Initialization of Large Language Models via Reparameterization to Mitigate Loss Spikes

Kosuke Nishida **Kyosuke Nishida** **Kuniko Saito**
NTT Human Informatics Laboratories, NTT Corporation
{kosuke.nishida, kyosuke.nishida, kuniko.saito}@ntt.com

Abstract

Loss spikes, a phenomenon in which the loss value diverges suddenly, is a fundamental issue in the pre-training of large language models. This paper supposes that the non-uniformity of the norm of the parameters is one of the causes of loss spikes. Here, in training of neural networks, the scale of the gradients is required to be kept constant throughout the layers to avoid the vanishing and exploding gradients problem. However, to meet these requirements in the Transformer model, the norm of the model parameters must be non-uniform, and thus, parameters whose norm is smaller are more sensitive to the parameter update. To address this issue, we propose a novel technique, weight scaling as reparameterization (WeSaR). WeSaR introduces a gate parameter per parameter matrix and adjusts it to the value satisfying the requirements. Because of the gate parameter, WeSaR sets the norm of the original parameters uniformly, which results in stable training. Experimental results with the Transformer decoders consisting of 130 million, 1.3 billion, and 13 billion parameters showed that WeSaR stabilizes and accelerates training and that it outperformed compared methods including popular initialization methods.

1 Introduction

Transformer-based large language models (LLMs) have attracted remarkable attention (Vaswani et al., 2017; Brown et al., 2020). The discovery of a scaling-law (Kaplan et al., 2020) has been driving the model and corpus sizes ever larger, causing huge computational costs for pre-training. During pre-training of LLMs, the loss value often diverges suddenly (Chowdhery et al., 2023; Zhang et al., 2022), as illustrated at the top of Figure 1. This phenomenon, known as loss spikes, is a fundamental issue in the LLM pre-training because it not only increases the final loss value, but also causes the pre-training to fail if the loss diverges completely.

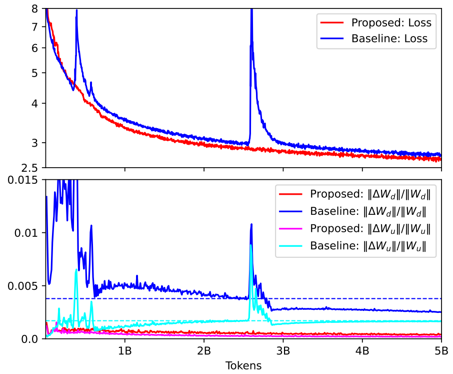

Figure 1: Loss of Transformer models with 13 billion (13B) parameters at the beginning of training (top). Update ratios for the up and down projection in the last feed-forward layer, $\|\Delta \boldsymbol{W}_u\|/\|\boldsymbol{W}_u\|$ and $\|\Delta \boldsymbol{W}_d\|/\|\boldsymbol{W}_d\|$, of the same (bottom). The horizontal lines are the update ratios before the largest spike. The baseline sets $\|\boldsymbol{W}_d\|$ smaller than the other parameters. The update ratio of $\boldsymbol{W}_d$ is larger at the very beginning and gets smaller after loss spikes occur. The baseline uses standard techniques for stable training, such as gradient clipping.

Here, let $\Delta \boldsymbol{W}$ be the update of the parameter $\boldsymbol{W}$ at an optimization step. $\|\Delta \boldsymbol{W}\|/\|\boldsymbol{W}\|$ represents the magnitude of the parameter update relative to the parameter itself, and we call it the *update ratio*. The bottom of Figure 1 shows the update ratios. We consider that different scales of update ratios among parameter matrices can lead to unstable training. Indeed, before the loss spike, the update ratio of $\boldsymbol{W}_d$ is larger than that of $\boldsymbol{W}_u$. That is, $\boldsymbol{W}_d$ undergoes a more pronounced change. After the spike, the difference between the update ratios decreases. This observation motivated us to regulate the update ratios in the model in a certain range.

We consider that uneven and large update ratios are due to non-uniformity of the norm of the parameters. With the current initialization methods, $\boldsymbol{W}_d$ is set smaller than other parameters, which is required to avoid the vanishing and exploding gra-

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 22699–22714
November 12-16, 2024 ©2024 Association for Computational Linguistics

dients problem. Consequently, by definition, the update ratio of $\boldsymbol{W}_d$ tends to be larger.

To address this issue, we propose a novel technique, called weight scaling as reparameterization (WeSaR). WeSaR introduces a gate parameter $\alpha \in \mathbb{R}$ for each parameter matrix $\boldsymbol{W}$ and uses $\alpha\boldsymbol{W}$ instead of $\boldsymbol{W}$ inside the model. WeSaR relieves the parameter $\boldsymbol{W}$ of non-uniformity by adjusting α to the values required to avoid the vanishing and exploding gradients problem. Moreover, WeSaR enables an arbitrary small common standard deviation to set be for all parameters, which results in not only stable, but also accelerated, training.

We conducted pre-training of Transformer decoders consisting of 130 million (13M), 1.3 billion (1.3B), and 13B parameters. Our experimental results show that WeSaR stabilized and accelerated their training due to the stable and equal-scale update ratios, as shown in Figure 1. We also confirmed that WeSaR outperformed compared methods, including a initialization method widely used for pre-training LLMs (Nguyen and Salazar, 2019) and the existing reparameterization methods (Salimans and Kingma, 2016; Zhai et al., 2023; Noci et al., 2022).

Our contributions can be summarized as follows:

- We clarify one of the causes of loss spikes, *i.e.,* the non-uniformity of parameters that arises to meet the requirements for avoiding the vanishing and exploding gradients problem.
- We address the non-uniformity problem by reparameterizing the parameter as $\alpha\boldsymbol{W}$ with a gate parameter α. α determines the scale of $\alpha\boldsymbol{W}$. $\boldsymbol{W}$ is initialized with a small common standard deviation throughout the model.
- Experimental results show that the proposed method stabilizes and accelerates training. It outperformed compared methods, including a popular initialization method of LLMs.

2 Preliminaries

We consider Transformer models (Vaswani et al., 2017) consisting of the following layers: an embedding layer with $\boldsymbol{W}_e$, self-attention layers with $\boldsymbol{W}_q$, $\boldsymbol{W}_k$, $\boldsymbol{W}_v$, and $\boldsymbol{W}_o$ (query, key, value, and output projections), feed-forward layers with $\boldsymbol{W}_u$ and $\boldsymbol{W}_d$ (up and down projections)[1], and a prediction layer with $\boldsymbol{W}_p$. Each parameter $\boldsymbol{W}_\cdot$ is initialized according to a Gaussian distribution $\mathcal{N}(0, \sigma_\cdot^2)$.

The input first passes through the embedding layer; then it is processed by N Transformer blocks, which consist of self-attention layers and feed-forward layers. The transformation f of the self-attention layer and the feed-forward layer with a residual connection can be written as

$$\boldsymbol{y} = f(\mathrm{LN}(\boldsymbol{x})) + \boldsymbol{x}, \tag{1}$$

where LN indicates a layer normalization (Ba et al., 2016) that is applied after the residual connection, called the Pre-LN type (Liu et al., 2020).

In this section, we first review the back-propagation algorithm (Rumelhart et al., 1986). Then, we describe the initialization strategies of the Transformer models to avoid the vanishing and exploding gradients problem.

2.1 Back-Propagation

Back-propagation passes the gradients of the loss function from the top layer to the bottom layer through the network. Here, to avoid the vanishing and exploding gradients problem in deep neural networks, the scale of the gradients must be kept constant throughout the model. Let us consider a layer $\boldsymbol{y} = g(\boldsymbol{x}) \quad (\boldsymbol{y} \in \mathbb{R}^{d_{\text{out}}}, \boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}})$. $\mathcal{L}$ denotes the loss, and $\boldsymbol{\delta} \in \mathbb{R}^{d_{\text{out}}}$ denotes the gradient of the loss with respect to the output $\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}}$. To keep the scale of the gradients before and after the layer, a layer g must satisfy the condition,

$$E\left[\left\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}\right\|^2\right] = E\left[\left\|\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\boldsymbol{\delta}\right\|^2\right] = E\left[\|\boldsymbol{\delta}\|^2\right]. \tag{2}$$

Back-propagation is a chain of differentiation. Therefore, the scale of the gradients in the entire model is maintained when each layer in the model meets this requirement.

2.2 Initialization Strategies of Transformer

Embedding Scaling. σ_e plays an essential role in back-propagation through the Transformer layers (Takase et al., 2023). Here, we use the RMSNorm $\boldsymbol{y} = \gamma_{\text{LN}} \odot \frac{\sqrt{d}\boldsymbol{x}}{\sqrt{\|\boldsymbol{x}\|^2}}$ (Zhang and Sennrich, 2019) as the layer normalization, where γ_{LN} is a parameter, d is the number of dimensions, and $\odot$ indicates the Hadamard product. Back-propagation through RMSNorm is

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \sqrt{\frac{d}{\|\boldsymbol{x}\|^2}}\left(\boldsymbol{I} - \frac{\boldsymbol{x}\boldsymbol{x}\top}{\|\boldsymbol{x}\|^2}\right)\mathrm{diag}(\gamma_{\text{LN}}),$$

[1] We did not use GLU (Shazeer, 2020) for simplicity.

where $\mathrm{diag}(\cdot)$ is a diagonal matrix and $\boldsymbol{I}$ is an identity matrix. Because $\sqrt{\frac{d}{\|\boldsymbol{x}\|^2}}$ is the inverse of the standard deviation of $\boldsymbol{x}$ if the mean of $\boldsymbol{x}$ is zero, the standard deviation of $\boldsymbol{x}$ affects the norm of the gradients. The standard deviation of the embedding matrix σ_e influences the standard deviation of the input in RMSNorm through the residual connections (Equation 1). Thus, to avoid the vanishing and exploding gradients problem, σ_e should be set to 1.

On the basis of the above discussion, Takase et al. (2023) presented two previous studies achieving a standard deviation of 1 for $\boldsymbol{x}$ without directly setting $\sigma_e = 1$. The first way multiplies the output of the embedding layer by a constant $1/\sigma_e$. This technique was introduced in the original Transformer (Vaswani et al., 2017) but was deleted from the implementations. The second way adds the layer normalization to the top of the embedding layer (Le Scao et al., 2022).

Residual Scaling. σ_o and σ_d are also important factors for stable training. The residual scaling technique was introduced to Transformer by GPT-2 (Radford et al., 2019) without explanation. Here, we present a theoretical analysis (Taki, 2017) originally designed for ResNet (He et al., 2016) while modifying it for Transformer. The analysis in a formal form is presented in Appendix A.

The back-propagation through Equation 1 is

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \boldsymbol{\delta}\left(\frac{\partial f(\mathrm{LN}(\boldsymbol{x}))}{\partial \boldsymbol{x}} + \boldsymbol{I}\right). \quad (3)$$

Let s^2 be $E\left[\left\|\frac{\partial f(\mathrm{LN}(\boldsymbol{x}))_i}{\partial \boldsymbol{x}}\right\|^2\right]$. Thus, a residual connection causes an (s^2+1)-fold increase in the squared norm of the gradient $E\left[\left\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}\right\|^2\right]$. As a result, the gradient explodes exponentially with respect to the depth of layers throughout the propagation. This exponential increase is unacceptable for LLMs consisting of many Transformer blocks.

To alleviate this problem, the residual scaling multiplies σ_o and σ_d by $\frac{1}{\sqrt{2N}}$ since the model has $2N$ residual connections. This multiplication achieves $E[s^2] = \mathcal{O}\left(\frac{1}{2N}\right)$, and the scale of the exploding gradient $(s^2+1)^{2N}$ converges to Napier's constant e in the limit $N \to \infty$. This avoids an exponential explosion with respect to N.

3 Existing Methods and Their Problems

Here, we review two of the existing initialization methods and their problems. The methods are summarized in Table 1.

3.1 He Initialization

He initialization (He et al., 2015) is one of the most popular initialization methods for neural networks. It is designed to keep the scale of the gradients constant throughout the network to meet the requirement of Equation 2. In the case of a linear layer $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} \quad (\boldsymbol{y} \in \mathbb{R}^{d_{\mathrm{out}}}, \boldsymbol{x} \in \mathbb{R}^{d_{\mathrm{in}}}, \boldsymbol{W} \in \mathbb{R}^{d_{\mathrm{out}} \times d_{\mathrm{in}}})$, the requirements can be written as

$$E\left[\left\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}\right\|^2\right] = E\left[\left\|\boldsymbol{W}^\top \boldsymbol{\delta}\right\|^2\right] = \mathrm{Var}\left[\left\|\boldsymbol{W}^\top \boldsymbol{\delta}\right\|\right]$$

$$= d_{\mathrm{in}} \mathrm{Var}\left[\boldsymbol{W}\right] E\left[\|\boldsymbol{\delta}\|^2\right] = E\left[\|\boldsymbol{\delta}\|^2\right].$$

Thus, the parameter $\boldsymbol{W} \in \mathbb{R}^{d_{\mathrm{out}} \times d_{\mathrm{in}}}$ must be initialized with the standard deviation $\sigma = \frac{1}{\sqrt{d_{\mathrm{in}}}}$. Note that the numerator, called the gain, is determined depending on the activation function. We assume the identity function in the above discussion for simplicity. For ReLU activation, the gain is $\sqrt{2}$.

3.2 Small Initialization

Small initialization (Nguyen and Salazar, 2019) is based on empirical findings that a small standard deviation leads to stable training. It sets a common small standard deviation $\sqrt{\frac{2}{5d}}$ for all parameters except for the $1/\sqrt{2N}$ scaling of σ_o and σ_d. Here, we should note that $\sqrt{\frac{2}{5d}}$ is the standard deviation which Xavier initialization (Glorot and Bengio, 2010) specifies for $\boldsymbol{W}_u$ and $\boldsymbol{W}_d$, and it is the smallest standard deviation among all of the parameters in the Transformer layers.

3.3 Problems

Although the He and Small initializations with the embedding and residual scaling stabilize the training, they often cause loss spikes, as shown at the top of Figure 1. Deep neural networks are designed to keep the scale of the gradients constant throughout the model. Therefore, in the parameters whose norm is smaller than that of the others, the update ratios $\|\Delta \boldsymbol{W}\|/\|\boldsymbol{W}\|$ are larger. Because the update ratio indicates the magnitude of the effect of the update on the parameter, parameters with large update ratios are fragile.

	He		Small		WeSaR	
	Gate	Weight	Gate	Weight	Gate	Weight
$\boldsymbol{W}_e$	1	$\sqrt{\frac{1}{d}}$	1	$\sqrt{\frac{2}{5d}}$	1	σ
$\boldsymbol{W}_k$	N/A	$\sqrt{\frac{1}{d}}$	N/A	$\sqrt{\frac{2}{5d}}$	$\sqrt{\frac{1}{d}}$	σ
$\boldsymbol{W}_q$	N/A	$\sqrt{\frac{1}{d}}$	N/A	$\sqrt{\frac{2}{5d}}$	$\sqrt{\frac{1}{d}}$	σ
$\boldsymbol{W}_v$	N/A	$\sqrt{\frac{1}{d}}$	N/A	$\sqrt{\frac{2}{5d}}$	$\sqrt{\frac{1}{d}}$	σ
$\boldsymbol{W}_o$	N/A	$\sqrt{\frac{1}{2Nd}}$	N/A	$\sqrt{\frac{2}{10Nd}}$	$\sqrt{\frac{1}{2Nd}}$	σ
$\boldsymbol{W}_u$	N/A	$\sqrt{\frac{1}{d}}$	N/A	$\sqrt{\frac{2}{5d}}$	$\sqrt{\frac{1}{d}}$	σ
$\boldsymbol{W}_d$	N/A	$\sqrt{\frac{2}{8Nd}}$	N/A	$\sqrt{\frac{2}{10Nd}}$	$\sqrt{\frac{2}{8Nd}}$	σ
$\boldsymbol{W}_p$	N/A	$\sqrt{\frac{1}{d}}$	N/A	$\sqrt{\frac{2}{5d}}$	$\sqrt{\frac{1}{d}}$	σ

Table 1: Standard deviations of initialization methods before and after the gate[2]. We assume that He and Small initializations use embedding scaling (Vaswani et al., 2017; Takase et al., 2023). The proposed method initializes all parameters with a common σ. We adopt the popular setting where d_{out} of $\boldsymbol{W}_u$ and d_{in} of $\boldsymbol{W}_o$ are $4d$ and d_{in} and d_{out} of the other parameters are d.

The bottom of Figure 1 shows the update ratios in the last feed-forward layer: $\|\Delta\boldsymbol{W}_d\|/\|\boldsymbol{W}_d\|$ and $\|\Delta\boldsymbol{W}_u\|/\|\boldsymbol{W}_u\|$. The update ratio of $\boldsymbol{W}_d$ is larger than that of $\boldsymbol{W}_u$ because the residual scaling multiplies $\|\boldsymbol{W}_d\|$ by $1/\sqrt{2N}$ (in the 13B model, $1/\sqrt{2N} \approx 0.11$). The update ratio of $\boldsymbol{W}_d$ is especially large at the very beginning. After the pre-training on 1B tokens with some loss spikes, it stays within a certain range. However, it is still much larger than that of $\boldsymbol{W}_u$. After the largest loss spike occurs, the update ratio of $\boldsymbol{W}_d$ gets closer to that of $\boldsymbol{W}_u$. Therefore, we consider that uneven and large update ratios can cause loss spikes, and we can mitigate loss spikes by regulating them.

4 Proposed Method

We propose WeSaR as a way to meet the two conflicting aforementioned requirements: (i) the criteria of any initialization method designed to avoid the vanishing and exploding gradients problem, as discussed in §2.2, and (ii) the common scales of all parameters to keep stable and uniform update ratios for mitigating loss spikes, as discussed in §3.3. In addition to stabilizing the training, WeSaR enables a hyperparameter setting that achieves a rapid decrease in loss.

4.1 Initialization via Reparameterization

We consider a situation where the parameter $\boldsymbol{W}_.$ is initialized according to $\mathcal{N}(0, \sigma_.^2)$. Here, the proposed method initializes $\boldsymbol{W}_.$ by using a common standard deviation σ among all parameters and uses $\bar{\boldsymbol{W}}_.$ instead of the original $\boldsymbol{W}_.$ inside the model,

$$\boldsymbol{W}_. \sim \mathcal{N}(0, \sigma^2)$$

$$\bar{\boldsymbol{W}}_. = \frac{\sigma_.}{\sigma}\boldsymbol{W}_. = \alpha_.\boldsymbol{W}_.,$$

where σ is a hyperparameter, and $\alpha_., \boldsymbol{W}_.$ are trainable parameters. The gate parameter $\alpha_.$ is initialized to $\frac{\sigma_.}{\sigma}$. We call $\boldsymbol{W}_.$ an actual parameter and $\bar{\boldsymbol{W}}_. = \alpha_.\boldsymbol{W}_.$ a virtual parameter.

Beyond introducing the gate parameters to all parameter matrices, WaSAR is designed to initialize the actual parameters with uniform standard deviations σ while aligning the standard deviations of the virtual parameter $\sigma_.$ to the criteria of the initialization methods by adjusting the gate parameter $\alpha_.$. Therefore, WeSaR eliminates the non-uniformity of $\|\boldsymbol{W}_.\|$ and $\|\Delta\boldsymbol{W}_.\|/\|\boldsymbol{W}_.\|$. The effect of WeSaR is shown at the bottom of Figure 1. Because $\boldsymbol{W}_d$ and $\boldsymbol{W}_u$ are initialized equally, their update ratios are comparable and stable during training.

Because just one trainable parameter is added to each parameter matrix $\boldsymbol{W}_. \in \mathrm{R}^{d_{\text{out}} \times d_{\text{in}}}$, WeSaR has little effect on the number of trainable parameters and the training cost. Moreover, it has no effect on the inference because the gate parameter can be merged after the training.

We can align the backbone initialization of WeSaR to any existing initialization methods. In this paper, we adopt He initialization $\frac{\text{gain}}{\sqrt{d_{\text{in}}}}$ with the embedding and residual scaling for the virtual parameter $\alpha_.\boldsymbol{W}_.$ to avoid gradient decay throughout the Transformer layers.

4.2 Theoretical Justification

Here, we explain that WeSaR does not affect the training dynamics of Transformer. We assume that the optimizer is Adam (Kingma and Ba, 2015) because of its benefits to Transformer (Zhang et al., 2020; Pan and Li, 2022; Zhang et al., 2024). Let us consider a parameter update $\Delta\boldsymbol{W}_t$ at step t. The update of Adam is

$$\Delta\boldsymbol{W}_t = \mu_t \frac{\boldsymbol{M}_t}{\sqrt{\boldsymbol{V}_t}}, \tag{4}$$

where $\boldsymbol{M}_t$ is the exponential moving average of the gradient $\frac{\partial\mathcal{L}}{\partial\boldsymbol{W}}$, $\boldsymbol{V}_t$ is that of the squared gradient, and μ_t is the learning rate.

[2] We approximate the gain of the activation function used in the feed-forward layer to that of ReLU (*i.e.*, $\sqrt{2}$).

Because of

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{\cdot}} = \frac{\partial \mathcal{L}}{\partial \bar{\boldsymbol{W}}_{\cdot}} \frac{\partial \bar{\boldsymbol{W}}_{\cdot}}{\partial \boldsymbol{W}_{\cdot}} = \frac{\sigma_{\cdot}}{\sigma} \frac{\partial \mathcal{L}}{\partial \bar{\boldsymbol{W}}_{\cdot}},$$

the gradient is multiplied by $\frac{\sigma_{\cdot}}{\sigma}$ through the gate. From the definition of Adam (Equation 4), the Adam states $\boldsymbol{M}_t$ and $\sqrt{\boldsymbol{V}_t}$ are multiplied by $\frac{\sigma_{\cdot}}{\sigma}$ equally, and thus the reparameterization does not affect the parameter update $\mu_t \frac{\boldsymbol{M}_t}{\sqrt{\boldsymbol{V}_t}}$. Therefore, the parameter update is independent of $\sigma_{\cdot}$ if we use Adam.

That is, WeSaR relieves the actual parameters and their update of the restriction with respect to $\sigma_{\cdot}$ that is specified in order to avoid the vanishing and exploding gradients problem. Secondarily, different from the existing methods that define the standard deviations as functions of d, we can determine the standard deviation of the actual parameters independently of d, because the gate α undertakes the dependence on d.

4.3 Hyperparameter Setting

Here, we explain the hyperparameter setting that enables a stable and rapid loss decrease. Different from conventional initialization methods, WeSaR can set the common standard deviation σ to an arbitrary value. In addition, the stability afforded by WeSaR enables us to set the learning rate and batch size to accelerate training.

Standard Deviation σ. In this paper, we set to $\sigma^2 =$ 4e-5, unless otherwise mentioned. This setup corresponds to $d = 10,000$ in the Small initialization criteria $\sqrt{\frac{2}{5d}}$. That is, our σ setup is smaller than those of conventional setups[3]. We can expect a rapid decrease in loss with the same learning rate because of the large parameter update $\Delta \boldsymbol{W}$ relative to the parameter $\boldsymbol{W}$ itself. Zhang et al. (2019a) confirmed the preference to a smaller standard deviation in the Transformer models, which justifies our setup.

Learning rate. Because WeSaR enables stable training, we can increase the learning rate from the conventional values (an order of 1e-4). Here, we set it to 1e-3.

Batch size. In the conventional pre-training of an LLM, the batch size is set to a large value (e.g., 4M tokens) to avoid loss spikes. We can decrease the batch size for a rapid loss decrease if the training is stable. However, the batch size has to be large enough in order to pre-train the model efficiently on large numbers of GPUs, as is commonly done when pre-training LLMs. Thus, we set the batch size to 1M tokens.

	130M	1.3B	13B
# Param.	134.1M	1,339.1M	12,911.0M
Hidden Size d	768	2048	5120
# Layer N	12	24	40
# Attention Head	12	16	40

Table 2: Model configuration.

	Rapid Setting	Stable Setting
Batch Size [tokens]	1M	4M
Learning rate μ	1e-3	5e-4
Warmup Steps	100	2000
Gradient Clipping Threshold	1	
Weight decay	0.01	
Z-loss	1e-4	

Table 3: Training configuration.

5 Experimental Evaluation

5.1 Experimental Setup

We pre-trained the 130M, 1.3B, and 13B models on the basis of the configuration listed in Table 2. The model architecture was based on LLaMA (Touvron et al., 2023), except for the feed-forward layer with gelu activation. Our experiments mainly focused on the 1.3B models. The training was based on the hyperparameters listed in Table 3. There were two settings for the learning rate, batch size, and warmup steps: One was a conventional setting emphasizing on a stable training; the other emphasized a rapid decrease in loss. We used perplexity as a metric. Appendix B describes the detailed configuration.

5.2 Dataset

We sampled 30B tokens from RefinedWeb (Penedo et al., 2023) and used them as the pre-training corpus. Hoffmann et al. (2022) found that the optimal pre-training corpus size is roughly 20 tokens per model parameter. Thus, 30B tokens were sufficient for our main experiments using 1.3B models. For the 13B models, we investigated the behavior in the first 1/10th of the training. For the evaluation, we used LAMBADA (Paperno et al.) and WikiText (Merity et al., 2017).

[3] Even in LLaMA3 70B, $d = 8192$ (AI@Meta, 2024).

Method	Weights	Train	Norm	Scale
Weight Normalization	all	✓	✓	by-row
σReparam	all	✓	✓	by-matrix
Residual Scaling	$\boldsymbol{W}_o$, $\boldsymbol{W}_d$			by-matrix
WeSaR	all	✓		by-matrix

Table 4: Comparison of reparameterization methods. “Weights” means the reparameterized weight matrices. "Train" means that each method uses trainable gate parameters. “Norm” means that each method uses reparameterization via weight-based normalization. “Scale” means the unit of scaling in the reparameterization.

5.3 Compared Models

As a baseline, we trained the model with the most popular method, *i.e.,* **Small initialization**.

In addition, we compared the proposed method with the three reparameterization methods listed in Table 4. Because all methods have their own motivation, we discuss the detailed difference in Appendix C. In short, the difference from the former two methods is efficiency because WeSaR does not conduct any normalization. From the last method, WeSaR reparameterizes all parameters and sets a common small value to the standard deviations of all parameters.

Weight Normalization. Weight Normalization (Salimans and Kingma, 2016) was proposed to decouple the length of the weight vectors from their direction. It conducts L2 normalization and scaling of each row of the parameter matrix $\boldsymbol{w} \in \mathbb{R}^{d_\text{in}}$ as $\bar{\boldsymbol{w}} = \frac{\alpha}{\|\boldsymbol{w}\|}\boldsymbol{w}$.

σReparam. σReparam (Zhai et al., 2023) was proposed to control the spectral norm (*i.e.,* the maximum singular value) of the parameter for stable Transformer training. It conducts spectral normalization (Miyato et al., 2018) and scaling of the parameter matrix $\boldsymbol{W} \in \mathrm{R}^{d_\text{out} \times d_\text{in}}$: $\bar{\boldsymbol{W}} = \frac{\alpha}{\|\boldsymbol{W}\|_2}\boldsymbol{W}$, where $\|\boldsymbol{W}\|_2$ is the spectral norm. The original σReparam adopts Post-LN; and we tried both Post-LN and the more popular Pre-LN.

Residual Scaling as Reparameterization. Noci et al. (2022) overcomes the limitation of the $(1/\sqrt{2N})$-fold multiplications of σ_o and σ_d caused by the residual connection (Equation 1). It modifies the residual connection to $\boldsymbol{y} = \frac{1}{\sqrt{2N}} f(\mathrm{LN}(\boldsymbol{x})) + \boldsymbol{x}$. Different from the original residual scaling, which changes the standard deviations, this equation can be viewed as a reparameterization of $\boldsymbol{W}_o$ and $\boldsymbol{W}_d$ because of its linearity.

		WikiText	LAMBADA
130M	Small Init. (Rapid)	26.57	33.56
	Small Init. (Stable)	37.68	40.41
	WeSaR	**25.07**	**31.89**
1.3B	Small Init. (Rapid)	16.55	26.29
	Small Init. (Stable)	21.44	28.81
	WeSaR	**14.51**	**22.87**
13B	Small Init. (Rapid)	12.72	21.79
	Small Init. (Stable)	18.66	25.34
	WeSaR	**12.05**	**21.57**

Table 5: Main results.

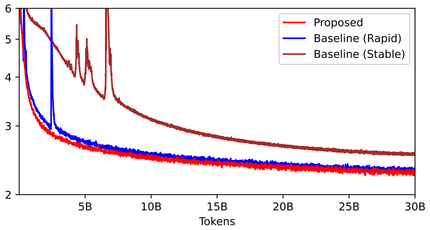

Figure 2: Loss of 13B models during training.

Setup. For Weight Normalization and σReparam, which reparameterize all parameters, we tuned σ^2 in $\{1, 4, 16, 64, 256\}$e-5 and set the initial α to the values defined by each method. Because residual scaling does not reparameterize all of the parameters and does not specify a backbone initialization method, we chose the He and Small initializations. All methods used embedding scaling because Takase et al. (2023) confirmed its benefit.

5.4 Results and Discussion

Main results. Table 5 shows the main results. WeSaR outperformed the widely used Small initialization. Figure 1 and 2 show the decrease in loss of the 13B models at the beginning of and over the whole training, respectively. We found that WeSaR achieved stable training, whereas Small initialization caused loss spikes. Moreover, under the hyperparameter setting that aimed to stabilize training, Small initialization still caused loss spikes and eventually had higher (*i.e.,* worse) perplexity due to the small learning rate and large batch size. As well, due to the lower learning rate, the stable setting took more steps until reaching stable states without loss spikes. Thus, we used the rapid hyperparameter setting in the following experiments. The loss decreases for the 130M and 1.3B models are shown in Appendix E.

	WikiText	LAMBADA	Time	# Param.	Best σ^2
Small Init.	20.64 (0.52)	29.50 (0.53)	18.88	1,339.1M	N/A
Weight Normalization	18.87 (0.59)	27.69 (0.86)	21.27 (+12.6%)	1,339.6M	16e-5
σReparam w./ Pre-LN	25.26 (1.65)	30.74 (0.74)	20.06 (+6.25%)	1,339.1M	64e-5
σReparam w./ Post-LN	23.64 (1.03)	30.56 (0.89)	20.09 (+6.39%)	1,339.1M	16e-5
Residual Scaling w./ He	23.15 (0.37)	31.03 (0.20)	19.19 (+1.66%)	1,339.1M	N/A
Residual Scaling w./ Small	23.56 (1.03)	30.78 (0.35)	19.18 (+1.58%)	1,339.1M	N/A
WeSaR	**17.74** (0.05)	**27.52** (0.28)	19.25 (+1.95%)	1,339.1M	4e-5

Table 6: Comparison of reparameterization methods in five runs based on 10B tokens. Mean and standard deviation are listed. The best method is in bold, and the methods within one standard deviation are underlined.

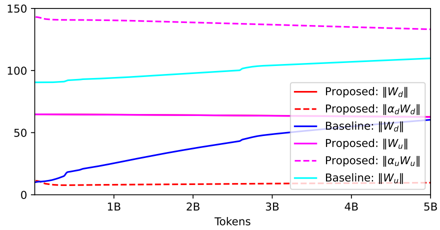

Figure 3: Norm of parameters $\|\boldsymbol{W}_d\|$ and $\|\boldsymbol{W}_u\|$ in the last layer at the beginning of the training. $\|\boldsymbol{W}_d\|$ and $\|\boldsymbol{W}_u\|$ of the proposed method overlap.

	WikiText	LAMBADA
Small Init.	16.55	26.29
He Init.	16.70	26.50
WeSaR (w./ He Init.)	**14.51**	**22.87**
w./ Small Init.	15.91	24.37
w./ fixed α	15.21	25.61

Table 7: Ablation studies.

Why does the reparameterization stabilize training? The bottom of Figure 1 shows that, during the training using Small initialization, $\|\Delta \boldsymbol{W}_d\|/\|\boldsymbol{W}_d\|$ was large at the very beginning of the training and became small and stable after the loss spikes occurred. However, the proposed method kept $\|\Delta \boldsymbol{W}_d\|/\|\boldsymbol{W}_d\|$ and $\|\Delta \boldsymbol{W}_u\|/\|\boldsymbol{W}_u\|$ in a certain range during the training, which led to stable training. The update ratios in other parameters are shown in Appendix F.

To investigate the reason for this remarkable difference, we analyzed the values of $\|\boldsymbol{W}_d\|$ and $\|\boldsymbol{W}_u\|$ in the last layer during training. As shown in Figure 3, $\|\boldsymbol{W}_d\|$ and $\|\boldsymbol{W}_u\|$ of Small initialization became larger during training because of the small initial values. To achieve such large change in $\boldsymbol{W}_d$ and $\boldsymbol{W}_u$, the parameter update should be also large enough. Therefore, the update ratios of Small initialization were larger and more unstable than those of WeSaR. A large update is especially harmful to $\boldsymbol{W}_d$ due to the non-uniformity, which causes the training to become unstable.

Although the virtual parameters $\alpha_d \boldsymbol{W}_d$ and $\alpha_u \boldsymbol{W}_u$ of WeSaR changed their norms during training, WeSaR assigned the role of changing the norm to the gate parameter α_d and α_u. Therefore, the norm of the actual parameters $\|\boldsymbol{W}_d\|$ and $\|\boldsymbol{W}_u\|$ did not change by much. This nearly constant scale of the actual parameters contributed to the stability.

Is the reparameterization effective? Table 7 shows the results of the ablation studies. Among the existing methods, Small initialization outperformed He initialization. He initialization also caused loss spikes. Thus, as Nguyen and Salazar (2019) confirmed, Small initialization is more suitable than He initialization for pre-training LLMs.

However, He initialization outperformed Small initialization as a backbone initialization method of WeSaR. We consider that He initialization is suitable for propagating the gradients to lower layers, although a small standard deviation (*e.g.,* Small initialization) is suitable as the parameter itself. The advantage of WeSaR is that it sets the standard deviations of the actual parameter to smaller values, while it sets the norm of the virtual parameter to a sufficient value for the back-propagation.

Also, in relation to discussed with Figure 3, the trainability of the gate parameter α contributes to the model performance.

Does WeSaR outperform the existing reparameterization methods? We compared WeSaR with the existing reparameterization methods, shown in Table 6. In pilot experiments, we confirmed that the pre-training on 10B tokens is sufficient to rank the methods. Thus, we conducted five runs of each method with 10B tokens and report the means and

Dataset	BoolQ	CB		COPA	MultiRC	ReCoRD		RTE	WiC	WSC	Total		
Metric	ACC	ACC	F1	ACC	ACC	EM	F1	ACC	ACC	ACC	ACC	EM	F1
Small Init.	$\underline{60.28}$	32.14	22.26	$\underline{73.00}$	46.12	73.23	73.93	$\underline{53.43}$	**50.00**	40.38	51.73	73.23	73.65
Weight Normalization	58.27	$\underline{42.86}$	25.13	$\underline{69.00}$	**57.32**	75.11	75.83	**57.76**	**50.00**	36.54	$\underline{57.18}$	75.11	75.55
σReparam w./ Pre-LN	$\underline{61.19}$	**48.21**	28.78	66.00	50.08	68.32	69.02	$\underline{52.71}$	**50.00**	$\underline{44.23}$	54.16	68.32	68.79
σReparam w./ Post-LN	57.65	$\underline{46.43}$	26.63	$\underline{68.00}$	52.83	71.69	72.41	$\underline{53.79}$	**50.00**	**52.88**	54.46	71.69	72.16
Residual Scaling w./ He	57.80	33.93	33.28	$\underline{69.00}$	$\underline{57.10}$	72.17	72.82	$\underline{54.15}$	**50.00**	$\underline{51.92}$	56.70	72.17	72.60
Residual Scaling w./ Small	$\underline{60.73}$	33.93	23.04	66.00	$\underline{57.08}$	71.32	72.01	51.62	**50.00**	42.31	$\underline{57.51}$	71.32	71.74
WeSaR	**61.62**	$\underline{41.07}$	**38.54**	**76.00**	$\underline{56.81}$	**76.68**	**77.37**	50.54	$\underline{48.75}$	$\underline{44.23}$	**57.73**	**76.68**	**77.16**

Table 8: Evaluation of 1.3B models on downstream tasks. The best method is in bold, and the methods within one standard deviation are underlined.

the standard deviations. The single runs on the full 30B tokens are described in Appendix D.

WeSaR achieved a lower (*i.e.,* better) perplexity on average and smaller (*i.e.,* more stable) standard deviations than Weight Normalization. In addition, Weight Normalization took the longest time. This is because that it calculates the back-propagation through the normalization, different from the other methods. We confirmed that our simple reparameterization without normalization is efficient and effective for LLM's pre-training.

Moreover, WeSaR outperformed σReparam. Whereas σReparam controls the attention entropy for stability, WeSaR stabilizes the training by sharing the standard deviations of all of the parameters even without spectral normalization. In addition, we consider that setting the initial standard deviation to the criteria of He initialization achieved a more rapid decrease in loss than did setting the initial maximum singular value to 1.

Third, WeSaR outperformed residual scaling in terms of perplexity. Because residual scaling only reparameterizes $\boldsymbol{W}_o$ and $\boldsymbol{W}_d$, we consider that the relief of all of the parameters from the requirements by the back-propagation, which also results in smaller standard deviations than in the conventional setting, is important for a stable and rapid decrease in loss.

		WikiText	LAMBADA
130M ($d = 768$)	$\sigma^2 = $ 16e-5 ($d = 5000$)	28.64	36.52
	$\sigma^2 = $ 4e-5 ($d = 10000$)	25.07	**31.89**
	$\sigma^2 = $ 1e-5 ($d = 20000$)	**24.51**	33.46
	$\mu = $ 2e-3	24.55	33.15
	$\mu = $ 1e-3	25.07	**31.89**
	$\mu = $ 5e-4	**24.50**	33.25
1.3B ($d = 2048$)	$\sigma^2 = $ 16e-5 ($d = 5000$)	16.37	26.05
	$\sigma^2 = $ 4e-5 ($d = 10000$)	**14.51**	**22.87**
	$\sigma^2 = $ 1e-5 ($d = 20000$)	14.85	24.19
	$\mu = $ 2e-3	14.67	24.02
	$\mu = $ 1e-3	**14.51**	**22.87**
	$\mu = $ 5e-4	15.98	25.59

Table 9: Robustness versus standard deviation σ and learning rate μ. The parentheses in the second columns indicate the number of dimensions measured against the criteria of Small initialization.

Is WeSaR effective on downstream tasks? To confirm the effectiveness of WeSaR on downstream tasks, we evaluated the compared models on the SuperGLEU dataset (Wang et al., 2019) via lm-evaluation-harness (Gao et al., 2024). We used BoolQ (Clark et al., 2019), CB (De Marneffe et al., 2019), COPA (Roemmele et al., 2011), MultiRC (Khashabi et al., 2018), ReCoRD (Zhang et al., 2018), RTE (Dagan et al., 2006; Bar Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009), WiC (Pilehvar and Camacho-Collados, 2019), and WSC (Levesque et al., 2011) with the official metrics in lm-evaluation-harness. We did not conduct fine-tuning and report the results with 3-shot in-context learning.

Table 8 lists the results. In addition to the perplexity as language modeling, the model pre-trained with WeSaR outperformed the compared models on the downstream tasks on average.

Are the hyperparameter settings robust to changes in model size? Table 9 clarifies the robustness with respect to the model size.

Here, we observed that the $\sigma^2 =$ 4e-5 setting outperformed the other settings in the 1.3B model experiments, while the $\sigma^2 =$ 4e-5 and 1e-5 settings achieved comparable performance in the 130M model experiments. Although there remains room for tuning the hyperparameters, we found that the optimal standard deviations are not necessarily proportional to the dimension size d, different from the conventional setup; a larger model does not always prefer a smaller standard deviation. This is because the back-propagation to lower layers must depend on d and the proposed method assigns the role of ensuring this dependence to the gate parameter. Second, regarding the learning rate, we confirmed that WeSaR achieves stable training even with a higher rate (order of 1e-3) than that of conventional settings (order of 1e-4).

6 Related Work

Loss spikes. PaLM (Chowdhery et al., 2023) and OPT (Zhang et al., 2022) found the loss spike phenomenon and used a simple strategy against it that restarts the training from an earlier checkpoint and skips batches that may have caused the spike. GLM (Zeng et al., 2023) found that the abnormal gradients of the embedding layer usually cause spikes and proposed to shrink the gradients of $\boldsymbol{W}_e$. Li et al. (2022) and Zhai et al. (2023) argued that large context lengths and abnormal attention behavior lead to spikes. Molybog et al. (2023) indicated that the Adam optimizer, which assumes time-domain independence of gradients, induces loss spikes. Takase et al. (2023) presented embedding scaling (Vaswani et al., 2017) and LayerNorm on the top of the embedding layer (Le Scao et al., 2022) by focusing the differentiation of the layer normalization. The causes of loss spikes are still under intense discussion. We clarified that one of the causes is the non-uniformity of the parameter norms and provided a method to address this issue.

Residual scaling. The $(1/\sqrt{2N})$-fold initialization of σ_d, σ_o was first proposed in LLM studies by GPT-2 (Radford et al., 2019). Apart from Transformer, Taki (2017); Hanin and Rolnick (2018); Zhang et al. (2019b) presented a weight scaling for ResNet (He et al., 2016) together with a mathematical justification. Some recent studies have proposed weight scaling for Transformer and have given theoretical analyses, including $\mathcal{O}(N^{-1/4})$-fold scaling of $\boldsymbol{W}_v, \boldsymbol{W}_o$ (Huang et al., 2020), $\mathcal{O}(N^{-1/2})$ of $\boldsymbol{W}_o, \boldsymbol{W}_d$ as reparameterization (Noci et al., 2022), and $\mathcal{O}(N^{-1/4})$ of $\boldsymbol{W}_v, \boldsymbol{W}_o, \boldsymbol{W}_u$, and $\boldsymbol{W}_d$ (Wang et al., 2022). We have extended this line of work to the novel reparameterization method. Although we used the most popular GPT-2's strategy for the initial scale, we can use any of the scaling strategies described above.

Initialization methods. Some studies have determined the initial scale of the parameters with a prior optimization phase before the pre-training (Dauphin and Schoenholz, 2019; Zhu et al., 2021; Yang et al., 2022; Bingham and Miikkulainen, 2023). Our method can use them as the backbone initialization instead of He initialization.

7 Conclusion

Loss spikes are a fundamental issue in pre-training of LLMs because they increase the pre-training cost and degrade the performance of the model. To address this problem, we identified one of the causes as the non-uniformity of the norm of the model parameters. We proposed a novel reparameterization method, WeSaR, that addresses the non-uniformity problem by adjusting the gate parameter to the required scale and initializing the actual parameters with a common standard deviation. WeSaR not only stabilizes the pre-training, but also accelerates the pre-training by setting a standard deviation smaller than in the conventional setting. Experimental results showed that WeSaR outperformed the compared methods, and the parameters and their update ratios were stable during pre-training.

The use of LLMs has been spreading. We believe this study to be a significant contribution that increases both the efficiency of the LLM's pre-training and the effectiveness of the pre-trained LLMs.

Limitations

The proposed method and the presented theoretical analysis focus on one aspect of the loss spike problem and does not solve it entirely. In the experiments, we used various techniques designed for stable training: warmup, Adam $\beta_2 = 0.95$, gradient clipping, weight decay, and Z-loss. We do not insist that such techniques are no longer required. For example, in pilot experiments with the 1.3B model, we found that no warmup or no gradient clipping training achieved higher perplexity due to unstable behavior at the very beginning of the training. We argue that there is no silver bullet against loss spikes and that we should address this issue with a combination of techniques, including WeSaR.

Another limitation is the restriction of the computational resources. For example, our experiments investigated the behavior of the models with up to 13B parameters. Moreover, we did not use SWiGLU activation (Shazeer, 2020) in the feed-forward layers, as has been done in popular LLMs, *e.g.*, PaLM (Chowdhery et al., 2023) and LLaMA (Touvron et al., 2023). However, we note that the effectiveness of SWiGLU remains controversial in the community: Narang et al. (2021) has a positive opinion, and Allen-Zhu and Li (2024) a negative one. In spite of these restrictions, our experiments showed the effectiveness of WeSaR on a standard Transformer architecture. Our experiments took 30,272 GPU hours on H100 totally. This would cost $148,824 if the experiments were conducted on Amazon Web Service in June, 2024. We believe that our findings based on the intensive experiments shed new light on LLMs.

References

AI@Meta. 2024. Llama 3 model card.

Zeyuan Allen-Zhu and Yuanzhi Li. 2024. Physics of language models: Part 3.3, knowledge capacity scaling laws. *arXiv preprint arXiv:2404.05405*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Roy Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second PASCAL recognising textual entailment challenge.

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. 2009. The fifth PASCAL recognizing textual entailment challenge.

Garrett Bingham and Risto Miikkulainen. 2023. Autoinit: Analytic signal-preserving weight initialization for neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(6):6823–6833.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of NAACL-HLT 2019*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The PASCAL recognising textual entailment challenge. In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*, pages 177–190. Springer.

Yann N Dauphin and Samuel Schoenholz. 2019. Metainit: Initializing learning by learning to initialize. In *Advances in Neural Information Processing Systems*, volume 32.

Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The CommitmentBank: Investigating projection in naturally occurring discourse. To appear in proceedings of Sinn und Bedeutung 23. Data can be found at https://github.com/mcdm/CommitmentBank/.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2024. A framework for few-shot language model evaluation.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9. Association for Computational Linguistics.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256.

Boris Hanin and David Rolnick. 2018. How to start training: The effect of initialization and architecture. In *Advances in Neural Information Processing Systems*, volume 31.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. 2020. Improving transformer optimization through better initialization. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4475–4483.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

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

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR (Poster)*.

Teven Le Scao, Thomas Wang, Daniel Hesslow, Stas Bekman, M Saiful Bari, Stella Biderman, Hady Elsahar, Niklas Muennighoff, Jason Phang, Ofir Press, Colin Raffel, Victor Sanh, Sheng Shen, Lintang Sutawika, Jaesung Tae, Zheng Xin Yong, Julien Launay, and Iz Beltagy. 2022. What language model to train if you have one million GPU hours? In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 765–782.

Hector J Levesque, Ernest Davis, and Leora Morgenstern. 2011. The Winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, volume 46, page 47.

Conglong Li, Minjia Zhang, and Yuxiong He. 2022. The stability-efficiency dilemma: Investigating sequence length warmup for training gpt models. In *Advances in Neural Information Processing Systems*, volume 35, pages 26736–26750.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. 2020. Understanding the difficulty of training transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 5747–5763.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. 2018. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*.

Igor Molybog, Peter Albert, Moya Chen, Zachary DeVito, David Esiobu, Naman Goyal, Punit Singh Koura, Sharan Narang, Andrew Poulton, Ruan Silva, et al. 2023. A theory on adam instability in large-scale machine learning. *arXiv preprint arXiv:2304.09871*.

Sharan Narang, Hyung Won Chung, Yi Tay, Liam Fedus, Thibault Fevry, Michael Matena, Karishma Malkan, Noah Fiedel, Noam Shazeer, Zhenzhong Lan, Yanqi Zhou, Wei Li, Nan Ding, Jake Marcus, Adam Roberts, and Colin Raffel. 2021. Do transformer modifications transfer across implementations and applications? In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5758–5773.

Toan Q. Nguyen and Julian Salazar. 2019. Transformers without tears: Improving the normalization of self-attention. In *Proceedings of the 16th International Conference on Spoken Language Translation*, Hong Kong. Association for Computational Linguistics.

Lorenzo Noci, Sotiris Anagnostidis, Luca Biggio, Antonio Orvieto, Sidak Pal Singh, and Aurelien Lucchi. 2022. Signal propagation in transformers: Theoretical perspectives and the role of rank collapse. In *Advances in Neural Information Processing Systems*, volume 35, pages 27198–27211.

Yan Pan and Yuanzhi Li. 2022. Toward understanding why adam converges faster than SGD for transformers. In *OPT 2022: Optimization for Machine Learning (NeurIPS 2022 Workshop)*.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*.

Mohammad Taher Pilehvar and Jose Camacho-Collados. 2019. WiC: The word-in-context dataset for evaluating context-sensitive meaning representations. In *Proceedings of NAACL-HLT*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S. Gordon. 2011. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Tim Salimans and Durk P Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, volume 29.

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

Sho Takase, Shun Kiyono, Sosuke Kobayashi, and Jun Suzuki. 2023. Spike no more: Stabilizing the pre-training of large language models. *arXiv preprint arXiv:2312.16903*.

Masato Taki. 2017. Deep residual networks and weight initialization. *arXiv preprint arXiv:1709.02956*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*, pages 5998–6008.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. *arXiv preprint 1905.00537*.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. 2022. Deepnet: Scaling transformers to 1,000 layers. *arXiv preprint arXiv:2203.00555*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Yibo Yang, Hong Wang, Haobo Yuan, and Zhouchen Lin. 2022. Towards theoretically inspired neural initialization optimization. In *Advances in Neural Information Processing Systems*, volume 35, pages 18983–18995.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. GLM-130b: An open bilingual pre-trained model. In *The Eleventh International Conference on Learning Representations*.

Shuangfei Zhai, Tatiana Likhomanenko, Etai Littwin, Dan Busbridge, Jason Ramapuram, Yizhe Zhang, Jiatao Gu, and Joshua M. Susskind. 2023. Stabilizing transformer training by preventing attention entropy collapse. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 40770–40803.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32.

Biao Zhang, Ivan Titov, and Rico Sennrich. 2019a. Improving deep transformer with depth-scaled initialization and merged attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 898–909.

Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. 2019b. Residual learning without normalization via better initialization. In *International Conference on Learning Representations*.

Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank Reddi, Sanjiv Kumar, and Suvrit Sra. 2020. Why are adaptive methods good for attention models? *Advances in Neural Information Processing Systems*, 33:15383–15393.

Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. 2018. ReCoRD: Bridging the gap between human and machine commonsense reading comprehension. *arXiv preprint 1810.12885*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhi-Quan Luo. 2024. Why transformers need adam: A hessian perspective. *arXiv preprint arXiv:2402.16788*.

Chen Zhu, Renkun Ni, Zheng Xu, Kezhi Kong, W. Ronny Huang, and Tom Goldstein. 2021. Gradinit: Learning to initialize neural networks for stable and efficient training. In *Advances in Neural Information Processing Systems*, volume 34, pages 16410–16422.

A Analysis of Residual Scaling

Here, we present the detailed explanation of residual scaling. The back-propagation through Equation 1 is

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \boldsymbol{\delta} \left(\frac{\partial f(\mathrm{LN}(\boldsymbol{x}))}{\partial \boldsymbol{x}} + \boldsymbol{I} \right). \quad (5)$$

	130M	1.3B	13B
Hidden Size d	768	2048	5120
# Layer N	12	24	40
# Attention Head	12	16	40
Context Length		2048	
Vocabulary Size		32000	
RMSNorm ϵ		1e-5	
Positional Encoding		RoPE	
Bias in Linear		none	

Table 10: Detailed model configuration.

	Rapid Setting	Stable Setting
Batch Size [tokens]	1M	4M
Learning rate μ	1e-3	5e-4
Warmup Steps	100	2000
Precision	bfloat16	
Corpus Size [tokens]	30B	
Adam β_1	0.9	
Adam β_2	0.95	
Gradient Clipping Threshold	1	
Weight decay	0.01	
Z-loss	1e-4	

Table 11: Detailed training configuration.

We assume that δ_i and $\frac{\partial f(\mathrm{LN}(\boldsymbol{x}))_i}{\partial x_j}$ are independent and the average of $\frac{\partial f(\mathrm{LN}(\boldsymbol{x}))_i}{\partial x_j}$ is zero. Let s^2 be $E\left[\left\|\frac{\partial f(\mathrm{LN}(\boldsymbol{x}))_i}{\partial \boldsymbol{x}}\right\|^2\right]$. Here, the expectation of the norm of Equation 5 is

$$
\begin{aligned}
&E\left[\left\|\boldsymbol{\delta}\left(\frac{\partial f(\mathrm{LN}(\boldsymbol{x}))}{\partial \boldsymbol{x}}+\boldsymbol{I}\right)\right\|^2\right] \\
&\quad = E\left[\left\|\boldsymbol{\delta}\frac{\partial f(\mathrm{LN}(\boldsymbol{x}))}{\partial \boldsymbol{x}}\right\|^2\right] + E\left[\|\boldsymbol{\delta}\|^2\right] \\
&\quad = d_{\text{in}} d_{\text{out}} E\left[\|\delta_i\|^2\right] E\left[\left\|\frac{\partial f(\mathrm{LN}(\boldsymbol{x}))_i}{\partial x_j}\right\|^2\right] \\
&\qquad + d_{\text{out}} E\left[\|\delta_i\|^2\right] \\
&\quad = \left(d_{\text{in}} E\left[\left\|\frac{\partial f(\mathrm{LN}(\boldsymbol{x}))_i}{\partial x_j}\right\|^2\right] + 1\right) E\left[\|\boldsymbol{\delta}\|^2\right] \\
&\quad = (s^2+1) E\left[\|\boldsymbol{\delta}\|^2\right].
\end{aligned}
$$

Thus, a residual connection causes an (s^2+1)-fold increase in the squared norm of the gradient $E\left[\left\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}\right\|^2\right]$.

B Experimental Setup

Table 10 and 11 list the detailed model and training configurations, respectively. We used eight NVIDIA H100 (80GB) GPUs for pre-training the 130M and 1.3B models and 64 GPUs for pre-training the 13B models. The pre-trainings took roughly 12 hours, 60 hours, and 40 hours, respectively. We used the Adam optimizer (Kingma and Ba, 2015), PyTorch (ver. 2.1.0)[4] (Paszke et al., 2017), transformers (ver. 4.37.2)[5] (Wolf et al., 2019), and llm-foundry (ver. 0.5.0) [6].

C Relation to Existing Reparameterization Methods

C.1 Weight Normalization

Weight Normalization (Salimans and Kingma, 2016) conducts L2 normalization and scaling of each row of the parameter matrix $\boldsymbol{w} \in \mathbb{R}^{d_{\text{in}}}$:

$$\bar{\boldsymbol{w}} = \frac{\alpha}{\|\boldsymbol{w}\|}\boldsymbol{w}.$$

It differentiates the whole operation including the normalization and propagates the gradient to $\boldsymbol{w}$. It determines the initial α from the value of the forward computation in the first step. The proposed method is efficient because it does not conduct normalization and provides a matrix-wise reparameterization; the number of the additional parameter α per parameter matrix is one.

C.2 σReparam

σReparam (Zhai et al., 2023) conducts spectral normalization and scaling of the parameter matrix $\boldsymbol{W} \in \mathrm{R}^{d_{\text{out}} \times d_{\text{in}}}$,

$$\bar{\boldsymbol{W}} = \frac{\alpha}{\|\boldsymbol{W}\|_2}\boldsymbol{W},$$

where $\|\boldsymbol{W}\|_2$ is the spectral norm (*i.e.,* the maximum singular value). The maximum singular value is calculated by the power method that is iterated once per batch (Miyato et al., 2018). It does not differentiate the spectral normalization. σReparam is based on the fact that the entropy in the self-attention affects the stability of the training. It regulates the singular value of $\bar{\boldsymbol{W}}$ so as to control the entropy. α is initialized to 1. Therefore, σReparam is different from the proposed method, which is designed to align the virtual parameter $\alpha.\boldsymbol{W}.$ to any initialization algorithm, such as He initialization, while setting the standard deviations of the actual parameter $\boldsymbol{W}.$ independently.

[4] https://pytorch.org/

[5] https://github.com/huggingface/transformers

[6] https://github.com/mosaicml/llm-foundry

	WikiText	LAMBADA
Small Init.	16.55	26.29
Weight Normalization	**14.13**	24.97
σReparam w./ Pre-LN	18.83	26.22
σReparam w./ Post-LN	16.52	25.58
Residual Scaling w./ He Init.	19.05	27.36
Residual Scaling w./ Small Init.	18.03	26.88
WeSaR	14.51	**22.87**

Table 12: Comparison of reparameterization methods on 30B tokens.

C.3 Residual Scaling as Reparameterization

Noci et al. (2022) overcomes the limitation of the $(1/\sqrt{2N})$-fold multiplication of σ_o and σ_d caused by the residual connection (Equation 1). It modifies the residual connection to

$$\boldsymbol{y} = \frac{1}{\sqrt{2N}} f(\mathrm{LN}(\boldsymbol{x})) + \boldsymbol{x}.$$

Different from the original residual scaling, which changes the standard deviations, this equation can be viewed as a reparameterization of $\boldsymbol{W}_o$ and $\boldsymbol{W}_d$ because of its linearity. The proposed method can be interpreted as a generalization of the reparameterization to all parameters. Because of the generalization, the proposed method overcomes any limitations to the norms of the parameters that is caused by an initialization algorithm. Therefore, it can determine a common σ for all parameters even without a dependence on d. Also, it makes the gate parameters trainable.

D Comparison of Reparameterization Methods on 30B Tokens

We compared WeSaR with the existing reparameterization methods on 30B tokens. The results shown in Table 12 achieved the same tendency as the results on 10B tokens. In particular, similar to the results of five runs on 10B tokens in Table 6, Weight Normalization achieved comparable performance. However, Weight Normalization took the longest time for the training due to the back-propagation through the normalization. Thus, WeSaR's simple reparameterization is efficient and effective for LLM's pre-training.

E Loss Values without Loss Spikes

Figure 4 and 5 show the loss decrease and the update ratios at the beginning of the training of the 1.3B models and the 130M models, respectively. Because the 1.3B and 130M models did not cause loss spikes, we did not observe a drastic decrease in the update ratios like with the 13B models. Except for that point, the update ratio behaved similarly to the 13B models. We should note that, in smaller models, the effect of $(1/\sqrt{2N})$-fold scaling gets smaller, and thus there is less difference between the baseline method and WeSaR. Figure 6 and 7 show the loss values during the training.

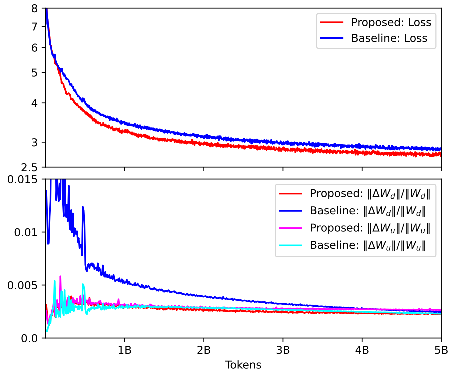

Figure 4: Loss of the 1.3B Transformer models at the beginning of the training (top). Update ratios $\|\Delta\boldsymbol{W}_d\|/\|\boldsymbol{W}_d\|$ and $\|\Delta\boldsymbol{W}_u\|/\|\boldsymbol{W}_u\|$ of the same (bottom).

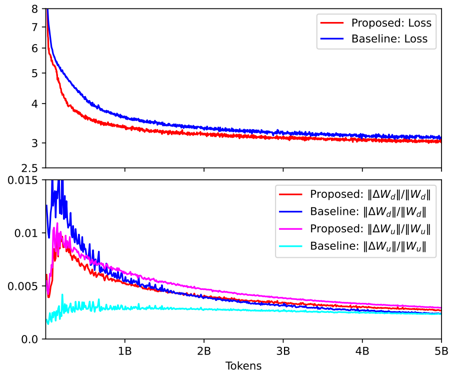

Figure 5: Loss of the 130M Transformer models at the beginning of the training (top). Update ratios $\|\Delta\boldsymbol{W}_d\|/\|\boldsymbol{W}_d\|$ and $\|\Delta\boldsymbol{W}_u\|/\|\boldsymbol{W}_u\|$ of the same (bottom).

Moreover, we confirmed that WeSaR outperformed Small initialization both in the loss values in Figure 1, 2, 4, 5, 6, and 7 and the perplexity in Table 5. We consider that the small standard deviation $\sigma^2 = $ 4e-5, which corresponds to $d = 10,000$ in the Small initialization criteria, accelerated the training.

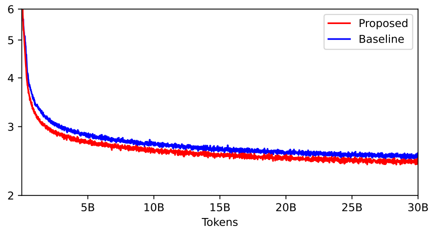

Figure 6: Loss of 1.3B models during training.

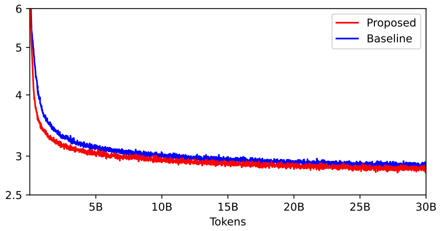

Figure 7: Loss of 130M models during training.

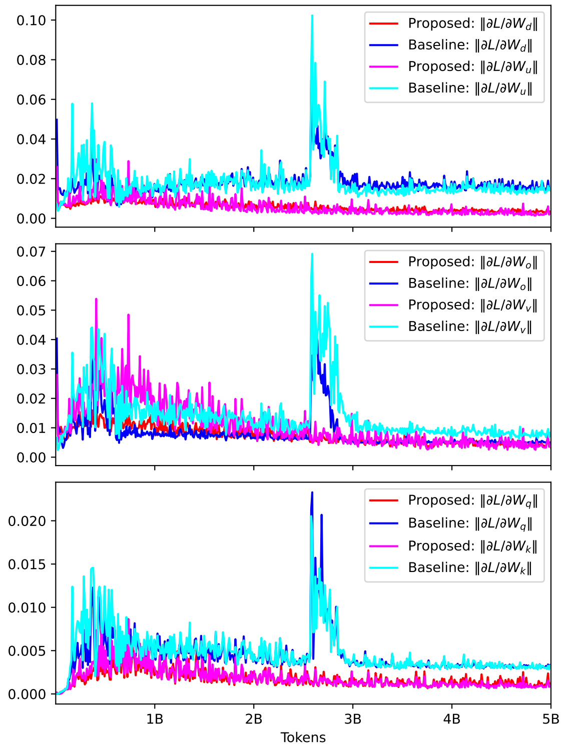

Figure 8: Norm of the gradient of the parameters at the 40th layer of 13B models during training.

F Update Ratios in Other Layers and Comparison with Gradient Norm

Figure 9, 10, 11, and 12 show the update ratios $\|\Delta \boldsymbol{W}_{\cdot}\|/\|\boldsymbol{W}_{\cdot}\|$ of all linear layers at the 40th, 27th, 14th, and 1st Transformer layers in the 13B models, respectively. Because the 1.3B and 130M models did not cause loss spikes as shown in Figure 4 and Figure 5, we only list the update ratios of the 13B models.

We observed that the update ratios $\|\Delta \boldsymbol{W}_d\|/\|\boldsymbol{W}_d\|$ and $\|\Delta \boldsymbol{W}_o\|/\|\boldsymbol{W}_o\|$ in the baseline method decreased after loss spikes, except for $\boldsymbol{W}_o$ in the 1st layer. We consider that $\boldsymbol{W}_d$ and $\boldsymbol{W}_o$, the parameters whose norm is smaller than the others, caused loss spikes due to their large update ratios. We also confirmed that the update ratios trained with WeSaR were stable among all layers and all parameters.

The existing studies that tackled loss spikes (Zeng et al., 2023; Zhai et al., 2023; Molybog et al., 2023; Takase et al., 2023) focused on the gradient norm $\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}\|$ as a clue to understand loss spikes. However, instead of the gradient norm itself, we focused the update ratio. Figure 8 shows the norm of the gradients of the parameters at the last layer of the 13B models, which corresponds to the update ratios shown in Figure 9. We observed that a phenomenon of a drastic change in scale before and after loss spikes only appeared in the update ratio. Thus, we introduced the update ratio as a novel clue to understand loss spikes.

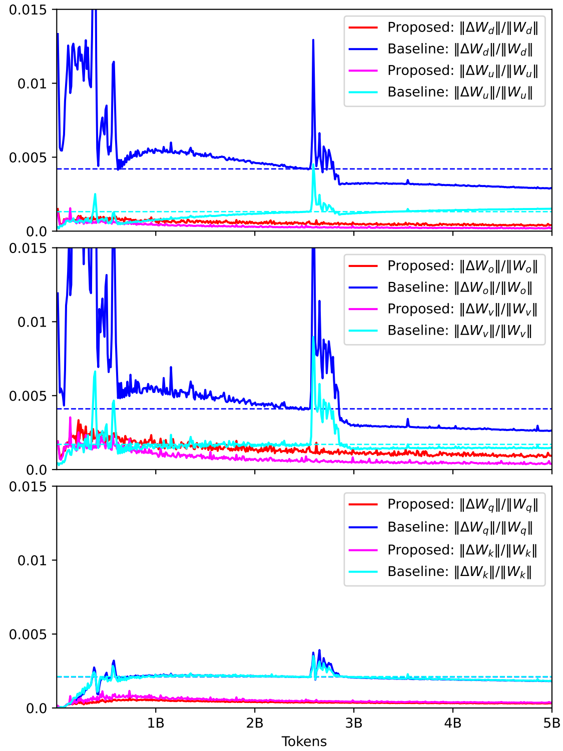

Figure 9: Update ratio at the 40th layer of 13B models during training.

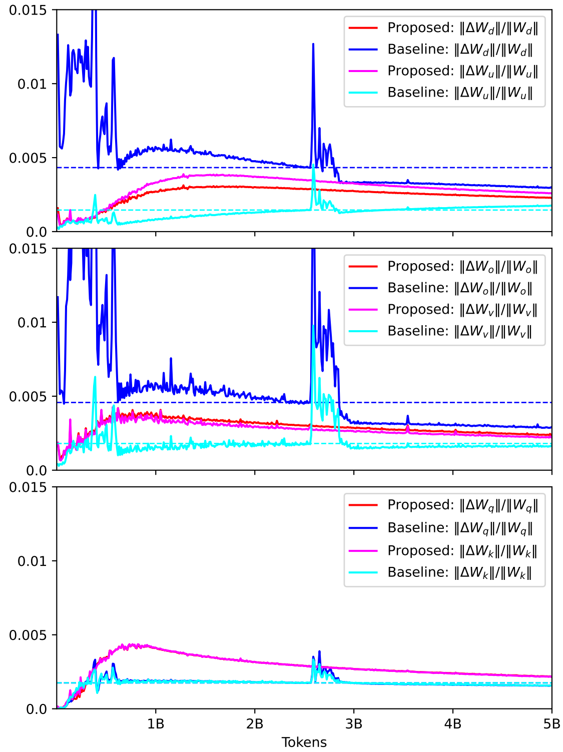

Figure 10: Update ratio at the 27th layer of 13B models during training.

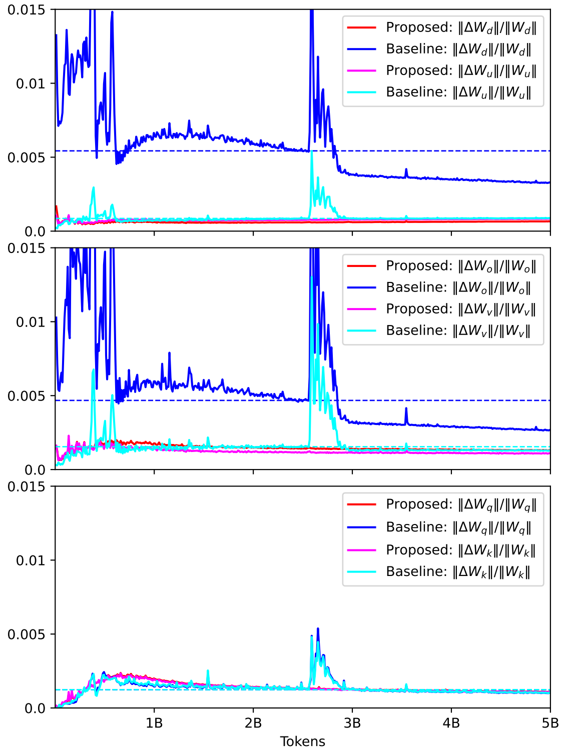

Figure 11: Update ratio at the 14th layer of 13B models during training.

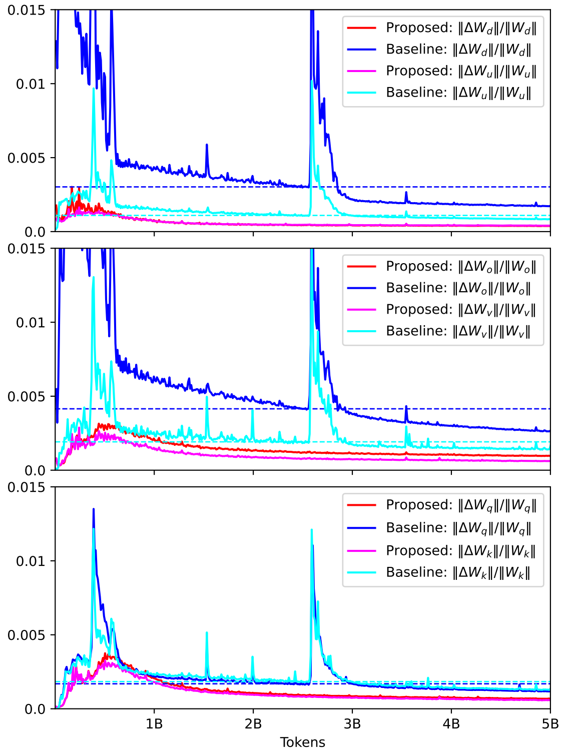

Figure 12: Update ratio at the 1st layer of 13B models during training.