaca (Taori et al., 2023) and evaluate the fine-tuned models on 5-shot MMLU (Hendrycks et al., 2020). LLaMA uses SwiGLU (Shazeer, 2020) (containing SiLU in its implementation) for activation and RMSNorm for layer normalization. The training uses model parallel provided in the Transformers package (Wolf et al., 2020) with 2×H800 GPUs. The reported peak memory usage is the max value of those from the 2 GPUs.

Fine-tuning. We deploy our method into QLoRA (Dettmers et al., 2023) to fine-tune LLaMA-7B and LLaMA-13B. QLoRA uses NF4 data type to store the pretrained weights and uses Bfloat16 to store the parameters in LoRA. In QLoRA, all projection weights in linear layers are adapted by LoRA. When applying our MS-RMSNorm to merge the affine parameters, we transpose the weight matrix of the pretrained parameters, to avoid changing the conditional distribution of the block-wise quantization in QLoRA. Please refer to Appendix H for more implementation details.

Results on fine-tuning LLaMA-7B and LLaMA-13B are summarized in Table 3. We observe that fine-tuning LLaMAs by our method achieves comparable MMLU accuracy to the baseline. Our method substantially reduces the peak memory usage on fine-tuning LLaMAs by QLoRA, *i.e.*, $\sim$6.0 GiB on fine-tuning LLaMA-7B and $\sim$9.1 GiB on fine-tuning LLaMA-13B, representing a significant amount of GPU memory savings. The reduction amounts both occupy $\sim 30\%$ of the baseline's peak GPU memory usage. What's more, our method yields an $\sim 10\%$ improvement of training throughput on fine-tuning LLaMA-7B and LLaMA-13B with QLoRA. Fine-tuning LLaMA-7B and LLaMA-13B with our method suffer from slight accuracy drops of 0.40% and 0.14%, respectively. This indicates that our method can be potentially applied to larger transformers.

Note that fine-tuning LLaMAs with both ReSiLU2 and MS-RMSNorm achieves larger memory usage reduction than the sum of reductions by using them separately. This is possibly attributed to the implementation details of QLoRA.

6.3. Fine-Tuning RoBERTa on Language Understanding

Benchmark. We fine-tune the pretrained RoBERTa-base (Liu et al., 2019) on five taskes of GLUE (Wang et al., 2018), *i.e.*, CoLA, SST-2, MRPC, STS-B and RTE. RoBERTa-base uses GELU and LayerNorm. The training uses model parallel provided in the Transformers package (Wolf et al., 2020) with 2×RTX4090 GPUs. The reported usage of peak memory overhead is the sum of those from the 2 GPUs.

Fine-tuning. We implement our method with LoRA to fine-tune the pretrained RoBERTa-base. The data type in this experiment is FP32. Please refer to Appendix H for more implementation details.

Results on fine-tuning RoBERTa-base are summarized in Table 4. Fine-tuning RoBERTa-base with our method achieves comparable accuracy and training throughput to the baseline. Our method reduces the amount of GPU memory usage by $\sim 21\%$. Here, MS-LN gets less reduction of GPU memory usage than ReGELU2, which may be attributed to two reasons. First, we use FP32 in this experiment, so that LayerNorm occupies less proportion of activation memory usage than that in AMP training. Secondly, since LoRA only adapts projection weights in the queries and keys in the attention modules, the third condition of Proposition 5.1 is not satisfied for the LN in the FFN modules.

7. Conclusion

To reduce the activation memory overhead in backpropagation (BP), in this paper, we introduced an Approximate Backpropagation (Approx-BP) theory and a Memory-sharing Backpropagation (MS-BP) strategy. Our Approx-BP theory revealed the feasibility of decoupling the primitive and derivative functions of network layers for training. We derived the ReGELU2 and ReSiLU2 as alternatives of the GELU and SiLU, respectively, used in transformers. We applied our MS-BP strategy into layer normalization (LN), and proposed MS-LN (or MS-RMSNorm) to remove the activation memory redundancy between LN and the following linear layers in regular BP. Experimental results demonstrated that our method reduces up to $\sim 30\%$ of the peak GPU memory usage on fine-tuning transformers, with comparable accuracy and no drop on training throughput.

We believe that our method can be applied to not only fine-tuning stage but also pretraining stage. Even though pretraining exceeds our research scope, we have explored how our method can benefit the pretraining from two aspects. In Appendix J.2, we show that our method can increase the length of training sequence substantially. In Appendix J.2, our method can reduce the communication times in the distributed training by allowing a large batch size, thereby increasing the training throughput significantly.

Acknowledgements

This work is supported in part by National Natural Science Foundation of China (No. 12226007 and 62176068), the Fundamental Research Funds for the Central Universities, and CAAI-Huawei MindSpore Open Fund.

Impact Statement

This paper presents a work whose goal is to advance the field of machine learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here. However, our work has the potential contribution to positively lowering the fine-tuning barrier of large models and promoting their popularity in both research community and industrial applications.

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Banner, R., Hubara, I., Hoffer, E., and Soudry, D. Scalable methods for 8-bit training of neural networks. *Advances in neural information processing systems*, 31, 2018.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer, 2020.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization methods for large-scale machine learning. *SIAM review*, 60(2):223–311, 2018.

Chakrabarti, A. and Moseley, B. Backprop with approximate activations for memory-efficient network training. *Advances in Neural Information Processing Systems*, 32, 2019.

Chen, J., Zheng, L., Yao, Z., Wang, D., Stoica, I., Mahoney, M., and Gonzalez, J. Actnn: Reducing training memory footprint via 2-bit activation compressed training. In *International Conference on Machine Learning*, pp. 1803–1813. PMLR, 2021.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost, 2016.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers, 2019.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and FeiFei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning, 2017.

Evans, R. D. and Aamodt, T. Ac-gc: Lossy activation compression with guaranteed convergence. In *Advances in Neural Information Processing Systems*, 2021.

Everingham, M., Eslami, S. M., Gool, L., Williams, C. K., Winn, J., and Zisserman, A. The pascal visual object classes challenge: A retrospective. *Int. J. Comput. Vision*, 111(1):98–136, jan 2015. ISSN 0920-5691. doi: 10.1007/s11263-014-0733-5. URL https://doi.org/10.1007/s11263-014-0733-5.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus), 2023.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 2019.

Hu, E. J., Shen, Y., Wallis, P., AllenZhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Jia, M., Tang, L., Chen, B.-C., Cardie, C., Belongie, S., Hariharan, B., and Lim, S.-N. Visual prompt tuning. In *European Conference on Computer Vision (ECCV)*, 2022.

Jiang, Z., Chen, X., Huang, X., Du, X., Zhou, D., and Wang, Z. Back razor: Memory-efficient transfer learning by self-sparsified backpropagation. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2022.

Jie, S. and Deng, Z.-H. Fact: Factor-tuning for lightweight adaptation on vision transformer, 2023.

Kirkpatrick, S., Gelatt Jr, C. D., and Vecchi, M. P. Optimization by simulated annealing. *science*, 220(4598): 671–680, 1983.

Kitaev, N., Łukasz Kaiser, and Levskaya, A. Reformer: The efficient transformer, 2020.

Kopiczko, D. J., Blankevoort, T., and Asano, Y. M. Vera: Vector-based random matrix adaptation, 2023.

Korthikanti, V. A., Casper, J., Lym, S., McAfee, L., Andersch, M., Shoeybi, M., and Catanzaro, B. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5, 2023.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Lester, B., AlRfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

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

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

Liu, X., Ji, K., Fu, Y., Tam, W. L., Du, Z., Yang, Z., and Tang, J. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*, 2021a.

Liu, X., Zheng, L., Wang, D., Cen, Y., Chen, W., Han, X., Chen, J., Liu, Z., Tang, J., Gonzalez, J., et al. Gact: Activation compressed training for generic network architectures. In *International Conference on Machine Learning*, pp. 14139–14152. PMLR, 2022.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach, 2019.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021b.

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

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Pan, Z., Chen, P., He, H., Liu, J., Cai, J., and Zhuang, B. Mesa: A memory-saving training framework for transformers. *arXiv preprint arXiv:2111.11124*, 2021.

Piessens, R., de Doncker-Kapenga, E., and Ueberhuber, C. Quadpack. a subroutine package for automatic integration. *Springer Series in Computational Mathematics*, 1983.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. ISBN 9781728199986.

Rajbhandari, S., Ruwase, O., Rasley, J., Smith, S., and He, Y. Zero-infinity: breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384421. doi: 10.1145/3458817.3476205. URL https://doi.org/10.1145/3458817.3476205.

Rajpurkar, P., Jia, R., and Liang, P. Know what you don't know: Unanswerable questions for SQuAD. In Gurevych, I. and Miyao, Y. (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2124. URL https://aclanthology.org/P18-2124.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions, 2017.

Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3406703. URL https://doi.org/10.1145/3394486.3406703.

Robbins, H. and Monro, S. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Shazeer, N. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Sung, Y.-L., Cho, J., and Bansal, M. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *Advances in Neural Information Processing Systems*, 35: 12991–13005, 2022.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Stanford alpaca: An instruction-following llama model, 2023.

Tolstikhin, I., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., Lucic, M., and Dosovitskiy, A. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021.

Touvron, H., Cord, M., and Jégou, H. Deit iii: Revenge of the vit. In *European Conference on Computer Vision*, pp. 516–533. Springer, 2022.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Wang, G., Liu, Z., Jiang, Z., Liu, N., Zou, N., and Hu, X. Division: memory efficient training via dual activation precision. In *International Conference on Machine Learning*, pp. 36036–36057. PMLR, 2023.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

Woo, S. and Jeon, D. Learning with auxiliary activation for memory-efficient training. In *The Eleventh International Conference on Learning Representations*, 2022.

Woo, S., Lee, S., and Jeon, D. Alam: Averaged low-precision activation for memory-efficient training of transformer models. In *The Twelfth International Conference on Learning Representations*, 2023.

Yu, W., Luo, M., Zhou, P., Si, C., Zhou, Y., Wang, X., Feng, J., and Yan, S. Metaformer is actually what you need for vision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10819–10829, 2022.

Zaken, E. B., Ravfogel, S., and Goldberg, Y. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*, 2021.

Zhang, B. and Sennrich, R. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Zhang, L., Zhang, L., Shi, S., Chu, X., and Li, B. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning, 2023a.

Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W., and Zhao, T. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2023b.

A. Qualitative Comparison of Related Works

In Table 5, we provide qualitative comparison of different methods on three aspects, *i.e.*, applicable to non-linear layers ("Non-Linear"), keep training throughput ("Keep Throughput"), and applicable beyond LoRAs ("Beyond LoRA"). Our method can reduce the activation memory usage in non-linear layers, which can not be achieved by parameter freezing techniques (Hu et al., 2022; Jia et al., 2022) or LoRA-FA (Zhang et al., 2023a). One key advantage of our method over gradient checkpointing (Chen et al., 2016) and ACT methods (Pan et al., 2021; Liu et al., 2022) is that our method does not degrade the training efficiency.

Table 5. **Comparison of different methods on activation memory reduction.** "Freeze": freezing some parameters in fine-tuning. "CKPT": Gradient Checkpointing (Chen et al., 2016). "ACT": Activation Compression Training (Pan et al., 2021; Liu et al., 2022).

Method	Non-Linear	Keep Throughput	Beyond LoRA
Freeze	✗	✓	✓
CKPT (Chen et al., 2016)	✓	✗	✓
ACT (Pan et al., 2021; Liu et al., 2022)	✓	✗	✓
LoRA-FA (Zhang et al., 2023a)	✗	✓	✗
Our Method	✓	✓	✓

B. Analyses on activation memory allocation in each block of ViT and LLaMA

We present detailed analysis of the activation memory allocation for each operator within the transformer blocks of ViT (Dosovitskiy et al., 2021) and LLaMA (Touvron et al., 2023). For ViT, refer to Figure 5; for LLaMA, refer to Figure 6.

C. Possibility of Substituting the Forward Pass of Activation Funcition

We also investigate the possibility of changing the whole activation function including forward pass. Nevertheless, empirical results show that changing forward pass of activation function severely degrades the fine-tuning performance. We attribute this phenomenon to the criticality of model initialization. Specifically, replacing SiLU by $\widetilde{h}_{\boldsymbol{a}^*_{silu},\boldsymbol{c}^*_{silu}}(x)$ in (13), the no-tuning MMLU accuracy of LLaMA-7B degrades from 35.62% to 23.44% and the no-tuning MMLU accuracy of LLaMA-13B degrades from 45.26% to 23.51%. Hence, we retain the forward pass in activation function.

D. Proof of theorems

Proof of Theorem 4.1. According to the definitions in Section 4.1, we have the following decomposition:

$$\begin{aligned}\|\widehat{\boldsymbol{g}}-\boldsymbol{g}\|_2 &= \|\widehat{\boldsymbol{g}}-\widetilde{\boldsymbol{g}}+\widetilde{\boldsymbol{g}}-\boldsymbol{g}\|_2 \le \|\widehat{\boldsymbol{g}}-\widetilde{\boldsymbol{g}}\|_2+\|\widetilde{\boldsymbol{g}}-\boldsymbol{g}\|_2 \\ &= \|\widetilde{\boldsymbol{g}}(\ell(\boldsymbol{z}^L),\boldsymbol{z},\boldsymbol{\theta})-\widetilde{\boldsymbol{g}}(\ell(\widetilde{\boldsymbol{z}}^L),\widetilde{\boldsymbol{z}},\boldsymbol{\theta})\|_2+\|\frac{\partial}{\partial\boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}))-\frac{\partial}{\partial\boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}))\|_2.\end{aligned} \tag{19}$$

By A1, $\exists a_1>0, \forall\boldsymbol{\theta}$, we have

$$\|\widetilde{\boldsymbol{g}}(\ell(\boldsymbol{z}^L),\boldsymbol{z},\boldsymbol{\theta})-\widetilde{\boldsymbol{g}}(\ell(\widetilde{\boldsymbol{z}}^L),\widetilde{\boldsymbol{z}},\boldsymbol{\theta})\|_2 \le a_1(\|\ell(\boldsymbol{z}^L)-\ell(\widetilde{\boldsymbol{z}}^L)\|_2+\|\boldsymbol{z}-\widetilde{\boldsymbol{z}}\|_2). \tag{20}$$

By A2, $\exists\, a_2>0$, such that

$$\|\ell(\boldsymbol{z}^L)-\ell(\widetilde{\boldsymbol{z}}^L)\|_2 \le a_2\|\boldsymbol{z}^L-\widetilde{\boldsymbol{z}}^L\|_2. \tag{21}$$

Combining the above inequalities, we have

$$\begin{aligned}\|\widetilde{\boldsymbol{g}}(\ell(\boldsymbol{z}^L),\boldsymbol{z},\boldsymbol{\theta})-\widetilde{\boldsymbol{g}}(\ell(\widetilde{\boldsymbol{z}}^L),\widetilde{\boldsymbol{z}},\boldsymbol{\theta})\|_2 \le& a_1a_2\|\boldsymbol{z}^L-\widetilde{\boldsymbol{z}}^L\|_2+a_1\|\boldsymbol{z}-\widetilde{\boldsymbol{z}}\|_2 \\ \le&(1+a_1a_2)\|\boldsymbol{z}^L-\widetilde{\boldsymbol{z}}^L\|_2+a_1\sum_{i=1}^{L-1}\|\boldsymbol{z}^i-\widetilde{\boldsymbol{z}}^i\|_2.\end{aligned} \tag{22}$$

By A3, $\exists M_1>0, \exists M_2>0, \forall\boldsymbol{x}, \forall\boldsymbol{\theta}\in\mathbb{R}^M, \forall\boldsymbol{q}\in\mathbb{R}^M$, we have

$$\|\frac{\partial^2}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}))\boldsymbol{q}\|_2 \le M_1\|\boldsymbol{q}\|_2 \quad\text{and}\quad \|\frac{\partial^2}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}))\boldsymbol{q}\|_2 \le M_1\|\boldsymbol{q}\|_2. \tag{23}$$

ViT Block	Save for Backward: if parameters trainable		Save for Backward: if parameters frozen			Our Method	Save for Backward: if parameters trainable	
$\widetilde{X}_{ln1} = \mathrm{LN}(X_{ln1})$	$X_{in1}, \mu_{in1}, (\sigma^2_{in1}+\varepsilon)^{-\frac{1}{2}}$	+2	$X_{in1}, \mu_{in1}, (\sigma^2_{in1}+\varepsilon)^{-\frac{1}{2}}$	+2	MS-BP	$X_{ln1} = \text{MS-LN}(X_{in1})$	$X_{ln1}, (\sigma^2_{in1}+\varepsilon)^{-\frac{1}{2}}$	+1
$X_{ln1} = \widetilde{X}_{in1}\mathrm{diag}(\alpha_1) + \beta_1$								
for $y = q, k, v$ $y = X_{ln1}W_y^\top + b_y$	X_{ln1}	+1	\	+0		for $y = q, k, v$ $y = X_{ln1}\widetilde{W}_y^\top + \widetilde{b}_y$		
for $y = q, k, v$ $y_{\mathrm{MH}} = \mathrm{reshape}(y, [b,n,c] \to [b,h,n,c/h])$	\	+0	\	+0		for $y = q, k, v$ $y_{\mathrm{MH}} = \mathrm{reshape}(y, [b,n,c] \to [b,h,n,c/h])$	\	+0
$o_{\mathrm{MH}} = \mathrm{flashattn}(q_{\mathrm{MH}}, k_{\mathrm{MH}}, v_{\mathrm{MH}})$	$q_{\mathrm{MH}}, k_{\mathrm{MH}}, v_{\mathrm{MH}}, o_{\mathrm{MH}}, m, l$	+4	$q_{\mathrm{MH}}, k_{\mathrm{MH}}, v_{\mathrm{MH}}, o_{\mathrm{MH}}, m, l$	+4		$o_{\mathrm{MH}} = \mathrm{flashattn}(q_{\mathrm{MH}}, k_{\mathrm{MH}}, v_{\mathrm{MH}})$	$q_{\mathrm{MH}}, k_{\mathrm{MH}}, v_{\mathrm{MH}}, o_{\mathrm{MH}}, m, l$	+4
$X_{attn} = \mathrm{reshape}(o_{\mathrm{MH}}, [b,h,n,c/h] \to [b,n,c])$	\	+0	\	+0		$X_{attn} = \mathrm{reshape}(o_{\mathrm{MH}}, [b,h,n,c/h] \to [b,n,c])$	\	+0
$X_{proj} = X_{attn}W_{proj}^\top + b_{proj}$	X_{attn}	+1	\	+0		$X_{proj} = X_{attn}W_{proj}^\top + b_{proj}$	X_{attn}	+1
$X_{in2} = X_{in1} + X_{proj}$	\	+0	\	+0		$X_{in2} = X_{in1} + X_{proj}$	\	+0
$\widetilde{X}_{ln2} = \mathrm{LN}(X_{in2})$	$X_{in2}, \mu_{in2}, (\sigma^2_{in2}+\varepsilon)^{-\frac{1}{2}}$	+2	$X_{in2}, \mu_{in2}, (\sigma^2_{in2}+\varepsilon)^{-\frac{1}{2}}$	+2	MS-BP	$X_{ln2} = \text{MS-LN}(X_{in2})$	$X_{ln2}, (\sigma^2_{in2}+\varepsilon)^{-\frac{1}{2}}$	+1
$X_{ln2} = \widetilde{X}_{in2}\mathrm{diag}(\alpha_2) + \beta_2$								
$X_{fc1} = X_{ln2}W_{fc1}^\top + b_{fc1}$	X_{ln2}	+1	\	+0		$X_{fc1} = X_{ln2}\widetilde{W}_{fc1}^\top + \widetilde{b}_{fc1}$		
$X_{gelu} = \mathrm{GELU}(X_{fc1})$	X_{fc1}	+4	X_{fc1}	+4	Approx-BP	$X_{gelu} = \text{ReGELU2}(X_{fc1})$	$sgns$	+0.5
$X_{fc2} = X_{gelu}W_{fc2}^\top + b_{fc2}$	X_{gelu}	+4	\	+0		$X_{fc2} = X_{gelu}W_{fc2}^\top + b_{fc2}$	X_{gelu}	+4
$X_{out} = X_{in2} + X_{fc2}$	\	+0	\	+0		$X_{out} = X_{in2} + X_{fc2}$	\	+0
Activation Memory	**19**		**12**				**11.5**	

Figure 5. **Composition of the activation memory in each block of ViT (Dosovitskiy et al., 2021).** We assume Layer Normalization uses fp32, other operators use fp16 data type and each operator in the table is implemented as a single CUDA kernel. **The unit of memory is the memory size of a tensor (16 bits type) with the shape** $[b, n, c]$.

By Taylor expansion with Lagrange remainder, $\forall t \in (0, \infty)$ and $\forall \boldsymbol{q} \in \mathbb{R}^M$, we have

$$\begin{aligned}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{\theta} + t\boldsymbol{q})) - \ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta} + t\boldsymbol{q})) =& \ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{\theta})) - \ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})) + t\boldsymbol{q}^\top \left(\frac{\partial}{\partial \boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{\theta})) - \frac{\partial}{\partial \boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}))\right) \\ &+ \frac{t^2}{2}\boldsymbol{q}^\top \left(\frac{\partial^2}{\partial \boldsymbol{\theta}\partial \boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{\theta} + \xi_1\boldsymbol{q})) - \frac{\partial^2}{\partial \boldsymbol{\theta}\partial \boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta} + \xi_1\boldsymbol{q}))\right)\boldsymbol{q}, \qquad (24a)\end{aligned}$$

$$\begin{aligned}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{\theta} - t\boldsymbol{q})) - \ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta} - t\boldsymbol{q})) =& \ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{\theta})) - \ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})) - t\boldsymbol{q}^\top \left(\frac{\partial}{\partial \boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{\theta})) - \frac{\partial}{\partial \boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}))\right) \\ &+ \frac{t^2}{2}\boldsymbol{q}^\top \left(\frac{\partial^2}{\partial \boldsymbol{\theta}\partial \boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{\theta} - \xi_2\boldsymbol{q})) - \frac{\partial^2}{\partial \boldsymbol{\theta}\partial \boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta} - \xi_2\boldsymbol{q}))\right)\boldsymbol{q}, \qquad (24b)\end{aligned}$$

LLaMA Block	Save for Backward: if parameters trainable		Save for Backward: if parameters frozen			Our Method	Save for Backward: if parameters trainable	
$\widetilde{X}_{in1} = \text{RMSNorm}(X_{in1})$	X_{in1}, $(\sigma_{in1}^2+\varepsilon)^{-\frac{1}{2}}$	+2	X_{in1}, $(\sigma_{in1}^2+\varepsilon)^{-\frac{1}{2}}$	+2	MS-BP	$X_{ln1} = \text{MS-RMSNorm}(X_{in1})$	X_{ln1}, $(\sigma_{in1}^2+\varepsilon)^{-\frac{1}{2}}$	+1
$X_{ln1} = \widetilde{X}_{in1}\text{diag}(\alpha_1)$								
for $y = q,k,v$ $y = X_{ln1}W_y^\top$	X_{ln1}	+1	\	+0		for $y = q,k,v$ $y = X_{ln1}\widetilde{W}_y^\top$		
for $y = q,k,v$ $y_{\text{MH}} = \text{reshape}(y, [b,n,c] \to [b,h,n,c/h])$	\	+0	\	+0		for $y = q,k,v$ $y_{\text{MH}} = \text{reshape}(y, [b,n,c] \to [b,h,n,c/h])$	\	+0
$o_{\text{MH}} = \text{flashattn}(q_{\text{MH}}, k_{\text{MH}}, v_{\text{MH}})$	$q_{\text{MH}}, k_{\text{MH}}, v_{\text{MH}}$, o_{MH}, m, l	+4	$q_{\text{MH}}, k_{\text{MH}}, v_{\text{MH}}$, o_{MH}, m, l	+4		$o_{\text{MH}} = \text{flashattn}(q_{\text{MH}}, k_{\text{MH}}, v_{\text{MH}})$	$q_{\text{MH}}, k_{\text{MH}}, v_{\text{MH}}$, o_{MH}, m, l	+4
$X_{attn} = \text{reshape}(o_{\text{MH}}, [b,h,n,c/h] \to [b,n,c])$	\	+0	\	+0		$X_{attn} = \text{reshape}(o_{\text{MH}}, [b,h,n,c/h] \to [b,n,c])$	\	+0
$X_{proj} = X_{attn}W_{proj}^\top$	X_{attn}	+1	\	+0		$X_{proj} = X_{attn}W_{proj}^\top$	X_{attn}	+1
$X_{in2} = X_{in1} + X_{proj}$	\	+0	\	+0		$X_{in2} = X_{in1} + X_{proj}$	\	+0
$\widetilde{X}_{in2} = \text{RMSNorm}(X_{in2})$	X_{in2}, $(\sigma_{in2}^2+\varepsilon)^{-\frac{1}{2}}$	+2	X_{in2}, $(\sigma_{in2}^2+\varepsilon)^{-\frac{1}{2}}$	+2	MS-BP	$X_{ln2} = \text{MS-RMSNorm}(X_{in2})$	X_{ln2}, $(\sigma_{in2}^2+\varepsilon)^{-\frac{1}{2}}$	+1
$X_{ln2} = \widetilde{X}_{in2}\text{diag}(\alpha_2)$								
$X_{fc1} = X_{ln2}W_{fc1}^\top$	X_{ln2}	+1	\	+0		$X_{fc1} = X_{ln2}\widetilde{W}_{fc1}^\top$		
$X_{fc2} = X_{ln2}W_{fc2}^\top$			\	+0		$X_{fc2} = X_{ln2}\widetilde{W}_{fc2}^\top$		
$X_{silu} = \text{SiLU}(X_{fc2})$	X_{fc2}	+2.7	X_{fc2}	+2.7	Approx-BP	$X_{silu} = \text{ReSiLU2}(X_{fc2})$	$sgns$	+0.3375
$X_{gate} = X_{silu} * X_{fc1}$	X_{silu}, X_{fc1}	+5.4	X_{silu}, X_{fc1}	+5.4		$X_{gate} = X_{silu} * X_{fc1}$	X_{silu}, X_{fc1}	+5.4
$X_{fc3} = X_{gate}W_{fc3}^\top$	X_{gate}	+2.7	\	+0		$X_{fc3} = X_{gate}W_{fc3}^\top$	\	+2.7
$X_{out} = X_{in2} + X_{fc3}$	\	+0	\	+0		$X_{out} = X_{in2} + X_{fc3}$	\	+0
Activation Memory	**21.8**		**16.1**				**15.4375**	

Figure 6. **Composition of activation memory in each block of LLaMA (Touvron et al., 2023).** Here, RMSNorm uses fp32, other operators use bf16 data type, and each operator is implemented as a single CUDA kernel. In practice, RMSNorm is often implemented by multiple sub-operators, which may bring additional memory usage. The unit of memory in this figure is the memory size of a tensor (16 bits type) with the shape $[b, n, c]$. The expanding factor in LLaMA depends on the model size, we use LLaMA-13B as an example.

where $\xi_1, \xi_2 \in (0, t)$. $\frac{\partial^2}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}))$ and $\frac{\partial^2}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}))$ are the Hessian matrices of $\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta})$ and $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta})$, respectively.

From (24a) and (24b), we derive

$$\begin{aligned}
&\boldsymbol{q}^\top(\frac{\partial}{\partial\boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}))-\frac{\partial}{\partial\boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta})))\\
=&\frac{1}{2t}(\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}+t\boldsymbol{q}))-\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}+t\boldsymbol{q}))-\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}-t\boldsymbol{q}))+\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}-t\boldsymbol{q})))\\
+&\frac{t}{4}\boldsymbol{q}^\top(\frac{\partial^2}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}-\xi_2\boldsymbol{q}))-\frac{\partial^2}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}-\xi_2\boldsymbol{q}))-\frac{\partial^2}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}+\xi_1\boldsymbol{q}))+\frac{\partial^2}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}+\xi_1\boldsymbol{q})))\boldsymbol{q}\\
\leq&\frac{1}{t}\sup_{\boldsymbol{\theta}}|\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}))-\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}))|+\frac{t}{2}(M_1+M_2)\boldsymbol{q}^\top\boldsymbol{q}.
\end{aligned}\tag{25}$$

Since (25) is valid for all $\boldsymbol{q}\in\mathbb{R}^M$ and $t\in(0,\infty)$, by setting

$$\begin{aligned}
\boldsymbol{q}=&\frac{\partial}{\partial\boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}))-\frac{\partial}{\partial\boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta})),\\
t=&\sqrt{\frac{2\sup_{\boldsymbol{\theta}}|\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}))-\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}))|}{(M_1+M_2)\|\frac{\partial}{\partial\boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}))-\frac{\partial}{\partial\boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}))\|_2^2}},
\end{aligned}\tag{26}$$

we have

$$\begin{aligned}
\|\frac{\partial}{\partial\boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}))-\frac{\partial}{\partial\boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}))\|_2&\leq\sqrt{2(M_1+M_2)}\sqrt{\sup_{\boldsymbol{\theta}}|\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}))-\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}))|}\\
&=\sqrt{2(M_1+M_2)a_2}\sqrt{\sup_{\boldsymbol{\theta}}\|\widetilde{\boldsymbol{z}}^L-\boldsymbol{z}^L\|_2}.
\end{aligned}\tag{27}$$

By A2, for $i=2,...,L$, $\exists b_i>0$, $\forall\boldsymbol{\theta}^i$, we have

$$\|\boldsymbol{h}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)-\boldsymbol{h}^i(\widetilde{\boldsymbol{z}}^{i-1},\boldsymbol{\theta}^i)\|_2\leq b_i\|\boldsymbol{z}^{i-1}-\widetilde{\boldsymbol{z}}^{i-1}\|_2.\tag{28}$$

Therefore, we attain

$$\begin{aligned}
&\|\widetilde{\boldsymbol{z}}^i-\boldsymbol{z}^i\|_2\\
=&\|\widetilde{\boldsymbol{h}}^i_{\boldsymbol{\theta}}\circ\widetilde{\boldsymbol{h}}^{i-1}_{\boldsymbol{\theta}}\circ...\circ\widetilde{\boldsymbol{h}}^1_{\boldsymbol{\theta}}(\boldsymbol{x})-\boldsymbol{h}^i_{\boldsymbol{\theta}}\circ\boldsymbol{h}^{i-1}_{\boldsymbol{\theta}}\circ...\circ\boldsymbol{h}^1_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2\\
\leq&\|\widetilde{\boldsymbol{h}}^i_{\boldsymbol{\theta}}\circ\widetilde{\boldsymbol{h}}^{i-1}_{\boldsymbol{\theta}}\circ...\circ\widetilde{\boldsymbol{h}}^1_{\boldsymbol{\theta}}(\boldsymbol{x})-\boldsymbol{h}^i_{\boldsymbol{\theta}}\circ\widetilde{\boldsymbol{h}}^{i-1}_{\boldsymbol{\theta}}\circ...\circ\widetilde{\boldsymbol{h}}^1_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2+\|\boldsymbol{h}^i_{\boldsymbol{\theta}}\circ\widetilde{\boldsymbol{h}}^{i-1}_{\boldsymbol{\theta}}\circ...\circ\widetilde{\boldsymbol{h}}^1_{\boldsymbol{\theta}}(\boldsymbol{x})-\boldsymbol{h}^i_{\boldsymbol{\theta}}\circ\boldsymbol{h}^{i-1}_{\boldsymbol{\theta}}\circ...\circ\boldsymbol{h}^1_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2\\
\leq&\sup_{\boldsymbol{z}^{i-1}}\|\widetilde{\boldsymbol{h}}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)-\boldsymbol{h}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)\|_2+b_i\|\widetilde{\boldsymbol{h}}^{i-1}_{\boldsymbol{\theta}}\circ\widetilde{\boldsymbol{h}}^{i-2}_{\boldsymbol{\theta}}\circ...\circ\widetilde{\boldsymbol{h}}^1_{\boldsymbol{\theta}}(\boldsymbol{x})-\boldsymbol{h}^{i-1}_{\boldsymbol{\theta}}\circ\boldsymbol{h}^{i-2}_{\boldsymbol{\theta}}\circ...\circ\boldsymbol{h}^1_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2\\
\leq&\sup_{\boldsymbol{z}^{i-1}}\|\widetilde{\boldsymbol{h}}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)-\boldsymbol{h}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)\|_2+b_i\sup_{\boldsymbol{z}^{i-2}}\|\widetilde{\boldsymbol{h}}^{i-1}(\boldsymbol{z}^{i-2},\boldsymbol{\theta}^{i-1})-\boldsymbol{h}^{i-1}(\boldsymbol{z}^{i-2},\boldsymbol{\theta}^{i-1})\|_2\\
+&...+b_ib_{i-1}...b_2\sup_{\boldsymbol{z}^0}\|\widetilde{\boldsymbol{h}}^1(\boldsymbol{z}^0,\boldsymbol{\theta}^1)-\boldsymbol{h}^1(\boldsymbol{z}^0,\boldsymbol{\theta}^1)\|_2.
\end{aligned}\tag{29}$$

From (22) and (29), we derive that $\exists\alpha_1>0$, such that

$$\begin{aligned}
\|\widetilde{\boldsymbol{g}}(\ell(\boldsymbol{z}^L),\boldsymbol{z},\boldsymbol{\theta})-\widetilde{\boldsymbol{g}}(\ell(\widetilde{\boldsymbol{z}}^L),\widetilde{\boldsymbol{z}},\boldsymbol{\theta})\|_2&\leq\alpha_1\sum_{i=1}^L\sup_{\boldsymbol{z}^{i-1}}\|\widetilde{\boldsymbol{h}}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)-\boldsymbol{h}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)\|_2\\
&\leq\alpha_1\sum_{i=1}^L\sup_{\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i}\|\widetilde{\boldsymbol{h}}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)-\boldsymbol{h}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)\|_2.
\end{aligned}\tag{30}$$

From (27) and (29), we derive that $\exists\alpha_2>0$, such that

$$\|\frac{\partial}{\partial\boldsymbol{\theta}}\ell(\widetilde{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{\theta}))-\frac{\partial}{\partial\boldsymbol{\theta}}\ell(\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta}))\|_2\leq\alpha_2\sqrt{\sum_{i=1}^L\sup_{\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i}\|\widetilde{\boldsymbol{h}}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)-\boldsymbol{h}^i(\boldsymbol{z}^{i-1},\boldsymbol{\theta}^i)\|_2}.\tag{31}$$

By (19) and setting $\alpha = \max\{\alpha_1, \alpha_2\}$, we attain

$$\|\widehat{\boldsymbol{g}} - \boldsymbol{g}\|_2 \leq \alpha \left(\sum_{i=1}^{L} \sup_{\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i} \|\boldsymbol{h}^i(\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i) - \widetilde{\boldsymbol{h}}^i(\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i)\|_2 + \sqrt{\sum_{i=1}^{L} \sup_{\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i} \|\boldsymbol{h}^i(\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i) - \widetilde{\boldsymbol{h}}^i(\boldsymbol{z}^{i-1}, \boldsymbol{\theta}^i)\|_2} \right). \quad (32)$$

□

Proof of Theorem 4.2. In Approx-BP training, an update step of parameters is denoted by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \widehat{\boldsymbol{g}}_t. \quad (33)$$

By Assumption 4.1 in (Bottou et al., 2018), we have

$$\begin{aligned} \ell(\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{\theta}_{t+1})) \leq & \ell(\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{\theta}_t)) + \boldsymbol{g}_t^\top (\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t) + \frac{\beta}{2} \|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|_2^2 \\ = & \ell(\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{\theta}_t)) - \eta \boldsymbol{g}_t^\top \hat{\boldsymbol{g}}_t + \frac{\eta^2 \beta}{2} \|\hat{\boldsymbol{g}}_t\|_2^2. \end{aligned} \quad (34)$$

Then, using assumption $\eta < \frac{1}{2\beta}$, we have

$$\begin{aligned} \ell(\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{\theta}_{t+1})) - \ell(\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{\theta}_t)) \leq & -\eta \boldsymbol{g}_t^\top (\boldsymbol{g}_t - \boldsymbol{g}_t + \hat{\boldsymbol{g}}_t) + \frac{\eta^2 \beta}{2} \|\boldsymbol{g}_t - \boldsymbol{g}_t + \hat{\boldsymbol{g}}_t\|_2^2 \\ \leq & -\eta \|\boldsymbol{g}_t\|_2^2 + \eta \|\boldsymbol{g}_t - \hat{\boldsymbol{g}}_t\| \|\boldsymbol{g}_t\| + \eta^2 \beta \|\boldsymbol{g}_t\|_2^2 + \eta^2 \beta \|\boldsymbol{g}_t - \hat{\boldsymbol{g}}_t\|_2^2 \\ \leq & -\frac{\eta}{2} (\|\boldsymbol{g}_t\| - \|\boldsymbol{g}_t - \hat{\boldsymbol{g}}_t\|)^2 + \eta \|\boldsymbol{g}_t - \hat{\boldsymbol{g}}_t\|_2^2 \\ \leq & -\frac{\eta}{4} \|\boldsymbol{g}_t\|_2^2 + \frac{3\eta}{2} \|\boldsymbol{g}_t - \hat{\boldsymbol{g}}_t\|_2^2. \end{aligned} \quad (35)$$

Now we obtain that

$$\|\boldsymbol{g}_t\|_2^2 \leq \frac{4}{\eta} (\ell(\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{\theta}_t)) - \ell(\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{\theta}_{t+1}))) + 6 \|\boldsymbol{g}_t - \hat{\boldsymbol{g}}_t\|_2^2. \quad (36)$$

Taking expectation of (36) for $\boldsymbol{x}_t \sim \mathcal{D}$, we have

$$\mathbb{E}_{\mathcal{D}} \|\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_t))\|_2^2 \leq \frac{4}{\eta} [\mathbb{E}_{\mathcal{D}} \ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_t)) - \mathbb{E}_{\mathcal{D}} \ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_{t+1}))] + 6\sigma^2. \quad (37)$$

Taking average of (37) over $t = 0, ..., T-1$, we have

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}_{\mathcal{D}} \|\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_t))\|_2^2 \leq \frac{4}{\eta T} [\mathbb{E}_{\mathcal{D}} \ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_0)) - \mathbb{E}_{\mathcal{D}} \ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_T))] + 6\sigma^2 \leq \frac{4}{\eta T} [\mathbb{E}_{\mathcal{D}} \ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_0)) - \ell^*] + 6\sigma^2. \quad (38)$$

Therefore, we conclude that

$$\min_{t \in \{0, ..., T-1\}} \mathbb{E}_{\mathcal{D}} \|\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_t))\|_2^2 \leq \frac{4[\mathbb{E}_{\mathcal{D}} \ell(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}_0)) - \ell^*]}{\eta T} + 6\sigma^2. \quad (39)$$

□

E. Derivation of Our ReGELU2 and ReSiLU2

E.1. Proposed ReGELU2

We denote GELU as h, and

$$h(x) = \frac{x}{2} (1 + \operatorname{erf}(\frac{x}{\sqrt{2}})). \quad (40)$$

Then we define the approximate activation function $\widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}$ of GELU h as follows:

$$\widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x) = a_1\max\{x-c_1,0\} + a_2\max\{x-c_2,0\} + (1-a_1-a_2)\max\{x-c_3,0\}. \tag{41}$$

The optimization objective is

$$\min_{\boldsymbol{a},\boldsymbol{c}} \int_{-\infty}^{\infty} (h(x) - \widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x))^2 \mathrm{d}x. \tag{42}$$

We first perform a tail estimation for the integral in the objective. Note that $\widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x) \equiv 0$ for $x < \min\{\boldsymbol{c}\}$, *i.e.*, the minimal value in the vector $\boldsymbol{c}$, and $\widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x) \equiv x$ for $x > \max\{\boldsymbol{c}\}$, *i.e.*, the maximum value in the vector $\boldsymbol{c}$. So the left tail of the integral can be estimated as follows, for a certain $A < 0$:

$$\begin{aligned}
&\int_{-\infty}^{A} (h(x) - \widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x))^2 \mathrm{d}x = \int_{-\infty}^{A} (\frac{x}{2}(1+\mathrm{erf}(\frac{x}{\sqrt{2}})))^2 \mathrm{d}x \\
<&\int_{-\infty}^{A} -\frac{x}{2}(1+\mathrm{erf}(\frac{x}{\sqrt{2}}))\mathrm{d}x = \int_{-\infty}^{\frac{A}{\sqrt{2}}} -x(1+\mathrm{erf}(x))\mathrm{d}x \\
<&\int_{-\infty}^{\frac{A}{\sqrt{2}}} -x(1-\sqrt{1-e^{-x^2}})\mathrm{d}x < \int_{-\infty}^{\frac{A}{\sqrt{2}}} -xe^{-x^2}\mathrm{d}x = \frac{1}{2}e^{-\frac{A^2}{2}}.
\end{aligned} \tag{43}$$

The right tail of the integral can be estimated as follows, for a certain $B > 0$:

$$\begin{aligned}
&\int_{B}^{+\infty} (h(x) - \widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x))^2 \mathrm{d}x = \int_{B}^{+\infty} (\frac{x}{2}(1-\mathrm{erf}(\frac{x}{\sqrt{2}})))^2 \mathrm{d}x \\
<&\int_{B}^{+\infty} \frac{x}{2}(1-\mathrm{erf}(\frac{x}{\sqrt{2}}))\mathrm{d}x = \int_{\frac{B}{\sqrt{2}}}^{+\infty} x(1-\mathrm{erf}(x))\mathrm{d}x \\
<&\int_{\frac{B}{\sqrt{2}}}^{+\infty} x(1-\sqrt{1-e^{-x^2}})\mathrm{d}x < \int_{\frac{B}{\sqrt{2}}}^{+\infty} xe^{-x^2}\mathrm{d}x = \frac{1}{2}e^{-\frac{B^2}{2}}.
\end{aligned} \tag{44}$$

The condition of scaling inequalities above can be summarized as $\frac{|x|}{2}(1-\mathrm{erf}(\frac{|x|}{\sqrt{2}})) < 1$ for $|x| > \max\{|A|,|B|\}$. When setting $B = -A = \sqrt{-2\ln(\varepsilon)}$, we have the following bounds:

$$\int_{-\infty}^{A} (h(x) - \widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x))^2 \mathrm{d}x + \int_{B}^{+\infty} (h(x) - \widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x))^2 \mathrm{d}x < \varepsilon. \tag{45}$$

We set $\varepsilon = 10^{-8}$ to satisfy the condition of scaling inequalities and bound the two-side tails of integral in a negligible value.

Now we only need to solve the following optimization objective:

$$\min_{\boldsymbol{a},\boldsymbol{c}} \int_{A}^{B} (h(x) - \widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x))^2 \mathrm{d}x. \tag{46}$$

This time, the integral in the objective is a definite integral over a bounded interval, which can be calculated by many numerical computing methods (Piessens et al., 1983; Virtanen et al., 2020). Although the above optimization objective is not convex, it is not difficult to find a good solution, since there are only five scalar variables. We have tried simulated annealing algorithm (Kirkpatrick et al., 1983) and stochastic gradient descent algorithm (Robbins & Monro, 1951), and both can find good solutions that are close to each other, as long as searching multiple times with different initialization. The following solution is obtained by simulated annealing algorithm (Kirkpatrick et al., 1983), which is adopted in our code:

$$\begin{aligned}
\boldsymbol{a}^* &= [-0.04922261145617846, 1.0979632065417297]^\top, \\
\boldsymbol{c}^* &= [-3.1858810036855245, -0.001178821281161997, 3.190832613414926]^\top.
\end{aligned}$$

We plot our ReGELU2 in Figure 7. In principle, there should be an additional operation during or after the optimization to compel the solutions to fulfill the constraint in (13). However, we found the constraint is already satisfied due to the inherent property of the L2 metrics.

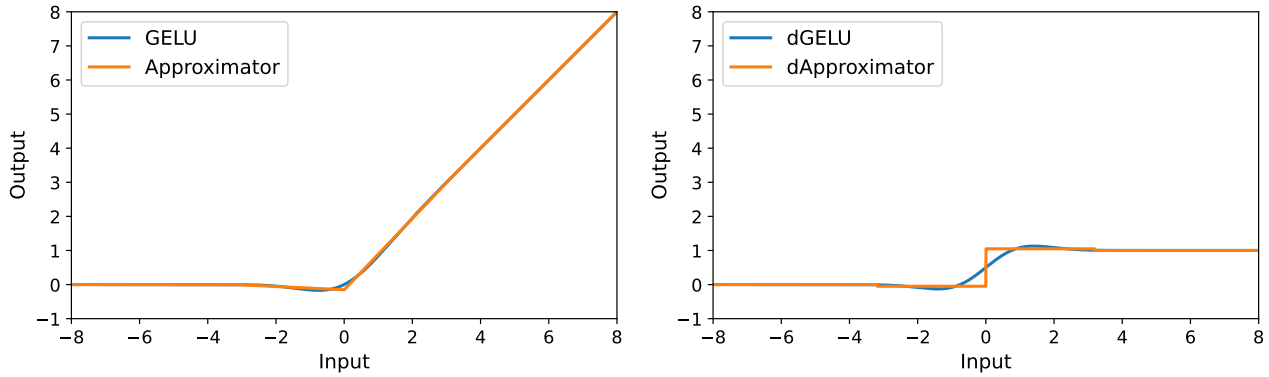

Figure 7. **Plot curve of our ReGELU2**. The primitive function is the same as GELU. The derivative function is the same as the dApproximator (derivative of the approximate activation function $\widetilde{h}_{\boldsymbol{a}^*,\boldsymbol{c}^*}$ of GELU h), a 4-segment step function that needs 2 bits to store the derivative information of each element.

E.2. Proposed ReSiLU2

The derivation of our ReSiLU2 is similar to that for our ReGELU2. We also denote SiLU as h,

$$h(x) = \frac{x}{1+e^{-x}}. \tag{47}$$

And our optimization objective is the same as ReGELU2,

$$\min_{\boldsymbol{a},\boldsymbol{c}} \int_{-\infty}^{\infty} (h(x) - \widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x))^2 \mathrm{d}x. \tag{48}$$

Again, we perform a tail estimation for the integral in the objective. Since $\widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x) \equiv 0$ for $x < \min\{\boldsymbol{c}\}$, *i.e.*, the minimal value in the vector $\boldsymbol{c}$, and $\widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x) \equiv x$ for $x > \max\{\boldsymbol{c}\}$, *i.e.*, the maximum value in the vector $\boldsymbol{c}$, the left tail of the integral can be estimated as follows, for a certain $A < 0$:

$$\begin{aligned} &\int_{-\infty}^{A} (h(x) - \widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x))^2 \mathrm{d}x = \int_{-\infty}^{A} (\frac{x}{1+e^{-x}})^2 \mathrm{d}x \\ <&\int_{-\infty}^{A} \frac{-x}{1+e^{-x}} \mathrm{d}x < \int_{-\infty}^{A} -xe^{x} \mathrm{d}x = (1-A)e^{A} < e^{\frac{A}{2}}. \end{aligned} \tag{49}$$

The right tail of the integral can be estimated as follows, for a certain $B < 0$:

$$\begin{aligned} &\int_{B}^{+\infty} (h(x) - \widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x))^2 \mathrm{d}x = \int_{B}^{+\infty} (\frac{x}{1+e^{x}})^2 \mathrm{d}x \\ <&\int_{B}^{+\infty} \frac{x}{1+e^{x}} \mathrm{d}x < \int_{B}^{+\infty} xe^{-x} \mathrm{d}x = (1+B)e^{-B} < e^{-\frac{B}{2}}. \end{aligned} \tag{50}$$

The condition of scaling inequalities above can be summarized as $\frac{|x|}{1+e^{|x|}} < 1$ for $|x| > \max\{|A|, |B|\}$ and $1 - A < e^{-\frac{A}{2}}$ and $1 + B < e^{\frac{B}{2}}$. When setting $B = -A = -2\ln(\frac{\varepsilon}{2})$, we have the following bounds:

$$\int_{-\infty}^{A} (h(x) - \widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x))^2 \mathrm{d}x + \int_{B}^{+\infty} (h(x) - \widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x))^2 \mathrm{d}x < \varepsilon. \tag{51}$$

We set $\varepsilon = 10^{-8}$ to satisfy the condition of scaling inequalities and bound the two-side tails of integral in a negligible value.

Now we only need to consider the following optimization objective:

$$\min_{\boldsymbol{a},\boldsymbol{c}} \int_{A}^{B} (h(x) - \widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x))^2 \mathrm{d}x. \tag{52}$$

This time, the integral in the objective is a definite integral over a bounded interval, which can also be calculated be many numerical methods (Piessens et al., 1983; Virtanen et al., 2020). Similarly, although the above optimization objective is not convex, it is not difficult to find a good solution, since there are only five scalar variables. We have tried simulated annealing algorithm (Kirkpatrick et al., 1983) and stochastic gradient descent algorithm (Robbins & Monro, 1951), and both can find good solutions that are close to each other, as long as searching multiple times with different initialization. The following solution is obtained by simulated annealing algorithm (Kirkpatrick et al., 1983), which is adopted in our code:

$$\begin{aligned}\boldsymbol{a}^* &= [-0.04060357190528599, 1.080925428529668]^\top,\\ \boldsymbol{c}^* &= [-6.3050461001646445, -0.0008684942046214787, 6.325815242089708]^\top.\end{aligned}$$

We plot our ReSiLU2 in Figure 8. In principle, there should be an additional operation during or after the optimization to compel the solutions to fulfill the constraint in (13). However, we found the constraint is already satisfied due to the inherent property of the L2 metrics.

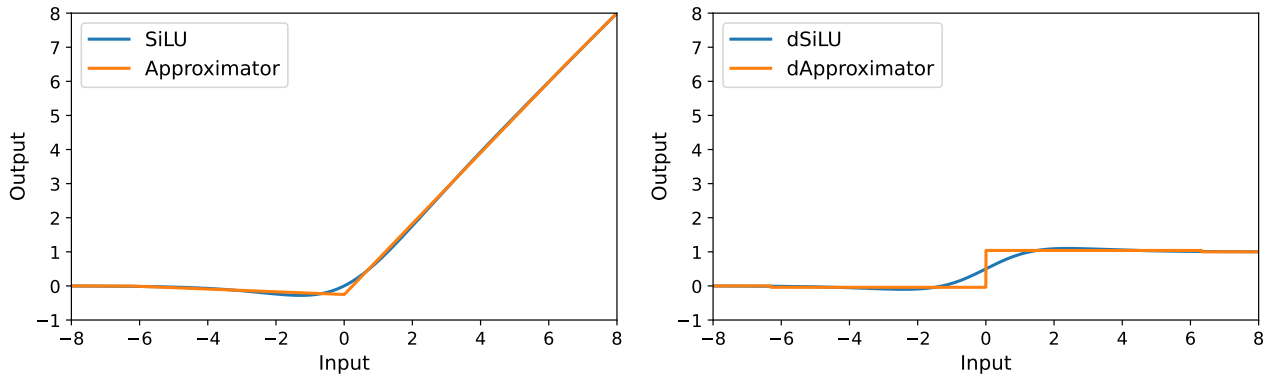

Figure 8. **Plot curve of our ReSiLU2**. The primitive function is the same as SiLU. The derivative function is the same as the dApproximator (derivative of the approximate activation function $\widetilde{h}_{\boldsymbol{a}^*,\boldsymbol{c}^*}$ of SiLU h), a 4-segment step function that needs 2 bits to store the derivative information of each element.

F. Memory-Sharing Activation Function

Suppose $\boldsymbol{h}^i$ is a layer of element-wise activation function. The forward pass at $\boldsymbol{h}^i$ can be expressed as:

$$\boldsymbol{z}^i = \boldsymbol{h}^i(\boldsymbol{z}^{i-1}). \tag{53}$$

The backward pass at $\boldsymbol{h}^i$ can be expressed as:

$$\frac{\partial \ell}{\partial \boldsymbol{z}^{i-1}} = \frac{\partial \boldsymbol{h}^i(\boldsymbol{z}^{i-1})}{\partial \boldsymbol{z}^{i-1}} \frac{\partial \ell}{\partial \boldsymbol{z}^i}. \tag{54}$$

The first condition of Proposition 5.1 is immediately satisfied. The third condition of Proposition 5.1 depends on the model architecture and the fine-tuning methods. Here, we mainly consider the second condition of Proposition 5.1. Since $\boldsymbol{h}^i$ is element-wise, we denote the scalar activation function in $\boldsymbol{h}^i$ as h. Now, the second condition of Proposition 5.1 can be rephrased as $\mathrm{d}h(x) = J(h(x))$, where J is a certain function. Some simple activation functions, such as ReLU and Sigmoid, satisfy this condition apparently:

$$\begin{aligned}\mathrm{dReLU}(x) &= \mathrm{sgn}(\mathrm{ReLU}(x)),\\ \mathrm{d}\sigma(x) &= \sigma(x)(1-\sigma(x)),\end{aligned} \tag{55}$$

where "sgn" is the sign function and $\sigma(x)$ is the Sigmoid function.

However, it is challenging to answer whether a complicated activation function like SiLU satisfies this condition. Here, we

conclude that SiLU does not satisfy such condition. To show this, we first give the analytic form of $h(x)$ and $\mathrm{d}h(x)$:

$$\begin{aligned} h(x) &= x\sigma(x), \\ \mathrm{d}h(x) &= \sigma(x) + x\sigma(x) - x\sigma(x)^2 \\ &= \frac{h(x) - h(x)^2}{x} + h(x). \end{aligned} \tag{56}$$

If $\mathrm{d}h(x) = J(h(x))$ for some function J, then $\mathrm{d}h(x)$ is decided only by $h(x)$. Since $h(x)$ is not injective, there exits $x_1 \neq x_2$ such that $h(x_1) = h(x_2) \notin \{0, 1\}$, which derive $\mathrm{d}h(x_1) - \mathrm{d}h(x_2) = J(h(x_1)) - J(h(x_2)) = 0$. However, from (56), we also derive $\mathrm{d}h(x_1) - \mathrm{d}h(x_2) = (h(x_1) - h(x_1)^2)(\frac{1}{x_1} - \frac{1}{x_2}) \neq 0$, resulting in a contradiction.

G. Memory-Sharing LayerNorm and RMSNorm

G.1. Proposed Memory-Sharing LayerNorm (MS-LN)

The forward pass at LayerNorm and its following linear layer is as follows:

$$\begin{aligned} &\text{Suppose } \boldsymbol{z}^{i-1} \in \mathbb{R}^{p_{i-1}}, \boldsymbol{H} = \mathbb{I} - p_{i-1}^{-1}\mathbb{1}\mathbb{1}^\top, \\ &\sigma = \sqrt{p_{i-1}^{-1}{\boldsymbol{z}^{i-1}}^\top \boldsymbol{H}\boldsymbol{z}^{i-1} + \varepsilon}, \\ &\widetilde{\boldsymbol{z}}^{i-1} = \sigma^{-1}\boldsymbol{H}\boldsymbol{z}^{i-1}, \\ &\boldsymbol{z}^i = \mathrm{diag}(\boldsymbol{\alpha})\widetilde{\boldsymbol{z}}^{i-1} + \boldsymbol{\beta}, \\ &\boldsymbol{z}^{i+1} = \boldsymbol{W}\boldsymbol{z}^i + \boldsymbol{b}. \end{aligned} \tag{57}$$

We can merge the affine parameters in LayerNorm and the parameters in the following linear layer as follows:

$$\begin{aligned} \widetilde{\boldsymbol{W}} &= \boldsymbol{W}\mathrm{diag}(\boldsymbol{\alpha}), \\ \widetilde{\boldsymbol{b}} &= \boldsymbol{W}\boldsymbol{\beta} + \boldsymbol{b}. \end{aligned} \tag{58}$$

Then the forward pass at a merged LayerNorm and the following linear layer becomes:

$$\begin{aligned} &\text{Suppose } \boldsymbol{x} \in \mathbb{R}^{p_{i-1}}, \boldsymbol{H} = \mathbb{I} - p_{i-1}^{-1}\mathbb{1}\mathbb{1}^\top, \\ &\sigma = \sqrt{p_{i-1}^{-1}{\boldsymbol{z}^{i-1}}^\top \boldsymbol{H}\boldsymbol{z}^{i-1} + \varepsilon}, \\ &\boldsymbol{z}^i = \sigma^{-1}\boldsymbol{H}\boldsymbol{z}^{i-1}, \\ &\boldsymbol{z}^{i+1} = \widetilde{\boldsymbol{W}}\boldsymbol{z}^i + \widetilde{\boldsymbol{b}}. \end{aligned} \tag{59}$$

The program of our MS-LN is shown in Algorithm 2.

Algorithm 2 Memory-Sharing LayerNorm (MS-LN)

Suppose $\boldsymbol{H} = \mathbb{I} - p_{i-1}^{-1}\mathbb{1}\mathbb{1}^\top$, ℓ is the loss function.
Input: $\boldsymbol{z}^{i-1} \in \mathbb{R}^{p_{i-1}}$
Forward:
$\sigma = \sqrt{p_{i-1}^{-1}{\boldsymbol{z}^{i-1}}^\top \boldsymbol{H}\boldsymbol{z}^{i-1} + \varepsilon}$
$\boldsymbol{z}^i = \sigma^{-1}\boldsymbol{H}\boldsymbol{z}^{i-1}$
Save for backward: $\boldsymbol{z}^i$, σ
Return Output: $\boldsymbol{z}^i$
Backward:
Receive gradient: $\frac{\partial \ell}{\partial \boldsymbol{z}^i}$
$\frac{\partial \ell}{\partial \boldsymbol{z}^{i-1}} = \sigma^{-1}(\boldsymbol{H} - p_{i-1}^{-1}\boldsymbol{z}^i{\boldsymbol{z}^i}^\top)\frac{\partial \ell}{\partial \boldsymbol{z}^i}$
Return Gradient: $\frac{\partial \ell}{\partial \boldsymbol{z}^{i-1}}$

Algorithm 3 Memory-Sharing RMSNorm (MS-RMSNorm)

Suppose ℓ is the loss function.
Input: $\boldsymbol{z}^{i-1} \in \mathbb{R}^{p_{i-1}}$
Forward:
$\sigma = \sqrt{p_{i-1}^{-1}{\boldsymbol{z}^{i-1}}^\top \boldsymbol{z}^{i-1} + \varepsilon}$
$\boldsymbol{z}^i = \sigma^{-1}\boldsymbol{z}^{i-1}$
Save for backward: $\boldsymbol{z}^i$, σ
Return Output: $\boldsymbol{z}^i$
Backward:
Receive gradient: $\frac{\partial \ell}{\partial \boldsymbol{z}^i}$
$\frac{\partial \ell}{\partial \boldsymbol{z}^{i-1}} = \sigma^{-1}(\mathbb{I} - p_{i-1}^{-1}\boldsymbol{z}^i{\boldsymbol{z}^i}^\top)\frac{\partial \ell}{\partial \boldsymbol{z}^i}$
Return Gradient: $\frac{\partial \ell}{\partial \boldsymbol{z}^{i-1}}$

G.2. Proposed Memory-Sharing RMSNorm (MS-RMSNorm)

The forward pass at RMSNorm and its following linear layer is as follows:

$$\begin{aligned}
&\text{Suppose } \boldsymbol{z}^{i-1} \in \mathbb{R}^{p_{i-1}}, \\
&\sigma = \sqrt{p_{i-1}^{-1} \boldsymbol{z}^{i-1\top} \boldsymbol{z}^{i-1} + \varepsilon}, \\
&\widetilde{\boldsymbol{z}}^{i-1} = \sigma^{-1} \boldsymbol{z}^{i-1}, \\
&\boldsymbol{z}^{i} = \mathrm{diag}(\boldsymbol{\alpha}) \widetilde{\boldsymbol{z}}^{i-1}, \\
&\boldsymbol{z}^{i+1} = \boldsymbol{W} \boldsymbol{z}^{i} + \boldsymbol{b}.
\end{aligned} \tag{60}$$

We can merge the affine parameters in RMSNorm and the parameters in the following linear layer as follows:

$$\widetilde{\boldsymbol{W}} = \boldsymbol{W} \mathrm{diag}(\boldsymbol{\alpha}). \tag{61}$$

Then the forward pass at a merged RMSNorm and the following linear layer becomes as follows:

$$\begin{aligned}
&\text{Suppose } \boldsymbol{z}^{i-1} \in \mathbb{R}^{p_{i-1}}, \\
&\sigma = \sqrt{p_{i-1}^{-1} \boldsymbol{z}^{i-1\top} \boldsymbol{z}^{i-1} + \varepsilon}, \\
&\boldsymbol{z}^{i} = \sigma^{-1} \boldsymbol{z}^{i-1}, \\
&\boldsymbol{z}^{i+1} = \widetilde{\boldsymbol{W}} \boldsymbol{z}^{i} + \boldsymbol{b}.
\end{aligned} \tag{62}$$

The program of our MS-RMSNorm is shown in Algorithm 3.

H. Implementation Details of Fine-Tuning ViT, LLaMA and RoBERTa in Our Experiments

For experiments on fine-tuning ViT-base and ViT-large with LoRA and LoRA-FA, we use slight data augmentations in our experiments, which are Resize (to 224×224 px), RandomCrop, RandomHorizontalFlip, Normalize for the train set and Resize (to 224×224 px), CenterCrop, Normalize for the test set. We use AdamW (Loshchilov & Hutter, 2017) with the weight decay 0.1 in all our experiments on ViTs. The batch size is set as 64. All ViT models are fine-tuned with WarmUp in the first 10 epochs, where the initial learning rate starts from 1e-6, and Cosine learning rate scheduler in the remaining 90 epochs. The base learning rate is 1.25e-3 in LoRA and 1.25e-5 in Full Tuning. ViT-base experiments are conducted with $1\times$2080Ti GPU and ViT-large experiments are conducted with $1\times$L40 GPU. We use automatic mixed precision (AMP) in Pytorch as the default setting.

For experiments on fine-tuning LLaMA-7B and LLaMA-13B with QLoRA, the batch size is set as 4 and the number of gradient accumulation steps is set as 4. The total training iterations are 10000 steps. For LLaMA-7B, we use paged AdamW with no weight decay, tune constant learning rate in $\{10^{-4}, 2\times10^{-4}\}$, and report the best 5-shot MMLU accuracy among them. For LLaMA-13B, we tune learning rate in $\{10^{-4}, 2\times10^{-4}\}$, while setting weight decay as 0 for $\{$SiLU, RMSNorm$\}$ and $\{$ReSiLU2, RMSNorm$\}$ configurations. We set learning rate as 1e-4, while tuning weight decay in $\{0.1, 0.2\}$ for $\{$SiLU, MS-RMSNorm$\}$ and $\{$ReSiLU2, MS-RMSNorm$\}$ configurations. Gradient checkpointing (Chen et al., 2016) is not used in our experiments.

For experiments on fine-tuning RoBERTa-base with LoRA, the batch size is set as 32. We use AdamW with the weight decay 0.01. All RoBERTa-base models are fine-tuned from the pretrained model independently for 30 epochs. We use Linear learning rate scheduler with WarmUp ratio 0.1. The base learning rate for each task is chosen as the best one among $\{0.00005, 0.0001, 0.0005, 0.001, 0.005\}$ in fine-tuning the baseline.

I. Choice of Optimization Objective for Approximate Activation Function $\widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x)$

In Section 4.2, we derive the optimization objective (14) from our Approx-BP theory. Meanwhile, we believe that there exist other feasible choices of optimization objective. A heuristic choice can be,

$$\min_{\boldsymbol{a},\boldsymbol{c}} \int_{-\infty}^{\infty} (\mathrm{d}h(x) - \mathrm{d}\widetilde{h}_{\boldsymbol{a},\boldsymbol{c}}(x))^2 \mathrm{d}x. \tag{63}$$

Applying the similar technique introduced in Appendix E to the above optimization problem (63), we obtain another alternative of GELU. We call this alternative as ReGELU2-d, which means ReGELU2-d directly approximates the derivatives of GELU. The according solution of (63) for $\{\boldsymbol{a}, \boldsymbol{c}\}$ is:

$$\boldsymbol{a}^* = [0.32465931184406527, 0.34812875668739607]^\top,$$
$$\boldsymbol{c}^* = [-0.4535743722857079, -0.0010587205574873046, 0.4487575313884231]^\top.$$

In our experiments (Table 6), the fine-tuning ViT-base using LoRA with the new alternative ReGELU-d is also stable, but the results by ReGELU2-d are consistently inferior to those by our ReGELU2. Therefore, we still employ ReGELU2 and ReSiLU2 in our main paper.

Table 6. **Results of fine-tuning ViT-base using LoRA with different activation functions** on the CIFAR10 (C10), CIFAR100 (C100), and FGVC benchmarks. We report the Top-1 accuracy (%) results on each dataset and the mean Top-1 accuracy (%) results on all seven datasets. The best results are highlighted in **bold**.

Method	Activation	Norm	C10	C100	CUB	NAB	Flower	Dogs	Cars	Mean
LoRA $r=4$ Q, V	GELU	LN	**98.8**	**92.0**	86.7	**83.2**	**99.3**	90.7	**81.5**	**90.3**
	ReGELU2-d	LN	98.7	**92.0**	86.8	82.9	**99.3**	90.8	81.1	90.2
	ReGELU2	LN	**98.8**	**92.0**	**86.9**	83.0	**99.3**	**91.0**	81.4	**90.3**
LoRA $r=4$ All Linear	GELU	LN	**98.9**	**93.0**	**87.3**	**83.0**	**99.2**	90.7	82.9	90.7
	ReGELU2-d	LN	**98.9**	92.7	87.2	82.6	**99.2**	91.0	82.6	90.6
	ReGELU2	LN	**98.9**	92.8	**87.3**	**83.0**	**99.2**	**91.1**	**83.2**	**90.8**

J. More Experiments Results

In this section, some experimental results are supplementary to the main text, while others provide more diverse evaluations of our method.

J.1. Experiments on ViT

The results in Table 7 are supplementary to those in Table 1. Here, we report the results of replacing the activation function of the pretrained ViT-base with ReLU as a reference. The training throughput of GELU, ReLU, and ReGELU2 is similar, while the training performance of ReLU is significantly inferior to other activation functions in the comparison. When all linear layers are adapted by LoRA, the reduction of GPU memory usage during fine-tuning is similar between ReLU and ReGELU2. When only the query and value projections are adapted, ReLU can not reduce the GPU memory usage, whereas ReGELU2 can reduce the GPU memory usage by $\sim 19\%$. That indicates that ReLU is probably implemented in Pytorch in a manner as we described in Appendix F.

J.2. Experiments on LLaMA

As a supplementary material to Table 3, we report the BoolQ, PIQA, HS, WG, ARC-e, ARC-c, and OBQA metrics on fine-tuned LLaMA-7B in Table 8. We observe that the released checkpoint by the authors of QLoRA does not achieve much better results than the pretrained (without fine-tuning) LLaMA checkpoint. Thus, we speculate that these metrics in Table 8 are not suitable to serve as the evaluation metrics for fine-tuning LLaMA-7b on Alpaca dataset. However, our method still gets comparable performance on these metrics to the baseline.

We also have evaluated the max affordable training sequence length of LLaMA-7B with QLoRA on single RTX4090, which is summarized as Table 9. Our method can increase the max affordable training sequence length by $\sim 46\%$.

J.3. Experiments on SwinTransformer

We fine-tune the pretrained SwinTransformer-Tiny (Swin-T) and SwinTransformer-Small (Swin-S) (Liu et al., 2021b) with the detection head RetinaNet (Lin et al., 2017) on the PASCAL VOC object detection benchmark (Everingham et al., 2015). This experiment is conducted by data parallel training using 4×RTX2080Ti. The reported peak memory usage is the max value of those from the 4 GPUs. We use the training sets from VOC2007 and VOC2012 as the training set and the test set

Table 7. **Results of fine-tuning ViT-base using LoRA or LoRA-FA with different activation function and layer normalization** on the CIFAR10 (C10), CIFAR100 (C100), and FGVC benchmarks. We report the Top-1 accuracy (%) results on each dataset and the mean Top-1 accuracy (%) results on all seven datasets. The best results are highlighted in **bold**.

Method	Activation	Norm	Dataset							Mean		
			C10	C100	CUB	NAB	Flower	Dogs	Cars	Top-1(%)	Mem.(MiB)	Thr.(images/s)
LoRA $r=4$ Q, V	GELU	LN	**98.8**	**92.0**	86.7	**83.2**	**99.3**	90.7	**81.5**	**90.3**	3827	288
	ReLU	LN	98.4	90.4	85.5	81.8	97.4	88.4	80.7	89.0	3828(+0%)	**290**(+1%)
	Mesa-GELU	LN	**98.8**	**92.0**	86.6	83.1	**99.3**	90.8	81.1	**90.3**	3453(-10%)	245(-15%)
	ReGELU2	LN	**98.8**	**92.0**	**86.9**	83.0	**99.3**	**91.0**	81.4	**90.3**	**3087**(-19%)	289(+0%)
	GELU	Mesa-LN	**98.8**	91.8	86.8	**82.9**	**99.2**	90.8	81.3	90.2	**3249**(-15%)	257(-11%)
	GELU	MS-LN	**98.8**	**92.3**	**88.1**	82.7	**99.2**	**90.9**	**83.1**	**90.7**	3441(-10%)	**288**(+0%)
	Mesa-GELU	Mesa-LN	**98.8**	92.1	86.7	**83.0**	**99.3**	**90.9**	**82.1**	90.4	2853(-25%)	226(-22%)
	ReGELU2	MS-LN	**98.8**	**92.3**	**88.0**	82.6	99.2	90.8	**82.1**	**90.5**	**2717**(-29%)	**290**(+1%)
LoRA $r=4$ All Linear	GELU	LN	**98.9**	**93.0**	**87.3**	**83.0**	**99.2**	90.7	82.9	90.7	5128	207
	ReLU	LN	98.8	92.0	86.0	81.8	97.0	89.0	82.1	89.5	**4300**(-16%)	**208**(+0%)
	Mesa-GELU	LN	**98.9**	92.9	**87.3**	82.8	99.0	**91.2**	**83.3**	**90.8**	4721(-8%)	186(-10%)
	ReGELU2	LN	**98.9**	92.8	**87.3**	**83.0**	**99.2**	91.1	83.2	**90.8**	4380(-15%)	207(+0%)
	GELU	Mesa-LN	99.0	92.8	87.4	**83.0**	**99.3**	**90.8**	83.2	90.8	4530(-12%)	189(-9%)
	GELU	MS-LN	**99.1**	**93.0**	**88.5**	82.9	99.2	**90.8**	**85.0**	**91.2**	**4316**(-16%)	**207**(+0%)
	Mesa-GELU	Mesa-LN	98.9	92.9	87.2	82.9	99.3	**91.1**	83.2	90.8	4209(-18%)	173(-17%)
	ReGELU2	MS-LN	**99.0**	**93.1**	**88.0**	**83.1**	**99.4**	90.9	**84.8**	**91.2**	**3601**(-30%)	**208**(+0%)
LoRA-FA $r=4$ Q, V	GELU	LN	**98.4**	91.7	**88.5**	82.8	**99.1**	91.8	**77.6**	**90.0**	3386	304
	Mesa-GELU	LN	**98.4**	**91.9**	88.1	82.8	**99.1**	91.6	**77.6**	89.9	3012(-11%)	261(-14%)
	Mesa-GELU	Mesa-LN	98.3	91.4	88.2	**83.0**	**99.1**	91.7	77.5	89.9	**2411**(-29%)	236(-22%)
	ReGELU2	LN	**98.4**	91.7	88.1	82.6	**99.1**	**91.9**	77.2	89.8	2597(-23%)	**306**(+1%)
LoRA-FA $r=4$ All Linear	GELU	LN	98.6	91.5	**88.1**	**83.1**	99.2	91.8	79.3	**90.2**	3430	249
	Mesa-GELU	LN	98.6	**91.8**	88.0	82.9	99.2	**91.9**	79.0	**90.2**	3021(-12%)	218(-12%)
	Mesa-GELU	Mesa-LN	**98.7**	91.6	87.8	82.7	**99.3**	91.8	79.2	90.1	**2457**(-28%)	200(-20%)
	ReGELU2	LN	98.6	91.7	88.0	82.9	99.1	91.8	**79.4**	**90.2**	2717(-21%)	**251**(+0%)

Table 8. **Supplementary results on fine-tuning LLaMA-7B using QLoRA on Alpaca**. The metrics are evaluated by "lm-evaluation-harness" package (Gao et al., 2023). The best results are highlighted in **bold**.

Method	Checkpoint	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA
QLoRA $r=64$ All Linear	pretrained	74.43	78.45	32.91	75.00	**70.09**	**71.25**	44.54	44.80
	officially released	72.02	78.73	32.65	**76.05**	69.61	68.90	46.42	43.20
	fine-tuned by us	**74.50**	78.02	33.06	75.93	67.48	68.94	46.33	45.00
	with ReSiLU2 and MS-RMSNorm	73.76	**79.54**	**33.21**	75.82	68.43	69.53	**47.18**	**45.60**

Table 9. **Max affordable sequence length on fine-tuning LLaMA-7B using QLoRA**. Batch size is set as 1. The best results are highlighted in **bold**.

Method	Activation	Norm	Max Length of Tokens
QLoRA $r=64$ All Linear	SiLU	RMSNorm	1354
	ReSiLU2	RMSNorm	1504(+11%)
	SiLU	MS-RMSNorm	1654(+22%)
	ReSiLU2	MS-RMSNorm	**1979**(+46%)

from VOC2007 as the test set. The number of training epochs is set as 12. The data type in this experiment is fp32. The results are summarized in Table 10. One can see that our method reduces $\sim 18\%$ of the total memory consumption on fine-tuning Swin-T and Swin-S.

Table 10. **Results of fine-tuning SwinTransformer-tiny (Swin-T) and SwinTransformer-small (Swin-S)** with the detection head RetinaNet on the PASCAL VOC object detection benchmark. The best results are highlighted in **bold**.

Head	Backbone	Batch Size	Activation	Norm	Mem.(MiB)	Min/Epoch	mAP	AP50
RetinaNet	Swin-T	4	GELU	LN	7026	29.7	**79.37**	**79.40**
			ReGELU2	MS-LN	**5756**(-18%)	**29.2(-2%)**	79.20	79.20
	Swin-S	2	GELU	LN	5810	52.2	**80.78**	**80.80**
			ReGELU2	MS-LN	**4773**(-18%)	**50.5**(-3%)	80.45	80.40

J.4. Experiments on BERT

We fine-tune pretrained Bert-base (Devlin et al., 2018) on Squad-v2 (Rajpurkar et al., 2018) benchmark using data parallel training by 4×RTX3060. The number of training epochs is 2. The data type in this experiment is fp32. The results are summarized in Table 11. Our method enables to increase the batch size by 20%.

It is worth noting that increasing batch size usually enables less communication times, and thus larger throughput, in the distributed training. To demonstrate that, we fine-tune pretrained Bert-large on Squad-v2 under the ZeRO training framework (Rasley et al., 2020; Rajbhandari et al., 2020; 2021) using 4×RTX3060. As shown in Table 12, our method can increase the throughput by $\sim 26\%$.

Table 11. **Results of fine-tuning Bert-base** on Squad-v2 using 4×RTX3060 GPUs. We set the batch size to the max affordable size. The batch size in the table is the batch size per GPU. The best results are highlighted in **bold**.

Model	Activation	Norm	Batch Size	Thr.(samples/s)	EM	F1
Bert-base	GELU	LN	30	76	70.94	74.14
	ReGELU2	MS-LN	36	**78**(+3%)	**71.36**	**74.63**

Table 12. **Results of fine-tuning Bert-large** on Squad-v2 using 4×RTX3060 GPUs. We set the batch size to the max affordable size. The batch size in the table is the batch size per GPU. The best results are highlighted in **bold**.

Model	ZeRO	Activation	Norm	Batch Size	Thr.(samples/s)	Hour/Epoch	EM	F1
Bert-large	Stage 3 + CPU offload	GELU	LN	10	9.57	3.83	**77.29**	**80.65**
		ReGELU2	MS-LN	14	**12.03**(+26%)	**3.05**(-20%)	77.19	80.59