BranchNorm: Robustly Scaling Extremely Deep Transformers

Yijin Liu, Fandong Meng* **and Jie Zhou**
Pattern Recognition Center, WeChat AI, Tencent Inc, China
{yijinliu, fandongmeng, withtomzhou}@tencent.com

Abstract

Recently, DeepNorm scales Transformers into extremely deep (*i.e.,*in0 layers) and reveals the promising potential of deep scaling. To stabilize the training of deep models, DeepNorm (Wang et al., 2022a) attempts to constrain the model update to a constant value. Although applying such a constraint can benefit the early stage of model training, it may lead to undertrained models during the whole training procedure. In this paper, we propose BranchNorm, which dynamically rescales the non-residual branch of Transformer in accordance with the training period. BranchNorm not only theoretically stabilizes the training with smooth gradient norms at the early stage, but also encourages better convergence in the subsequent training stage. Experimental results on multiple translation tasks demonstrate that BranchNorm achieves a better trade-off between training stability and converge performance.

1 Introduction

In recent years, Transformers (Vaswani et al., 2017) have been developed rapidly and achieved state-of-the-art (SOTA) performance on a wide range of tasks. Meanwhile, the model capacity gets substantially expanded by widening the model dimension (Devlin et al., 2019; Liu et al., 2019; Lin et al., 2021; Smith et al., 2022). Given that deep neural models learn feature representations with multiple layers of abstraction (LeCun et al., 2015), it is more attractive to increase the capacity of the model by scaling depths rather than widths. Unfortunately, due to the training instability of Transformers, the depths of these SOTA models are still relatively shallow (Kaplan et al., 2020; Hoffmann et al., 2022).

To stabilize the training of Transformers, there have been various efforts on better architectures (Shleifer et al., 2021; Wang et al., 2022b, 2023), or the implementation of proper initialization (Zhang et al., 2019a; Huang et al., 2020; Wang et al., 2022a). Among them, the most representative approach is DeepNorm (Wang et al., 2022a), which first scales Transformers to 1000 layers and significantly outperforms existing shallow counterparts.

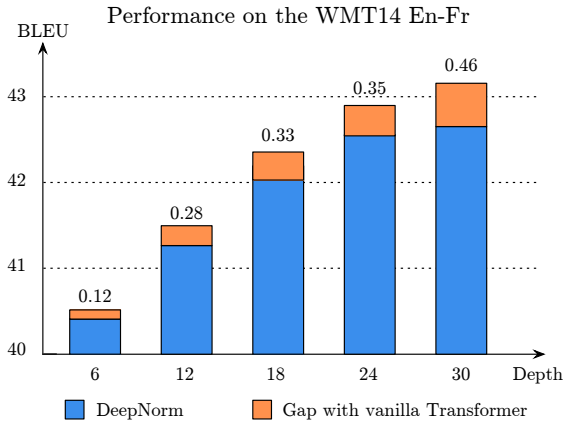

Figure 1: BLEU(%) scores on the WMT2014 En-Fr machine translation benchmark upon fully convergent models. The term 'Gap' denotes the observed decrease in performance when DeepNorm is applied to the vanilla Transformer."

Specifically, DeepNorm aims to constrain the model update to a constant level by upweighting the residual connections in Transformer and reducing the variance of parameter initialization. As a result, the stability of Transformers is improved in the early training stage. However, such constraints on the magnitude of parameter updates may ultimately yield undertrained models during the subsequent training stage. To verify the above conjecture, we first conduct experiments on shallow models to ensure that all models can be trained stably to convergence. As shown in Figure 1, it is observed that DeepNorm brings a certain degree of performance decline on vanilla Transformers, and this issue tends to get worse when models get deeper.

* Corresponding author.

Findings of the Association for Computational Linguistics ACL 2024, pages 11675–11687

To address the above issue, we propose a simple yet effective approach to robustly scale extremely deep Transformers, named BranchNorm. Specifically, the non-residual branch[1] of the Transformer is dynamically rescaled according to the training period. In the early stage of model training, BranchNorm theoretically stabilizes the training with smooth gradient norms. While in the subsequent training stage, BranchNorm progressively degenerates into a vanilla Post-LayerNorm (*i.e.,* Post-LN) to promote better convergence. Experiments on a wide range of tasks show that BranchNorm brings consistent improvement over DeepNorm and effectively alleviates the above undertrained issue. Moreover, BranchNorm performs more robustly on some key hyperparameters (*e.g.,* warmup, and learing rate) than DeepNorm, which makes it likely to be a portable alternative for scaling extremely deep Transformers.

The contributions of this paper can be summarized as follows [2]:

- We introduce BranchNorm, a straightforward yet effective normalization method to stabilize the training of extremely deep Transformers.
- BranchNorm demonstrates superior training stability and convergence performance compared to existing methods, across a series of extremely deep models and various tasks.
- BranchNorm is able to alleviate the problem of parameter redundancy in extremely deep models, from the perspective of representing similarity and sparsity of activation functions.

2 Background

In this section, we first provide a brief overview of the difference between Post-LN and Pre-LN in, and subsequently introduce the existing DeepNorm.

3 Intruduction of Post-LN and Pre-LN

Firstly, Wang et al. (2019); Nguyen and Salazar (2019) observe that the position of LayerNorm (Ba et al., 2016) has a significant effect on training stability, and propose the more stable Pre-LN variant compared to the original Post-LN (Vaswani et al., 2017). An example of these two architectures is shown in Figure 2. Subsequently, Liu et al. (2020) further analyze that Pre-LN may have an excessive reliance on its residual connections, which inhibits the model from unleashing its full potential. Motivated by the above observation, we base our approach on Post-LN in the remainder of our experiments. Formally, within the l-th sub-layer of the Post-LN Transformer, given the input x_l, the subsequent output x_{l+1} is computed as follows:

$$x_{l+1} = LN(x_l + \mathcal{F}(x_l; \theta_l)) \quad (1)$$

where LN is an abbreviation for LayerNorm[3], $\mathcal{F}$ represents the function of the current sub-layer (attention or feed-forward) and θ_l denotes the corresponding parameters of the l-th sub-layer.

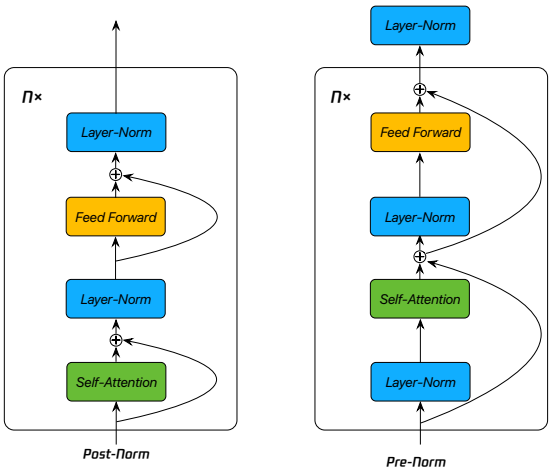

Figure 2: The architectures of Pre-Norm (*i.e.,* Pre-LN) and Post-Norm (*i.e.,* Post-LN) Transformers.

DeepNorm. DeepNorm follows the Post-LayerNorm (*i.e.,* Post-LN) architecture and rescales the residual branch with a scalar factor $\alpha > 1$. Formally, the l-th sub-layer in DeepNorm conduct calculations as follows:

$$x_{l+1} = LN(\alpha x_l + \mathcal{F}(x_l; \theta_l)) \quad (2)$$

where LN is an abbreviation for LayerNorm, $\mathcal{F}$ represents the function of the current sub-layer (attention or feed-forward) and θ_l denotes the corresponding parameters of the l-th sub-layer. In addition, DeepNorm reduces the variance of the initial parameters by scaling factor $\beta < 1$. Both the α and β are functions of model depths, which are derived from the assumption of constant model update. For a standard Transformer with N-layer encoder and

[1]Note that we name the residual connections in Transformer as 'residual branch' and the other branch as 'non-residual branch' in this paper.

[2]Codes are available at https://github.com/Adaxry/BranchNorm

[3]For brevity, the two learnable parameters γ and β in LayerNorm are omitted.

M-layer decoder, DeepNorm calculate α and β as follows:

$$\begin{aligned}
\alpha_{encoder} &= 0.81(N^4M)^{\frac{1}{16}} \\
\beta_{encoder} &= 0.87(N^4M)^{-\frac{1}{16}} \\
\alpha_{decoder} &= (3M)^{\frac{1}{4}} \\
\beta_{decoder} &= (12M)^{-\frac{1}{4}}
\end{aligned} \tag{3}$$

Note that β merely affects the model initialization, whereas α is used and remains fixed throughout the whole training procedure. Moreover, with the model gets deeper, DeepNorm will assign larger value to α, which leads to the model outputs x_{l+1} depend too much on the residual branch αx_l. In turn, the model parameters θ_l on the non-residual branch may get insufficient training.

4 Approaches

In this section, we first analyze the instability of Post-LN from the perspective of gradient norm, then demonstrate how DeepNorm can alleviate the unbalanced gradients to a certain extent, and finally introduce our proposed method BranchNorm.

4.1 Perspective of Gradient

Unbalanced gradients are mainly responsible for the instability of Transformer[4] (Wang et al., 2019; Shleifer et al., 2021; Zhu et al., 2021), we firstly explore the relation between gradient and model depth following Wang et al. (2019). Given a Transformer with L sub-layers and the training loss $\mathcal{E}$, the gradient for the l-th sub-layer is calculated by the chain rule[5]:

$$\frac{\partial \mathcal{E}}{\partial x_l} = \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1} \frac{\partial \mathrm{LN}\left(\mathcal{F}\left(x_k;\theta_k\right)\right)}{\partial \mathcal{F}\left(x_k;\theta_k\right)}}_{LN} \times \underbrace{\prod_{k=l}^{L-1}\left(1+\frac{\partial \mathcal{F}\left(x_k;\theta_k\right)}{\partial x_k}\right)}_{residual} \tag{4}$$

where the gradient consists of three terms, namely the irreducible item, the LN item and the residual item. It should be noted that the last two items are multiplicative with respect to the number of model layers L. Once L gets larger, the values of the last two items may become very large or very small, which can cause the gradient to vanish or explode.

Similarly, we analyze the gradient of DeepNorm based on Equation (2), and get its gradient at the l-th sub-layer is calculated by:

$$\frac{\partial \mathcal{E}}{\partial x_l} = \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1}\left(\frac{\partial \mathrm{LN}\left(\alpha x_k+\mathcal{F}\left(x_k;\theta_l\right)\right)}{\partial\left(\alpha x_k+\mathcal{F}\left(x_k;\theta_l\right)\right)}\right)}_{LN} \times \underbrace{\prod_{k=l}^{L-1}\left(\alpha+\frac{\partial \mathcal{F}\left(x_k;\theta_k\right)}{\partial x_k}\right)}_{residual} \tag{5}$$

As L increases, DeepNorm enhances training stability by increasing the value of α. Theoretically, α can go to infinity to represent the upper bound of DeepNorm's stability. Here, we introduce this assumption to simplify the derivation: If α get large enough, *i.e.,* $\alpha \to \infty$, the LN item can be approximated as $\prod_{k=l}^{L-1} \frac{\partial \mathrm{LN}(\alpha x_k)}{\partial(\alpha x_k)}$, and the residual item can be approximated as $\prod_{k=l}^{L-1} \alpha$, we put them into Equation (5) and can simplify it as follows:

$$\begin{aligned}
\frac{\partial \mathcal{E}}{\partial x_l} &\approx \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1}\left(\frac{\partial \mathrm{LN}\left(\alpha x_k\right)}{\partial\left(\alpha x_k\right)}\right)}_{LN} \times \underbrace{\prod_{k=l}^{L-1} \alpha}_{residual} \\
&= \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1}\left(\frac{\partial \mathrm{LN}\left(x_k\right)}{\partial x_k} \cdot \frac{1}{\alpha}\right)}_{LN} \times \underbrace{\prod_{k=l}^{L-1} \alpha}_{residual} \\
&= \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1} \frac{\partial \mathrm{LN}\left(x_k\right)}{\partial x_k}}_{LN} \quad (\alpha \to \infty)
\end{aligned} \tag{6}$$

When compared to the gradient of Post-LN in Equation (4), DeepNorm can approximately eliminate the last residual item $\prod_{k=l}^{L-1}\left(1+\frac{\partial \mathcal{F}(x_k;\theta_k)}{\partial x_k}\right)$. Considering that this residual item is calculated by the multiplication of a sequence of gradients, such a multiplication term can easily become very large or very small as the number of model layers L becomes large. Therefore, eliminating this term can help to mitigate the problem of vanishing or exploding gradients. Although a larger α in DeepNorm generally results in more stable gradients, it may come at the expense of final convergence performance, as mentioned above. Given that the

[4] In recent years, there are also researchers questioning this point and providing different perspectives (Liu et al., 2020; Wang et al., 2022a). Given that more explorations and discussions are needed to make it out, we still conduct analysis from the perspective of the gradient norm in this paper.

[5] More detailed derivations are listed in Appendix A.

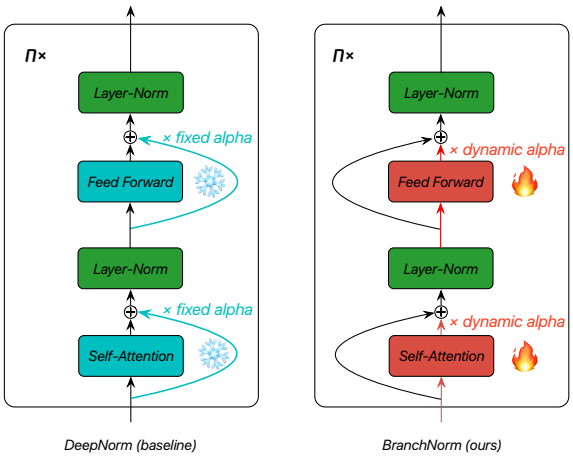

Figure 3: Structure differences between DeepNorm and BranchNorm: DeepNorm applies fixed α on residual branches, while BranchNorm applies dynamic α on non-residual branches.

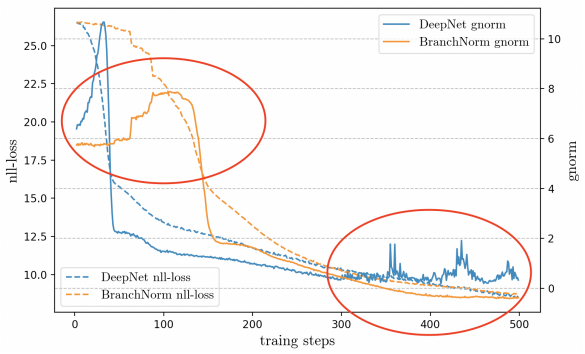

Figure 4: Gradient norm (solid line) and negative log likelihood loss (nll-loss, dotted line) at the beginning of training. BranchNorm presents a more smooth gradient.

unbalanced gradients generally occur during the early training stage, it may be more appropriate if α can be varied based on the training period.

4.2 BranchNorm

In this section, we summarize the observations from previous sections and introduce BranchNorm:

$$x_{l+1} = LN(x_l + \alpha \mathcal{F}(x_l; \theta_l)) \tag{7}$$

As shown in Figure 3, BranchNorm is analogous to the dual form of DeepNorm, but with two key differences: First, our BranchNorm utilizes a dynamic factor (*i.e.,* α) that allows it to normalize gradients during the early training stage and gradually eliminate the negative effects of these normalizations during the later stage. In contrast, DeepNorm applies a fixed factor to constrain the model updates regardless of the stage of training. Second, unlike DeepNorm, which stabilizes model gradients by upweighting the residual branch of the Transformer and requires the strong assumption in Equation (5), our BranchNorm directly adjusts the parameter branch of the Transformer (*i.e.,* non-residual branch). This allows for more precise control of model gradients, without the need for the strong assumption introduced by DeepNorm.

To make the α in BranchNorm sensitive to the training stage, we adopt a simple linear function[6] with respect to the training step t as follow:

$$\alpha_t = \min(1.0, t/T) \tag{8}$$

[6]We explore the effects of different growing strategies of α in Section 7.1, and end up with the simplest linear one.

where T is a predefined maximum number of steps to conduct BranchNorm. At the very beginning of training, α_t is approaching 0, which means that the model approximates the constant transformation and gets updated with smooth gradients. Once the training step t reaches the predefined maximum step T, BranchNorm degenerates to the vanilla Post-LN to achieve better convergence. Following the theoretical analysis in the preceding sections, we derive the gradient of BranchNorm for the l-th sub-layer as follows:

$$\begin{aligned}
\frac{\partial \mathcal{E}}{\partial x_l} &= \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1} \frac{\partial \mathrm{LN}\left(x_k + \alpha \mathcal{F}(x_k; \theta_l)\right)}{\partial \left(x_k + \alpha \mathcal{F}(x_k; \theta_l)\right)}}_{LN} \times \\
&\quad \underbrace{\prod_{k=l}^{L-1} \left(1 + \alpha \frac{\partial \mathcal{F}(x_k; \theta_l)}{\partial x_k}\right)}_{residual} \\
&= \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1} \frac{\partial \mathrm{LN}(x_k)}{\partial x_k}}_{LN} \quad (\alpha = 0)
\end{aligned} \tag{9}$$

At the very beginning of training, when compared with the Post-LN in Equation (4), BranchNorm can stabilize the gradient norm through approximately eliminating the last residual item $\prod_{k=l}^{L-1}\left(1 + \frac{\partial \mathcal{F}(x_k;\theta_k)}{\partial x_k}\right)$. However, DeepNorm requires a relatively strong assumption ($\alpha \to \infty$) in Equation (6) to achieve the same purpose. Experimentally, as shown in Figure 4, we observe that the gradient of BranchNorm is smoother at the beginning of training, indicating that its training process is more stable.

Models	LN	6L-6L	18L-18L	50L-50L	100L-100L	250L-250L
Vanilla Post-LN (Vaswani et al., 2017) †	Post	28.0		diverged		
DS-Init (Zhang et al., 2019a) †	Post	27.5		diverged		
Admin (Liu et al., 2020) †	Post	27.3	28.8	28.9	diverged	
ReZero (Bachlechner et al., 2020) †	No	27.2	28.3	28.5	diverged	
R-Fixup (Zhang et al., 2019b)	No	27.5	28.4	27.7	diverged	
T-Fixup (Huang et al., 2020)	No	27.5	28.4	27.9	diverged	
Vanilla Pre-LN (Vaswani et al., 2017) †	Pre	26.8	27.9	28.2	28.3	28.5
DLCL (Wang et al., 2019) †	Pre	27.2	28.3	28.5	28.8	29.1
NormFormer (Shleifer et al., 2021) †	Pre	27.0	28.3	28.0	28.7	diverged
Magneto (Wang et al., 2023) †	Pre	27.5	28.3	28.7	27.7	27.9
DeepNorm (Wang et al., 2022a)	Post	27.8	28.8	29.0	28.9	–
DeepNorm (Wang et al., 2022a) †	Post	28.6	29.1	29.7	29.3	29.5
BranchNorm (ours)	Post	**28.9**	**29.8***	**30.2***	**30.8***	**30.6***

Table 1: BLEU scores (%) on the WMT-17 En-De test set with depth-scaling. † indicates our reimplementations. AL-BL refers to a Transformer with A-layer encoder and B-layer decoder. '*' means BranchNorm is significantly better than DeepNorm with $p < 0.03$.

5 Experiment Settings

We conduct extensive experiments on bilingual translation, text summarization and language modeling tasks to verify our approach. In this section, we will describe our experimental settings and present the main results.

5.1 Datasets and Evaluations

Machine Translation. We use the standard WMT 2017 English-German (En-De) with 4.5M pairs and the WMT 2014 English-French (En-Fr) datasets with 30M pairs for bilingual translation tasks, which is processed following the official scripts of fairseq[7]. We use the *multibleu.perl* to calculate cased sensitive BLEU scores for WMT 2017 En-De[8] and WMT 2014 En-Fr. Besides, we use sacreBLEU[9] to calculate cased sensitive BLEU scores for OPUS-100 and cased insensitive BLEU scores for MultiUN following Wang et al. (2021).

Text Summarization. We use the popular CNN/DailyMail dataset for the text summarization task, comprising more than 300k unique articles authored by journalists from CNN and Daily Mail. We follow the pre-processing and post-processing scrips of existing studies (Qi et al., 2020). For evaluation, we set the beam size to 4 and the length penalty to 1.0 during inference, and report the F1 scores of ROUGE-1, ROUGE-2, and ROUGE-L on the standard test set with 11,490 samples.

Language Modeling. We use the decoder-only Transformer to carry out the language modeling task. We take the popular WikiText-103 dataset, which comprises a corpus of more than 100 million tokens extracted from a curated set of verified good and featured articles on Wikipedia. We report the perplexity on the standard test set for evaluation.

5.2 Training Settings

Our experiments are based on the fairseq (Ott et al., 2019). For all experiments, we use the standard Transformer base setting, which sets hidden dimensions to 512 and feedforward inner representation to 2048 We use the encoder-decoder Transformer for machine translation and text summarization, while use decoder-only Transformer for language modeling. We used the DeepNorm initialization method (Wang et al., 2022a) only for BranchNorm and DeepNorm, the initialization method derived from the corresponding paper for Magneto(Wang et al., 2023), and the Xavier initialization method (Glorot and Bengio, 2010) for vanilla Pre-LN/Post-LN Transformers. All experiments are conducted on 32 NVIDIA A100 GPUs, where each is allocated with a batch size of approximately 16,384 tokens. All Transformer models are trained for 100k steps with the early stop for small-scale datasets. The maximum norm step T of Branch-

[7]https://github.com/facebookresearch/fairseq

[8]After confirming with the authors of DeepNorm, we use the same test set (*i.e., newstest2014.*) with them for a rigorous comparison.

[9]https://github.com/mjpost/sacrebleu

Models	LN	6L-6L	18L-18L	50L-50L	100L-100L	250L-250L	500L-500L
Vanilla Post-LN (2017) †	Pre	41.5	43.25		diverged		
Vanilla Pre-LN (2017) †	Pre	41.0	42.5	42.7	43.1	43.3	43.2
DLCL (2019) †	Pre	41.3	42.8	43.1	43.3	43.3	43.5
Magneto (2023) †	Pre	41.1	42.7	43.3	43.3	43.4	43.2
DeepNorm (2022a) †	Post	41.2	43.0	43.8	43.8	43.9	43.6
BranchNorm (ours)	Post	**41.6***	**43.6***	**43.9**	**44.2***	**44.3***	**44.4***

Table 2: BLEU scores (%) on the WMT-14 En-Fr test set with depth-scaling. † indicates our reimplementations. AL-BL refers to a transformer with a A layer encoder and a B layer decoder. '*' means BranchNorm is significantly better than DeepNorm with $p < 0.03$.

Models	LN	12L	24L	100L	500L
Performer	Post	26.8	–	–	–
Reformer	Post	26.0	–	–	–
Transformer-XL	Post	24.0	18.3	–	–
GPT-2	Pre	29.9	17.0	–	–
Hybrid H3	Pre	**23.7**	16.9	–	–
Pre-LN †	Pre	24.7	20.5	17.4	16.5
DLCL †	Pre	24.1	20.1	17.2	15.8
Magneto †	Pre	25.8	18.9	16.6	15.3
DeepNorm †	Post	24.5	17.2	14.3	14.2
BranchNorm	Post	23.9	**16.6**	**13.9**	**12.1**

Table 3: Test perplexity scores (lower is better) on the WikiText-103 with depth-scaling. The **bolded** scores correspond to the best performance under the under the same or comparable setups. † indicates our reimplementations, while other results are cited from corresponding papers. AL refers to a decoder-only transformer with a A layer decoder.

Norm in Equation (8) is set to 4,000 for all experiments. More details about the model hyperparameters are elaborated in Appendix B.

5.3 Comparison Systems

We compare several state-of-the-art approaches for deep Transformers, including DeepNorm (Wang et al., 2022a), Magneto (Wang et al., 2022b, 2023), NormFormer (Shleifer et al., 2021), ReZero (Bachlechner et al., 2020), DLCL (Wang et al., 2019) and *etc.* To ensure that the training framework is the same across different approaches, we implement most of related methods following their official source codes or original papers on Fairseq. Other results are cited from corresponding papers.

6 Experimental Results

6.1 Results on Bilingual Translation Tasks.

Table 1 and Table 2 report the translation performance of different methods at various model depths on the WMT 2017 En-De and WMT 2014 En-Fr dataset. In most cases, the training of vanilla Post-LN Transformer gets diverged due to its own training instability. As the model depth increases, we observe that existing methods can stabilize the training to varying degrees. Among them, the well-performing existing work, DeepNorm, although capable of stably training deep models, does not fully exploit the potential performance of these models. For instance, on the WMT14 En-Fr dataset, deep models (*e.g.,* 500L-500L) and shallow models (*e.g.,* 50L-50L) trained with DeepNorm surprisingly exhibit similar performance levels. In contrast, our proposed BranchNorm consistently outperforms in different depth settings. Furthermore, the performance improves as the number of model layers increases, effectively mitigating the aforementioned performance degradation problem. Moreover, BranchNorm significantly outperforms previous deep models by up to +1.5 BLEU given the same model depths. Notably, on the WMT-14 En-Fr benchmark, our 1000 layer model outperforms existing deep models and achieves a new SOTA performance of 44.4 BLEU.

6.2 Language Modeling

On the WikiText-103 dataset, we compared a series of strong baseline models on shallow models (*i.e.,* 12-layer and 24-layer models), such as Transformer-XL (Dai et al., 2019) which models recurrent context, Performers (Choromanski et al., 2020) with linear computational efficiency, Hybrid H3 models (Dao et al., 2022) that improve the computational efficiency of state space models, and

Model	LN	RG-1	RG-2	RG-L
6L-6L structure				
Post-LN	Post	39.5	16.1	36.7
Mask Attention	Post	41.0	**18.2**	37.2
Dynamic Conv	Post	39.8	16.3	36.7
Pre-LN †	Pre	40.2	16.7	36.8
DLCL †	Pre	40.5	16.9	36.7
Magneto †	Pre	39.7	16.4	36.9
DeepNorm †	Post	40.9	17.6	37.2
BranchNorm (ours)	Post	**41.3**	17.9	**38.0**
18L-18L structure				
Pre-LN †	Pre	39.7	16.9	36.3
DLCL †	Pre	40.8	16.3	37.1
Magneto †	Pre	40.1	17.3	37.7
DeepNorm †	Post	40.8	17.8	37.9
BranchNorm (ours)	Post	**41.5**	**18.4**	**38.8**
50L-50L structure				
Pre-LN †	Pre	40.3	18.2	38.1
DLCL †	Pre	40.8	16.3	37.1
Magneto †	Pre	40.6	17.9	37.3
DeepNorm †	Post	41.2	19.6	39.6
BranchNorm (ours)	Post	**42.5**	**20.2**	**40.7**
200L-200L structure				
Pre-LN †	Pre	41.1	18.9	38.9
DLCL †	Pre	41.4	19.3	39.5
Magneto †	Pre	42.4	17.3	39.9
DeepNorm †	Post	42.2	19.8	40.8
BranchNorm (ours)	Post	**43.5**	**21.8**	**41.5**

Table 4: F1 scores of ROUGE-1 / ROUGE-2 / ROUGE-L on the test set of the CNN/DailyMail dataset with model depth-scaling. 'RG' is an abbreviation for 'ROUGE'. † indicates our reimplementations, while other results are cited from corresponding papers. AL-BL refers to a transformer with a A layer encoder and a B layer decoder. The **bolded** scores correspond to the best performance under the same or comparable setup.

so on. Experimental results at Table 3 indicate that our BranchNorm achieves comparable performance with existing state-of-the-art methods on shallow models. On deeper model structures (*i.e.,* 100-layer and 500-layer models), we reproduce the performance of a series of existing works for comparison. The experimental results demonstrate that BranchNorm exhibits superior convergence on these extremely deep models, with improvements of 3.2 and 2.1 points over Magneto and DeepNorm on the 500-layer models, respectively.

6.3 Results on Text Summarization

We compare the performance of models with different layers in Table 4. On the commonly used shallow model structure 6L-6L, we observe that

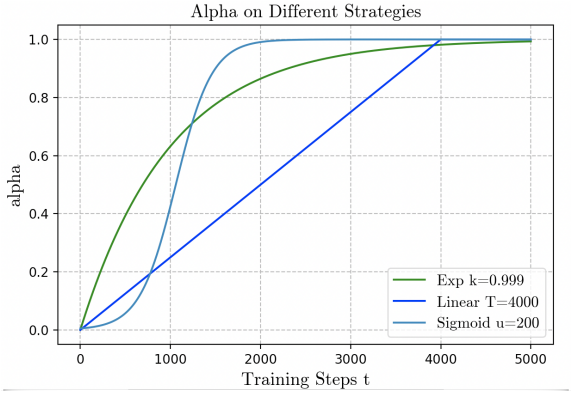

Figure 5: Different growth strategies of α in BranchNorm. Note tht α is clipped to 1.0 for all strategies.

BranchNorm demonstrat comparable performance with various existing works, *e.g.,* Mask Attention (Fan et al., 2021), Dynamic Convolutions (Wu et al., 2019) and Magneto (Wang et al., 2023). Specifically, BranchNorm achieves the best results on two out of three metrics among all models. On deeper structures, such as 200L-200L, BranchNorm exhibit a more significant performance improvement, with a 0.7∼2.0 ROUGE score increase relative to the strong baseline, DeepNorm.

7 Analysis

In this section, we take the machine translation task as an example and conduct analytical experiments with a 200-layer (*i.e.,* 100L-100L) Transformer on WMT14 En-Fr dataset. We first verify the robustness of BranchNorm to hyperparameters, and then analyze the decoding speed of deep models with shallow decoders, and finally analyze the issue of parameter redundancy.

7.1 Hyperparameter Sensitivity

Effects of Different Growing Strategies of α. We investigate the effects of various growing strategies including the default linear strategy, which is illustrated in Figure 5. For the 100L-100L Transformers on the WMT14 En-Fr, we respectively obtain 44.20, 44.15, and 44.21 BLEU on the linear, exp, and sigmoid strategies, indicating that our method is robust to strategy variants, therefore, we use the simplest linear strategy in all experiments.

Effects of Different T. We conduct experiments to evaluate the effect of varying the different maximum norm step T for the above linear strategy in Equation (8). A larger value of T corresponds to a slower degradation of BranchNorm to the vanilla

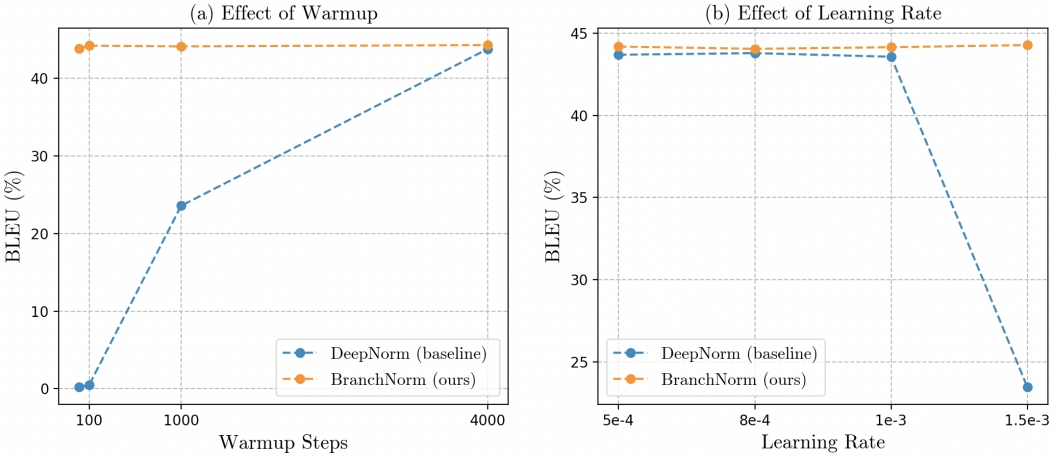

Figure 6: Effects of key hyperparameters (*i.e.,* warmup and learning rate) on training 100L-100L models on the WMT 2014 En-Fr dataset.

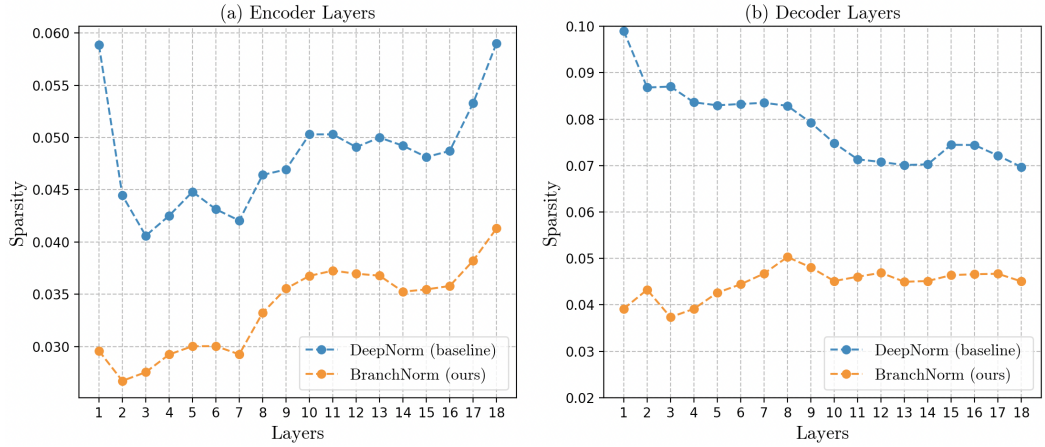

Figure 7: The sparsity of activation function of DeepNorm and BranchNorm models. The BranchNorm model is sparser than the DeepNorm one in all layers.

Post-LN, and generally yield a more stable training process. We vary $T \in [100, 400, 4000, 20000]$ for the 100L-100L Transformers on the WMT14 En-Fr. Consequently, we respectively obtain 43.97, 44.05, 44.24 and 44.15 BLEU for the above settings. Overall, BranchNorm is insensitive to the variation of T, thus we finally chose the best-performing setting (*i.e.,* $T = 4000$) for all experiments.

Effects of Different Warmup and Learning Rate. We explore the effects of key hyperparameters that are directly related to training stability, namely, warmup and learning rate. Results of the 100L-100L Transformer on WMT14 En-Fr dataset are shown in Figure 6. Our observations indicate that BranchNorm is able to stably train a 200-layers Transformer without the use of warmup, and exhibits better tolerance for larger learning rates when compared to DeepNorm.

7.2 Comparing with wide models of similar parameter size

Considering that the training data scale of LLMs is much larger than the data used in the deep models in our experiments, it is not easy to make a direct comparison between the two lines of models. Therefore, we trained wide and deep models with approximately the same number of parameters on the WMT14 En-Fr dataset, with the specific results in Table 5. We observed that extremely deep models have certain performance advantages over wide models under the similar number of parameters.

7.3 Deep Encoders and Shallow Decoders

It is well known that the decoding speed of the Transformer is primarily related to the number of decoder layers. In order to enhance the deep models for more practical applications, we further use

Systems	Hidden Size	Layers	#Parameters	BLEU	
Post-Ins	2048	1638	12L-12L	1.7B	43.41
DeepNorm	512	2048	250L-250L	1.8B	43.87
BranchNorm	512	2048	250L-250L	1.8B	44.30

Table 5: Comparing deep models with wide models of similar parameter size.

Models	**#Para.**	**BLEU**	**Speed (token/s)**
50L-50L	381M	**39.9**	42.7
95L-5L	340M	39.6	**78.6**
250L-250L	1.8B	**44.3**	8.5
475L-25L	1.6B	44.1	**15.9**

Table 6: BLEU scores (%) on the WMT-14 En-Fr test set and decoding speed (test on a V100-40G GPU). The **bolded** scores correspond to the best performance under similar number of parameters. '#Para.' means the number of model parameters. AL-BL refers to a transformer with a A layer encoder and a B layer decoder.

BranchNorm to train models that have a deep encoder and shallow decoder, which is more efficient in inference. The experimental results on the WMT 2014 En-Fr dataset are shown in the following Table 6. We observe that, under similar number of parameter, the deep encoder shallow decoder structure can significantly enhance the decoding speed with little or no damage to the translation quality. For example, compared to the 250L-250L model, the 475L-25L model achieves 1.8x speedup with only 0.2 BLEU decrease. This suggests that the deep encoder shallow decoder is a more practically valuable structure for extremely deep models.

7.4 Parameter Redundancy

Sparsity is quantified by the percentage of nonzero entries after the activation function (Li et al., 2022). As shown in Figure 7, we observe that models trained with BranchNorm had a relatively smaller sparsity than that trained with DeepNorm. To confirm the effect of sparsity on the robustness and generalization of the model, we conduct further experiments on the MTNT (Michel and Neubig, 2018). MTNT (Machine Translation of Noisy Text) consists of noisy comments on Reddit and professionally sourced translations and is a testbed for robust translation. We evaluate two 100L-100L En-Fr models that are trained with DeepNorm and BranchNorm on MTNT. Consequently, BranchNorm shows a significant improvement of 1.0 BLEU over DeepNorm, indicating that our BranchNorm could improve model robustness by increasing the sparsity of the model.

8 Conclusion

In this paper, we first explore the undertraining problem of DeepNorm from a gradient perspective and propose a more flexible approach, namely BranchNorm, which theoretically stabilizes the training with smooth gradient norms at the early stage. Once the early phase of training instability is passed, BranchNorm can then degenerate to a standard Post-LN, thus encouraging better convergence performance. Experiment results on various tasks show that BranchNorm achieves a better trade-off between training stability and converge performance. For instance, BranchNorm improves DeepNorm by up to 1.5 BLEU, 1.1 ROUGE, and 2.1 perplexity on machine translation, text summarization, and language modeling tasks, respectively.

Limitations

The training of deep Transformers generally requires large GPU resources, for example, training a 1,000-layer WMT14 En-Fr translation model requires 1000 GPU days. In addition, deeper decoders can lead to slower inference, and more model architecture design or compression techniques need to be further explored to make deep models practically deployable for applications.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *stat*, 1050:21.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Garrison W. Cottrell, and Julian J. McAuley. 2020. Rezero is all you need: Fast convergence at large depth. *CoRR*, abs/2003.04887.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019.

Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988.

Tri Dao, Daniel Y Fu, Khaled K Saab, Armin W Thomas, Atri Rudra, and Christopher Ré. 2022. Hungry hungry hippos: Towards language modeling with state space models. *arXiv preprint arXiv:2212.14052*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT 2019*, pages 4171–4186.

Zhihao Fan, Yeyun Gong, Dayiheng Liu, Zhongyu Wei, Siyuan Wang, Jian Jiao, Nan Duan, Ruofei Zhang, and Xuan-Jing Huang. 2021. Mask attention networks: Rethinking and strengthen transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1692–1701.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org.

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 Pérez, Jimmy Ba, and Maksims Volkovs. 2020. Improving transformer optimization through better initialization. In *ICML 2020*, 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*.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature*, 521(7553):436–444.

Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, et al. 2022. Large models are parsimonious learners: Activation sparsity in trained transformers. *arXiv preprint arXiv:2210.06313*.

Junyang Lin, An Yang, Jinze Bai, Chang Zhou, Le Jiang, Xianyan Jia, Ang Wang, Jie Zhang, Yong Li, Wei Lin, Jingren Zhou, and Hongxia Yang. 2021. M6-10T: A sharing-delinking paradigm for efficient multi-trillion parameter pretraining. *CoRR*, abs/2110.03888.

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 (EMNLP)*, pages 5747–5763, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Paul Michel and Graham Neubig. 2018. MTNT: A testbed for machine translation of noisy text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 543–553, Brussels, Belgium. Association for Computational Linguistics.

Toan Q. Nguyen and Julian Salazar. 2019. Transformers without tears: Improving the normalization of self-attention. *CoRR*, abs/1910.05895.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Demonstrations*, pages 48–53. Association for Computational Linguistics.

Weizhen Qi, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. 2020. ProphetNet: Predicting future n-gram for sequence-to-SequencePre-training. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2401–2410, Online. Association for Computational Linguistics.

Sam Shleifer, Jason Weston, and Myle Ott. 2021. Normformer: Improved transformer pretraining with extra normalization. *CoRR*, abs/2110.09456.

Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. 2022. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*.

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

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. 2022a. DeepNet: Scaling Transformers to 1,000 layers. *CoRR*, abs/2203.00555.

Hongyu Wang, Shuming Ma, Shaohan Huang, Li Dong, Wenhui Wang, Zhiliang Peng, Yu Wu, Payal Bajaj, Saksham Singhal, Alon Benhaim, Barun Patra, Zhun Liu, Vishrav Chaudhary, Xia Song, and Furu Wei. 2022b. Foundation Transformers. *CoRR*, abs/2210.06423.

Hongyu Wang, Shuming Ma, Shaohan Huang, Li Dong, Wenhui Wang, Zhiliang Peng, Yu Wu, Payal Bajaj, Saksham Singhal, Alon Benhaim, Barun Patra, Zhun Liu, Vishrav Chaudhary, Xia Song, and Furu Wei. 2023. Magneto: A foundation transformer. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 36077–36092. PMLR.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning deep transformer models for machine translation. In *ACL 2019*, pages 1810–1822.

Weizhi Wang, Zhirui Zhang, Yichao Du, Boxing Chen, Jun Xie, and Weihua Luo. 2021. Rethinking zero-shot neural machine translation: From a perspective of latent variables. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4321–4327, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*.

Biao Zhang, Ivan Titov, and Rico Sennrich. 2019a. Improving deep transformer with depth-scaled initialization and merged attention. In *EMNLP-IJCNLP 2019*, pages 898–909.

Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. 2019b. Fixup initialization: Residual learning without normalization. In *ICLR 2019*.

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. *Advances in Neural Information Processing Systems*, 34:16410–16422.

A Theoretical Proofs

A.1 Gradients of Post-LN

Given a Transformer with L sub-layers and the training loss $\mathcal{E}$, the gradient for the l-th sub-layer is calculated by the chain rule:

$$\frac{\partial \mathcal{E}}{\partial x_l} = \frac{\partial \mathcal{E}}{\partial x_L}\frac{\partial x_L}{\partial x_l} \tag{10}$$

Recursively decomposing $\frac{\partial x_L}{\partial x_l}$ in the above equation, we have:

$$\frac{\partial x_L}{\partial x_l} = \frac{\partial x_L}{\partial x_{L-1}}\frac{\partial x_{L-1}}{\partial x_{L-2}}\cdots\frac{\partial x_{l+1}}{\partial x_l} \tag{11}$$

Given the Post-LN calculate the x_{l+1} as :

$$x_{l+1} = LN(x_l + \mathcal{F}(x_l;\theta_l)) \tag{12}$$

If we name the output of residual connection as $y_l = x_l + \mathcal{F}(x_l;\theta_l)$, we can calculate the partial derivatives of two adjacent layers as:

$$\begin{aligned}\frac{\partial x_{l+1}}{\partial x_l} &= \frac{\partial x_{l+1}}{\partial y_l}\frac{\partial y_l}{\partial x_l} \\ &= \frac{\partial \mathrm{LN}(y_l)}{\partial y_l}\left(1 + \frac{\partial \mathcal{F}(x_l;\theta_l)}{\partial x_l}\right)\end{aligned} \tag{13}$$

We put Equation (13) and Equation (11) into Equation (10) and get:

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial x_l} = \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1}\frac{\partial \mathrm{LN}(y_k)}{\partial y_k}}_{LN} \times \\ \underbrace{\prod_{k=l}^{L-1}\left(1 + \frac{\partial \mathcal{F}(x_k;\theta_k)}{\partial x_k}\right)}_{residual}\end{aligned} \tag{14}$$

the above gradient consists of three terms and the last two items are multiplications with respect to the number of model layers L. Once L get larger, the gradient of Post-LN will face the risk of vanishing or exploding.

A.2 Gradients of DeepNorm

DeepNorm rescales the residual branch with a scalar multiplier $\alpha > 1$, and calculates the sub-layer as follows:

$$x_{l+1} = LN(\alpha x_l + \mathcal{F}(x_l;\theta_l)) \tag{15}$$

Follow the above process in A.1, we have the gradient of DeepNorm:

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial x_l} = \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1}\left(\frac{\partial \mathrm{LN}(\alpha x_k + \mathcal{F}(x_k;\theta_l))}{\partial(\alpha x_k + \mathcal{F}(x_k;\theta_l))}\right)}_{LN} \times \\ \underbrace{\prod_{k=l}^{L-1}\left(\alpha + \frac{\partial \mathcal{F}(x_k;\theta_k)}{\partial x_k}\right)}_{residual}\end{aligned} \tag{16}$$

Given that DeepNorm assigns a relative larger value for α to make it to amplify the output percentage of residual connections. Here, we introduce an assumption to simplify the derivation: If α gets

large enough, we can approximate the above equation as follows:

$$\frac{\partial \mathcal{E}}{\partial x_l} \approx \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1} \left(\frac{\partial \mathrm{LN}\left(\alpha x_k\right)}{\partial \left(\alpha x_k\right)} \right)}_{LN} \times \underbrace{\prod_{k=l}^{L-1} \alpha}_{residual} \tag{17}$$

We let $z_k = \alpha x_k$ and use the chain rule, then get:

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial x_l} &= \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1} \left(\frac{\partial \mathrm{LN}\left(z_k\right)}{\partial x_k} \times \frac{\partial x_k}{\partial z_k} \right)}_{LN} \times \underbrace{\prod_{k=l}^{L-1} \alpha}_{residual} \\ &= \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1} \left(\frac{\partial \mathrm{LN}\left(x_k\right)}{\partial x_k} \times \frac{1}{\alpha} \right)}_{LN} \times \underbrace{\prod_{k=l}^{L-1} \alpha}_{residual} \\ &= \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1} \frac{\partial \mathrm{LN}\left(x_k\right)}{\partial x_k}}_{LN} \end{aligned} \tag{18}$$

When compared with the gradient of Post-LN in Equation (14), DeepNorm can approximately eliminate the final multiplication item, and thus mitigate the risk of gradient vanishing or exploding to a certain degree.

A.3 Gradients of BranchNorm

BranchNorm directly rescale the non-residual branch in Transformer and conduct calculations for the l-th sub-layer as:

$$x_{l+1} = LN(x_l + \alpha \mathcal{F}\left(x_l; \theta_l\right)) \tag{19}$$

Similar to the previous analysis process, we can calculate the gradients of BranchNorm as:

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial x_l} = & \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1} \frac{\partial \mathrm{LN}\left(x_k + \alpha \mathcal{F}\left(x_k; \theta_l\right)\right)}{\partial \left(x_k + \alpha \mathcal{F}\left(x_k; \theta_l\right)\right)}}_{LN} \times \\ & \underbrace{\prod_{k=l}^{L-1} \left(1 + \alpha \frac{\partial \mathcal{F}\left(x_k; \theta_l\right)}{\partial x_k} \right)}_{residual} \\ = & \underbrace{\frac{\partial \mathcal{E}}{\partial x_L}}_{irreducible} \times \underbrace{\prod_{k=l}^{L-1} \frac{\partial \mathrm{LN}\left(x_k\right)}{\partial x_k}}_{LN} \quad (\alpha = 0) \end{aligned} \tag{20}$$

BranchNorm can stabilize the gradient norm into while DeepNorm require a relatively strong assumption in Equation (6). Experimentally, in Figure 4, we observe corresponding smoother gradients of BranchNorm at the very beginning of training.

B Hyperparameter

C Why setting $\alpha = 0$ leads to better convergence?

The motivation for the BranchNorm proposed in this paper, i.e., setting $\alpha < 1$ in the early stages of training to reduce the magnitude of the gradient for stability, and removing such restriction on α in the later stages of training and returning to the initial state, i.e., $\alpha = 1$. Below, we specifically discuss the impact of different ranges of α values after T training steps based on preliminary experiments on the 100L-100L model and the WMT14 En-Fr dataset.

What if we set α ¡ 1 after training T steps? We first tried to select a smaller α value (specifically, 0.3, 0.5, and 0.8) after T steps (here $T = 4000$). In terms of training stability, all three α settings can stabilize training, and it is shown that the smaller the α, the smaller the gradient magnitude. At the same number of training steps, the models with $\alpha = 0.3$ and $\alpha = 0.5$ perform significantly worse than the model with $\alpha = 1$, and although this performance gap slightly decreases after final training convergence, it still exists significantly (around 2.0 BLEU). Meanwhile, the model with $\alpha = 0.8$ has a slightly lower convergence performance than the model with $\alpha = 1$. Therefore, we speculate that when $\alpha < 1$, although the training gradient is stable, it faces a similar under-trained problem as DeepNorm, so we decide that the range of α may need to be equal to or greater than 1.

What if we set α ¿ 1 after training T steps? Based on the above experimental conclusions, we naturally wonder if further increasing the value of α to more than 1 can bring further performance improvement (specifically, we tried α=2, 3, 5)? Interestingly, compared to $\alpha = 1$, we did not observe any significant performance improvements. From the perspective of training stability, as α gradually increases, we observed a slight increase in the average gradient norm of the overall training process, and when $\alpha = 3$ or $\alpha = 5$, the model showed a few loss spike phenomenon after the number of training steps exceeded T, indicating that the training stability of the model was challenged to some extent when $\alpha > 1$. In summary, we believe that

Hyperparameters	**Small Scale**	**Medium Scale**	**Large Scale**
Learning rate		5e-4	
Learning rate scheduler		inverse sqrt	
Warm-up updates		4000	
Warm-up init learning rate		1e-7	
Max tokens		128×4096	
Adam ϵ		1e-8	
Adam β		(0.9, 0.98)	
Label smoothing		0.1	
Training updates		100K	
Gradient clipping		0.0	
Dropout	0.4	0.2	0.1
Weight decay		0.0001	
Hidden size		512	
FFN inner hidden size		2048	
Attention heads		8	

Table 7: Hyperparameters for the Transformer$_{base}$ experiments on different data sizes. 'Small Scale': WMT17 En-De and CNN/DailyMail. 'Medium Scale': WMT14 En-Fr and WikiText-103. 'Large Scale': OPUS-100 and MultiUN datasets.

although the setting of $\alpha = 1$ is simple, it is still a hard-to-beat and intuitive (i.e., the setting of vanilla Transformer) experimental setting. And the experimental results on multiple tasks (Tables 1 4) further corroborate that $\alpha = 1$ is a straightforward and reasonable setting.