Understanding the Difficulty of Training Transformers

Liyuan Liu[†‡] **Xiaodong Liu**[‡] **Jianfeng Gao**[‡] **Weizhu Chen**[§] **Jiawei Han**[†]
{ll2, hanj}@illinois.edu , {xiaodl,jfgao,wzchen}@microsoft.com
[†]University of Illinois at Urbana-Champaign
[‡]Microsoft Research
[§] Microsoft Dynamics 365 AI

Abstract

Transformers have been proved effective for many deep learning tasks. Training transformers, however, requires non-trivial efforts regarding carefully designing learning rate schedulers and cutting-edge optimizers (the standard SGD fails to train Transformers effectively). In this paper, we study Transformer training from both theoretical and empirical perspectives. Our analysis reveals that unbalanced gradients are not the root cause of the instability of training. Instead, we identify an *amplification* effect that substantially influences training. Specifically, we observe that for each layer in a multi-layer Transformer model, heavy dependency on its residual branch makes training unstable since it amplifies small parameter perturbations (*e.g.*, parameter updates) and result in significant disturbances in the model output, yet a light dependency limits the potential of model training and can lead to an inferior trained model. Inspired by our analysis, we propose *Admin* (**Ad**aptive **m**odel **in**itialization) to stabilize the training in the early stage and unleash its full potential in the late stage. Extensive experiments show that Admin is more stable, converges faster, and leads to better performance.

1 Introduction

Transformers (Vaswani et al., 2017) have led to a series of breakthroughs in various deep learning tasks (Devlin et al., 2019; Velickovic et al., 2018). They do not contain recurrent connections and can parallelize all computations in the same layer, thus improving effectiveness, efficiency, and scalability. Training Transformers, however, requires extra efforts. For example, although the standard stochastic gradient descent (SGD) is a canonical algorithm for the conventional RNNs and CNNs, it converges to bad/suspicious local optima for Transformers (Zhang et al., 2019b). Moreover, comparing to other neural architectures, removing the warmup stage in Transformer training results in more serious consequences such as model divergence (Popel and Bojar, 2018; Liu et al., 2020). In this paper, we conduct a comprehensive analysis in theoretical and empirical manners to answer the question: *what complicates Transformer training*.

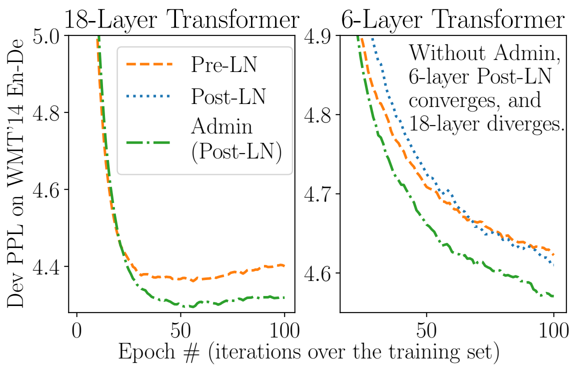

Figure 1: Lacking enough robustness and stability, the 18-Layer Post-LN Transformer training (*i.e.*the original architecture) diverges and is omitted in the left graph. Admin not only stabilizes model training but unleashes model potential for a better performance.

Our analysis starts from the observation: the original Transformer (referred to as Post-LN) is less robust than its Pre-LN variant (Baevski and Auli, 2019; Nguyen and Salazar, 2019). We recognize that gradient vanishing is not the direct reason causing such difference, since fixing this issue alone cannot stabilize Post-LN training. It implies that there exist factors other than unbalanced gradients that influence model training greatly.

With further analysis, we recognize that for each layer, the dependency on its residual branch[1] plays an important role in training stability. First, we find

Work in progress.

[1]For a residual layer $x + f(x)$, its shortcut output refers to x, its residual branch refers to $f(x)$, and the dependency on its residual branch refers to $\frac{\mathrm{Var}[f(x)]}{\mathrm{Var}[x+f(x)]}$.

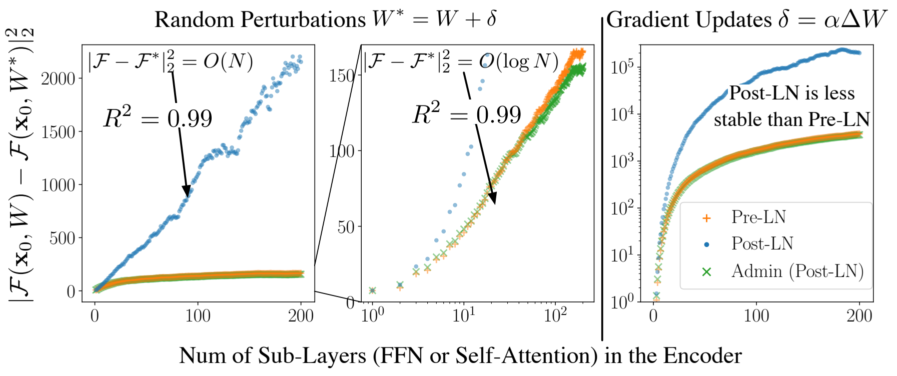

Figure 2: Output changes of Pre-LN, Post-LN and Admin encoders (1 to 100 layers, each has 2 sub-layers), *i.e.*, $|\mathcal{F}(\mathbf{x}_0, W) - \mathcal{F}(\mathbf{x}_0, W^*)|_2^2$ and W^* is calculated with random perturbations (left) or gradient updates (right).

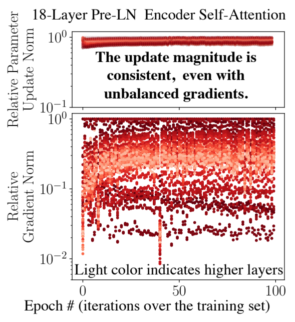

Figure 3: Histogram of relative norm of gradient and $|W_{i+1} - W_i|$ where W_i is the checkpoint saved after training for i epochs.

that a Post-LN layer has a heavier dependency on its residual branch than a Pre-LN layer. For example, at initialization, a Pre-LN layer has roughly the same dependency on its residual branch and any previous layer, whereas a Post-LN layer has a stronger dependency on its residual branch (more discussions are presented in Section 4.1). We find that strong dependencies of Post-LN amplify fluctuations brought by parameter changes and destabilize the model training (as in Theorem 1 and Figure 2). On the other hand, the loose reliance on residual branches in Pre-LN generally limits the potential of the training algorithm, and often produce an inferior model.

In light of our analysis, we propose Admin, an adaptive initialization method for training Post-LN Transformer models, which retains the merits of Pre-LN stability without hurting the performance. It restricts the layer dependency on its residual branches in the early stage and unleashes the model potential in the late stage. Admin is more stable, converges faster, and performs better in extensive experiments on IWSLT'14 De-En and WMT'14 En-De[2].

2 Preliminaries

Transformer Architectures and Notations. The Transformer architecture contains two types of sub-layers, *i.e.*, Attention sub-layers and Feedforward (FFN) sub-layers. They are composed of mainly three basic modules (Vaswani et al., 2017), *i.e.*, Layer Norm (f_{LN}), Multi-head Attention (f_{ATT}) and Feedforward Network (f_{FFN}). As illustrated in Figure 4, the Pre-LN Transformer and the Post-LN Transformer organize these modules differently. For example, a Pre-LN encoder organizes the Self-Attention sub-layer as: $\mathbf{x}^{(pe)}_{2i-1} = \mathbf{x}^{(pe)}_{2i-2} + f_{\text{S-ATT}}(f_{\text{LN}}(\mathbf{x}^{(pe)}_{2i-2}))$ and a Post-LN encoder as $\mathbf{x}^{(oe)}_{2i-1} = f_{\text{LN}}(\mathbf{x}^{(oe)}_{2i-2} + f_{\text{S-ATT}}(\mathbf{x}^{(oe)}_{2i-2}))$ where $\mathbf{x}^{(\cdot)}_{2i-2}$ is the input of the i-th Transformer layer and $\mathbf{x}^{(\cdot)}_{2i-1}$ is the output of the i-th Self-Attention sub-layer. Here, we refer $f_{\text{S-ATT}}(f_{\text{LN}}(\mathbf{x}^{(pe)}_{2i-2}))$ and $f_{\text{S-ATT}}(\mathbf{x}^{(oe)}_{2i-2})$ as the residual branches and their outputs as the residual outputs, in contrast to layer/sub-layer outputs, which integrates residual outputs and shortcut outputs. Notation elaborations are shown in Figure 4. In particular, we use superscript to indicate network architectures (*i.e.*, the Pre-LN Encoder), use subscript to indicate layer indexes (top layers have larger indexes), all inputs and outputs are formulated as Sequence-Len $\times$ Hidden-Dim.

Layer Norm. Layer norm (Ba et al., 2016) plays a key role in the Transformer architecture. It is defined as $f_{\text{LN}}(\mathbf{x}) = \boldsymbol{\gamma}\frac{\mathbf{x}-\mu}{\sigma} + \boldsymbol{\nu}$, where μ and σ are the mean and standard deviation of $\mathbf{x}$.

Feedforward Network. Transformers use two-layer perceptrons as feedforward networks, *i.e.*, $f_{\text{FFN}}(\mathbf{x}) = \phi(\mathbf{x}W^{(1)})W^{(2)}$, where $\phi(\cdot)$ is the non-linear function[3] and $W^{(\cdot)}$ are parameters.

Multi-head Attention. Multi-head Attention allows the network to have multiple focuses and has shown to be effective for many tasks (Chen et al., 2018). It is de-

[2]Code is released at: `https://github.com/LiyuanLucasLiu/Transforemr-Clinic`

[3]In our analysis, we use ReLU as the activation function while Admin can be applied to other non-linear functions.

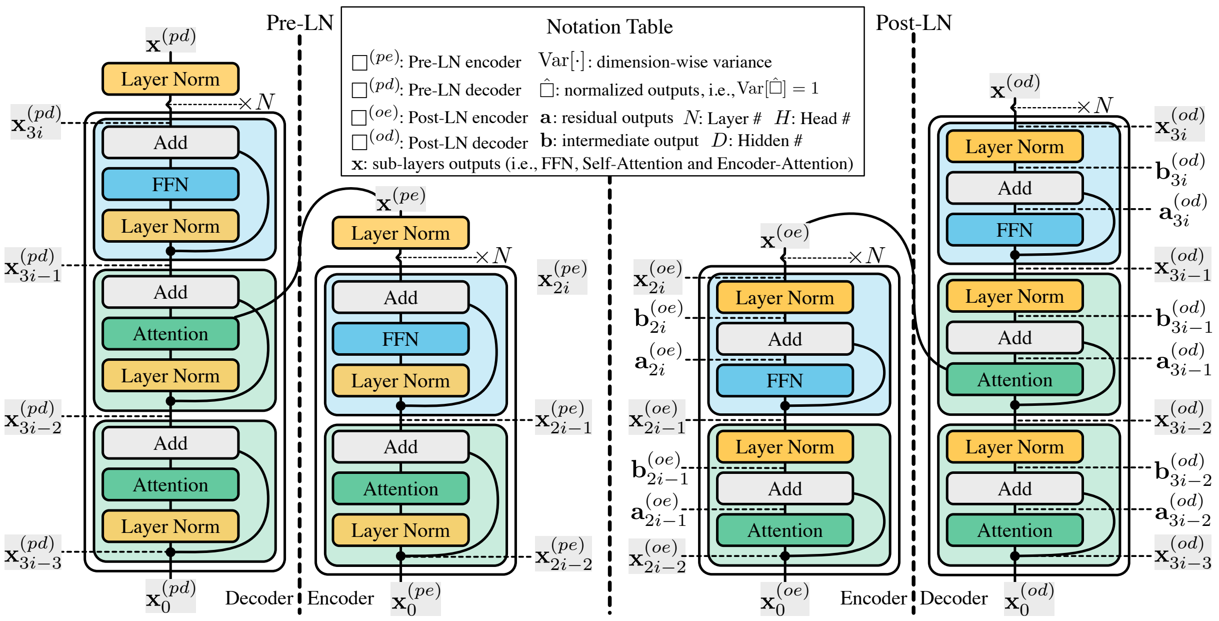
Pre-LN
Post-LN
Notation Table
$\square^{(pe)}$: Pre-LN encoder
$\square^{(pd)}$: Pre-LN decoder
$\square^{(oe)}$: Post-LN encoder
$\square^{(od)}$: Post-LN decoder
Var[·]: dimension-wise variance
$\hat{\square}$: normalized outputs, i.e., Var[$\hat{\square}$] = 1
a: residual outputs N: Layer # H: Head #
b: intermediate output D: Hidden #
x: sub-layers outputs (i.e., FFN, Self-Attention and Encoder-Attention)
Layer Norm
Add
FFN
Attention
×N
Decoder
Encoder
$\mathbf{x}^{(pd)}$
$\mathbf{x}_0^{(pd)}$
$\mathbf{x}^{(pe)}$
$\mathbf{x}_0^{(pe)}$
$\mathbf{x}^{(oe)}$
$\mathbf{x}_0^{(oe)}$
$\mathbf{x}^{(od)}$
$\mathbf{x}_0^{(od)}$

Figure 4: The Architecture and notations of Pre-LN Transformers (Left) and Post-LN Transformers (Right).

fined as (with H heads): $f_{\text{ATT}}(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \sum_{h=1}^{H} f_s(\mathbf{q}W_h^{(Q)}W_h^{(K)}\mathbf{k}^T)\mathbf{v}W_h^{(V_1)}W_h^{(V_2)}$, where f_s is the row-wise softmax function and $W_h^{(\cdot)}$ are parameters. $W_h^{(Q)}$ and $W_h^{(V_1)}$ are $D \times \frac{D}{H}$ matrices, $W_h^{(K)}$ and $W_h^{(V_2)}$ are $\frac{D}{H} \times D$ matrices, where D is the hidden state dimension. Parameters without subscript refer the concatenation of all H-head parameters, *e.g.*, $W^{(Q)} = [W_1^{(Q)}, \cdots, W_H^{(Q)}]$. Here, this module is used in two different manners: Encoder-Attention (*i.e.*, $f_{\text{E-ATT}}(\mathbf{x}) = f_{\text{ATT}}(\mathbf{x}, \mathbf{x}^{(\cdot e)}, \mathbf{x}^{(\cdot e)})$, where $\mathbf{x}^{(\cdot e)}$ is encoder outputs) and Self-Attention (*i.e.*, $f_{\text{S-ATT}}(\mathbf{x}) = f_{\text{ATT}}(\mathbf{x}, \mathbf{x}, \mathbf{x})$).

3 Unbalanced Gradients

Here, we strive to answer the question: *what complicates Transformer training*. Our analysis starts from the observation: Pre-LN training is more robust than that of Post-LN while Post-LN is more likely to reach a better performance than Pre-LN. For example, in a parameter grid search (as in Figure 10), Pre-LN converges in all 15 settings and Post-LN diverges in 7 out of 15 settings; when Post-LN converges, it outperforms Pre-LN in 7 out of 8 settings. We seek to reveal the underlying factor that destabilizes Post-LN and restricts the performance of Pre-LN.

In this section, we focus on the unbalanced gradients (*e.g.*, gradient vanishing). We find that, although Post-LN suffers from gradient vanishing and Pre-LN does not, gradient vanishing is not the direct reason causing the instability of Post-LN. Specifically, we first theoretically and empirically establish that only Post-LN decoders suffer from gradient vanishing and Post-LN encoders do not. We then observe that fixing the gradient vanishing issue alone cannot stabilize training.

3.1 Gradients at Initialization

As gradient vanishing can hamper convergence from the beginning, it has been regarded as the major issue leading towards unstable training. Also, recent studies show that this issue exists in the Post-LN Transformer, even after using residual connections (Xiong et al., 2019). Below, we establish that only Post-LN decoders suffer from the gradient vanishing, and neither Post-LN encoders, Pre-LN encoders nor Pre-LN decoders.

We use $\Delta\mathbf{x}$ to denote gradients, *i.e.*, $\Delta\mathbf{x} = \frac{\partial\mathcal{L}}{\partial\mathbf{x}}$ where $\mathcal{L}$ is the training objective. Following previous studies (Bengio et al., 1994; Glorot and Bengio, 2010; He et al., 2015; Saxe et al., 2013a), we analyze the gradient distribution at the very beginning of training. As established in Theorem 2 and Remark 1 (detailed derivations are included in Appendix A), neither Post-LN Encoders, Pre-LN Encoders nor Pre-LN Decoders suffer from gradient vanishing. In other words, only the Encoder-Attention layer in Post-LN suffers from gradient vanishing. Empirical studies are further conducted for verification. At initialization, we calculate

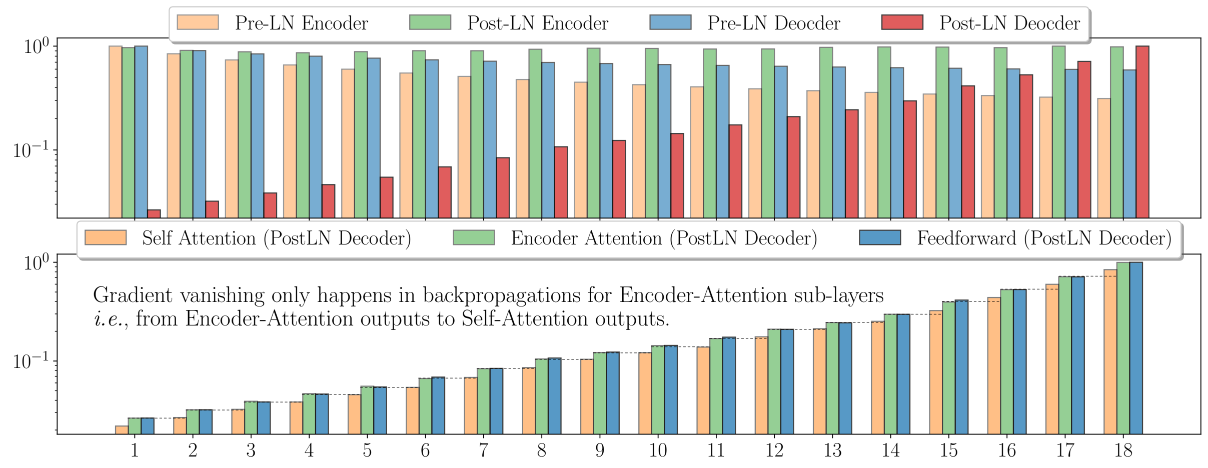

Figure 5: Relative gradient norm histogram (on a log scale) of 18-layer Transformers on the WMT'14 En-De dataset, *i.e.*, the gradient norm of sub-layer outputs, scaled by the largest gradient norm in the same network.

Encoder	**Decoder**	**Gradient**	**Training**
Post-LN	Post-LN	Varnishing	Diverged
Post-LN	Pre-LN	~~Varnishing~~	Diverged
Pre-LN	Pre-LN	~~Varnishing~~	Converged

Table 1: Changing decoders from Post-LN to Pre-LN fixes gradient vanishing, but does not stabilize model training successfully. Encoder/Decoder have 18 layers.

$||\Delta \mathbf{x}_i^{(\cdot)}||_2$ for 18-layer Transformers[4] and visualize $\frac{||\Delta \mathbf{x}_i^{(\cdot)}||_2}{\max_j ||\Delta \mathbf{x}_j^{(\cdot)}||_2}$ in Figure 5. It verifies that only Post-LN decoders suffer from the gradient vanishing. It also shows that the gradient vanishing happens in backpropagations from Encoder-Attention sub-layer outputs to its inputs (*i.e.*, Self-Attention sub-layer outputs).

3.2 Impact of the Gradient Vanishing

Now, we analyze whether gradient vanishing is the direct cause of training instability.

As in Section 3.1, only Post-LN decoders suffer from gradient vanishing, but not Post-LN encoders. Thus, we combine a Post-LN encoder and a Pre-LN decoder to construct a hybrid Transformer, which does not suffer from gradient vanishing. As shown in Table 1, fixing gradient vanishing alone (*i.e.*, changing Post-LN decoders to Pre-LN decoders) fails to stabilize model training. It implies that the gradient vanishing issue is not the direct cause of the unstable Post-LN training.

Moreover, we observe that gradients of all attention modules are unbalanced and are hard to be neutralized. Also, we find this issue is largely addressed by adaptive optimizers. For example, as in Figure 3, adaptive optimizers successfully assign different learning rates to different parameters and lead to consistent update magnitudes even with unbalanced gradients. It explains why the standard SGD fails in training Transformers (*i.e.*, lacking the ability to handle unbalanced gradients) and necessitates the use of adaptive optimizers. More discussions are included in Appendix A.4.

4 Instability from Amplification Effect

We find that unbalanced gradients are not the root cause of the instability of Post-LN, which implies the existence of other factors influencing model training. Now, we go beyond gradient vanishing and introduce the *amplification* effect. Specifically, we first examine the difference between Pre-LN and Post-LN, including their behaviors in the early-stage and late-stage of training. Then, we show that the training instability of Post-LN is attributed to the amplification effect of layer dependency, which intensifies gradient updates and destabilizes training.

4.1 Impact of Layer Norms Positions

As described in Section 2, both Pre-LN and Post-LN employ layer norm to regularize inputs and outputs. In residual networks, different residual outputs are aggregated and normalized before serving as inputs of other layers (*i.e.*, residual outputs will be scaled to ensure the integrated input to have a consistent variance). To some extend, layer norm treats the variance of residual outputs as weights

[4]Note if $\mathbb{E}[\Delta \mathbf{x}_{i-1}^{(p\cdot)}] = 0$, $\mathrm{Var}[\Delta \mathbf{x}_{i-1}^{(p\cdot)}] \approx |\Delta \mathbf{x}_{i-1}^{(p\cdot)}|_2^2$.

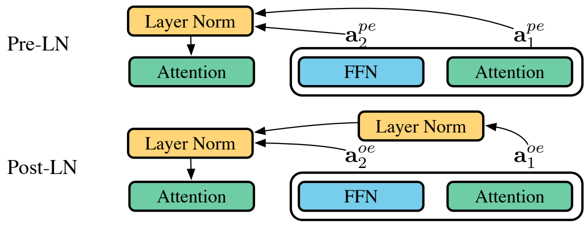

Figure 6: The major difference between Pre-LN and Post-LN is the position of layer norms.

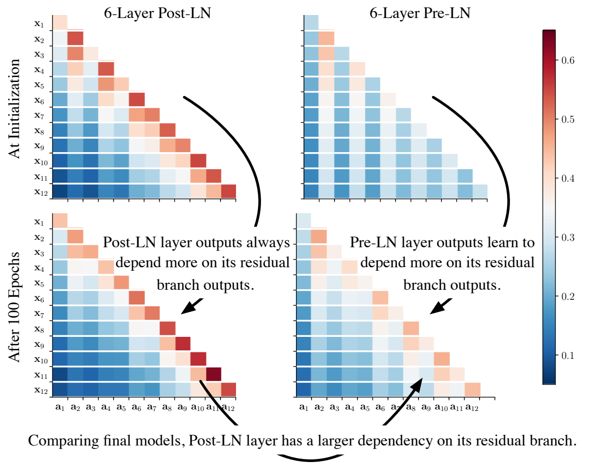

Figure 7: $\beta_{i,j}$ in 6-Layer Post-LN and Pre-LN on the WMT-14 En-De dataset (contains 12 sub-layers).

to average them. For example, for Post-LN Self-Attention, we have $\mathbf{x}_{2i-1}^{(o\cdot)} = \frac{\mathbf{x}_{2i-2}^{(o\cdot)}+\mathbf{a}_{2i-1}^{(o\cdot)}}{\sqrt{\mathrm{Var}[\mathbf{x}_{2i-2}^{(o\cdot)}]+\mathrm{Var}[\mathbf{a}_{2i-1}^{(o\cdot)}]}}$ at initialization. Larger $\mathrm{Var}[\mathbf{a}_{2i-2}^{(o\cdot)}]$ not only increases the proportion of $\mathbf{a}_{2i-2}^{(o\cdot)}$ in the output but decreases the proportion of other residual outputs (integrated in $\mathbf{x}_{2i-2}^{(o\cdot)}$), which is similar to weights in the weighted average.

The position of layer norms is the major difference between Pre-LN and Post-LN and makes them aggregate residual outputs differently (*i.e.*, using different weights). As in Figure 6, all residual outputs in Pre-LN are only normalized once before feeding into other layers (thus only treating residual output variances as weights); in Post-LN, most residual outputs are normalized more than once and different residual outputs are normalized for different times. For example, if all layers are initialized in the same way, output variances of different Pre-LN residual branches would be similar, and the aggregation would be similar to the simple average. Similarly, for Post-LN, nearby residual outputs are normalized by less times comparing to others, thus having relatively larger weights. We proceed to calculate and analyze these weights to understand the impact of layer norm positions.

First, we use $\widehat{\mathbf{a}}_i$ to refer $\frac{\mathbf{a}_i}{\sqrt{\mathrm{Var}\,\mathbf{a}_i}}$ (*i.e.*, normalized outputs of i-th residual branch) and $\widehat{\mathbf{x}}_i$ to refer $\frac{\mathbf{x}_i}{\sqrt{\mathrm{Var}\,\mathbf{x}_i}}$ (*i.e.*, normalized outputs of i-th layer or normalized inputs of (i+1)-th residual branch). Then, we describe their relationships as $\widehat{\mathbf{x}}_i = \sum_{j\le i}\beta_{i,j}\widehat{\mathbf{a}}_j$, where $\beta_{i,j}$ integrates scaling operations of all layer norms (including $\sqrt{\mathrm{Var}\,\mathbf{a}_i}$). For example, Pre-LN sets $\beta_{i,j} = \frac{\sqrt{\mathrm{Var}[\mathbf{a}_j]}}{\sqrt{\mathrm{Var}[\sum_{k\le i}\mathbf{a}_k]}}$. Intuitively, $\beta_{i,j}$ describes the proportion of j-th residual branch outputs in i-th layer outputs, thus reflects the dependency among layers.

We calculate and visualize $\beta_{i,j}$ in Figure 7. For each Post-LN layer, its outputs rely more on its own residual branch from the initialization to the end. At initialization, Pre-LN layer outputs have roughly the same reliance on all previous residual branches. As the training advances, each layer starts to rely more on its own residual outputs. But comparing to Post-LN, Pre-LN layer outputs in the final model are still less focused on their residual branches.

Intuitively, it is harder for Pre-LN layers to depend too much on their own residual branches. In Pre-LN, layer outputs (*i.e.*, $\mathbf{x}_i^{(p\cdot)}$) are not normalized, and their variances are likely to be larger for higher layers (*i.e.*, if $\mathbf{a}_0$ and $\mathbf{a}_1$ are independent, $\mathrm{Var}[\mathbf{a}_0+\mathbf{a}_1] = \mathrm{Var}[\mathbf{a}_0]+\mathrm{Var}[\mathbf{a}_1]$; also, in our experiments $\mathrm{Var}[\mathbf{x}_i^{(p\cdot)}]$ increases as i becomes larger). Since $\beta_{i,i} = \frac{\sqrt{\mathrm{Var}[\mathbf{a}_i]}}{\sqrt{\mathrm{Var}[\mathbf{x}_{i-1}^{(p\cdot)}+\mathbf{a}_i]}}$, $\beta_{i,i}$ is likely to be smaller for higher layers, which restricts i-th layer outputs from depending too much on its own residual branch and inhibits the network from reaching its full potential. In other words, Pre-LN restricts the network from being too deep (*i.e.*, if it is hard to distinguish $\mathbf{x}_i^{(p\cdot)}$ and $\mathbf{x}_{i+1}^{(p\cdot)}$, appending one layer would be similar to double the width of the last layer), while Post-LN allows the network to have the choice of being wider or deeper.

4.2 Amplification Effect at Initialization

Although depending more on residual branches allows the model to have a larger potential, it amplifies the fluctuation brought by parameter changes. For a network $\widehat{\mathbf{x}} = \mathcal{F}(\mathbf{x}_0, W)$ where $\mathbf{x}_0$ is the model input and W is the parameter, the output change caused by parameter perturbations is $\mathrm{Var}[\mathcal{F}(\mathbf{x}_0, W)-\mathcal{F}(\mathbf{x}_0, W^*)]$, where $W^* = W+\delta$. Its relationship with N is described in Theorem 1 and the derivation is elaborated in Appendix B.

THEOREM 1. — Consider a N-layer Transformer $\widehat{\mathbf{x}} = \mathcal{F}(\widehat{\mathbf{x}}_0, W)$, where $\widehat{\mathbf{x}}_0$ is the input and W is the parameter. If the layer dependency stays the same after a parameter change (*i.e.*, $\beta_{i,j}$ has the same value after changing W to W^*, where W is randomly initialized and $\delta = W^* - W$ is independent to W), the output change (*i.e.*, $\mathrm{Var}[\mathcal{F}(\mathbf{x}_0, W) - \mathcal{F}(\mathbf{x}_0, W^*)]$) can be estimated as $\sum_{i=1}^{N} \beta_{i,i}^2 C$ where C is a constant.

If $\mathrm{Var}[\mathbf{a}_i]$ is the same for all layers, Pre-LN sets $\beta_{i,i}^2$ as $1/i$ and Post-LN sets $\beta_{i,i}^2$ as a constant. Thus, we have Corollary 1 and 2 as below.

COROLLARY 1. — For N-layer Pre-LN $\mathcal{F}$, we have $\mathrm{Var}[\mathcal{F}(\mathbf{x}_0, W) - \mathcal{F}(\mathbf{x}_0, W^*)] = O(\log N)$.

COROLLARY 2. — For N-layer Post-LN $\mathcal{F}$, we have $\mathrm{Var}[\mathcal{F}(\mathbf{x}_0, W) - \mathcal{F}(\mathbf{x}_0, W^*)] = O(N)$.

They show that, since Post-LN relies more on residual branches comparing to Pre-LN (*i.e.*, has a larger $\beta_{i,i}^2$), the perturbation is amplified to a larger magnitude. To empirically verify these relationships, we calculate $|\mathcal{F}(\mathbf{x}_0, W) - \mathcal{F}(\mathbf{x}_0, W^*)|_2^2$ for Pre-LN and Post-LN and visualize the results in Figure 2. In Corollary 2, N is linearly associated with $|\mathcal{F} - \mathcal{F}^*|_2^2$ for Post-LN; and in Corollary 1, $\log N$ is linearly associated with $|\mathcal{F} - \mathcal{F}^*|_2^2$ for Pre-LN. These relationships match the observation in our experiments (as in Figure 2). For further verification, we measure their correlation magnitudes by R^2 and find $R^2 = 0.99$ in both cases.

Moreover, we replace the random noise δ with optimization updates (*i.e.*, setting $W^* = W - f_{\text{opt}}(\Delta W)$, where $f_{\text{opt}}(\cdot)$ is the Adam optimizer) and visualize the output shifts in Figure 2. The output shift $|\mathcal{F} - \mathcal{F}^*|_2^2$ for Post-LN is larger than Pre-LN by multiple magnitudes.

Intuitively, large output shifts would destabilize the training (Li et al., 2018). Also, as elaborated in Appendix B, the constant C in Theorem 1 is related to network derivatives, thus would be smaller as training advances, which explains why warmup is also helpful for the standard SGD. Therefore, we conjecture it is the large output shift of Post-LN results in unstable training. Now we proceed to stabilize Post-LN by controlling the dependency on residual branches in the early stage of training.

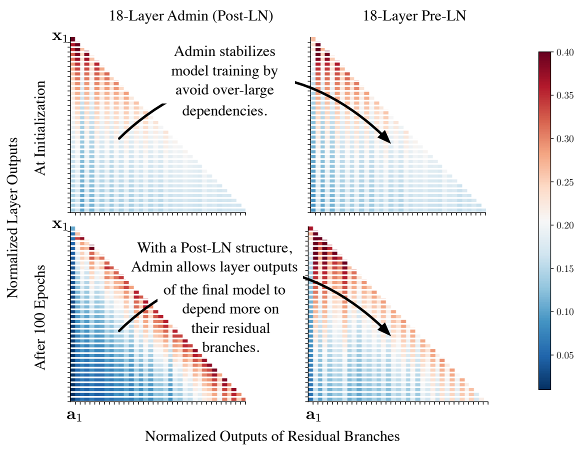

Figure 8: $\beta_{i,j}$ of 18-Layer Admin (Post-LN) and Pre-LN on the WMT-14 En-De dataset.

4.3 *Admin* – Adaptive Model Initialization

In light of our analysis, we add additional parameters (*i.e.*, $\boldsymbol{\omega}$) to control residual dependencies of Post-LN and stabilize training by adaptively initializing $\boldsymbol{\omega}$ to ensure a $O(\log N)$ output change.

Due to different training configurations and model specificities (*e.g.*, different model may use different activation functions and dropout ratios), it is hard to derive a universal initialization method. Instead, we decompose model initialization into two phrases: *Profiling* and *Initialization*. Specifically, Admin adds new parameters $\boldsymbol{\omega}$ and constructs its i-th sub-layer as $\mathbf{x}_i = f_{\text{LN}}(\mathbf{b}_i)$, where $\mathbf{b}_i = \mathbf{x}_{i-1} \cdot \boldsymbol{\omega}_i + f_i(\mathbf{x}_{i-1})$, $\boldsymbol{\omega}_i$ is a D-dimension vector and $\cdot$ is element-wise product. Then the *Profiling* phrase and *Initialization* phrase are:

Profiling. After initializing the network with a standard method (initializing $\boldsymbol{\omega}_i$ as $\mathbf{1}$), conduct forward propagation without parameter updating and record the output variance of residual branches (*i.e.*, calculate $\mathrm{Var}[f_i(\mathbf{x}_{i-1})]$).

Initialization. Set $\boldsymbol{\omega}_i = \sqrt{\sum_{j<i} \mathrm{Var}[f_j(\mathbf{x}_{j-1})]}$ and initialize all other parameters with the same method used in the *Profiling* phrase.

In the early stage, Admin sets $\beta_{i,i}^2$ to approximately $\frac{1}{i}$ and ensures a $O(\log N)$ output change thus stabilizing training. Model training would become more stable in the late stage (the constant C in Theorem 1 is related to parameter gradients) and each layer has the flexibility to adjust $\boldsymbol{\omega}$ and depends more on its own residual branch to calculate the layer outputs. After training finishes, Admin can be reparameterized as the conventional

Table 2: Evaluation Results on WMT14 De-En.

Method	BLEU		
	6-Layer	12-Layer	18-Layer
Post-LN	27.80	Diverged	Diverged
Pre-LN	27.27	28.26	28.26
Admin	**27.90**	**28.58**	**28.80**

Post-LN structure (*i.e.*, removing ω). More implementation details are elaborated in Appendix C.

To verify our intuition, we calculate the layer dependency of 18-Layer models and visualize the result in Figure 8. Figure 7 and 8 show that Admin avoids over-large dependencies at initialization and unleashes the potential to make the layer outputs depend more on their residual outputs in the final model. Moreover, we visualize the output change of Admin in Figure 2. Benefiting from the adaptive initialization, the output change of Admin gets roughly the same increase speed as Pre-LN, even constructed in the Post-LN manner. Also, although Admin is formulated in a Post-LN manner and suffers from gradient vanishing, 18-layer Admin successfully converges and outperforms 18-layer Pre-LN (as in Table 2). These evidences support our intuition that the large dependency on residual branches amplifies the output fluctuation and destabilizes training.

5 Experiments

We conduct experiments on two machine translation datasets, *i.e.*, IWSLT'14 De-En and WMT'14 En-De. The detailed experimental configurations are elaborated in Appendix D.

5.1 Performance Comparison

We use BLEU as the evaluation matric and summarize the model performance in Table 2 and Table 3. For the WMT'14 dataset, experiments are conducted using the Transformer-base model with 6, 12 or 18 layers. Admin achieves a better performance than Post-LN and Pre-LN in all three settings. Specifically, 12-Layer and 18-Layer Post-LN diverges without the adaptive initialization. Admin obtains comparable performance with Post-LN in the 6-layer setting and converges well in both the 12-layer and the 18-layer settings. Pre-LN also converges in all settings, but it results in a sub-optimal performance, which verifies our intuition that the Pre-LN structure limits the model potential. As depicted in Figure 1 and Figure 9, although the 6-layer Pre-LN converges faster than Post-LN, its final performance is worse than Post-LN. In contrast, Admin not only achieves the same convergence speed with Pre-LN in the early stage, but reaches a good performance in the late stage.

Table 3: Performance on IWSLT14 De-En (Transformer models are 6-layer Transformer-small models).

Method	BLEU
Post-LN (Vaswani et al., 2017)	34.60
DynamicConv (Wu et al., 2019)	35.2
Post-LN	35.64±0.23
Pre-LN	35.50±0.04
Admin	35.67±0.15

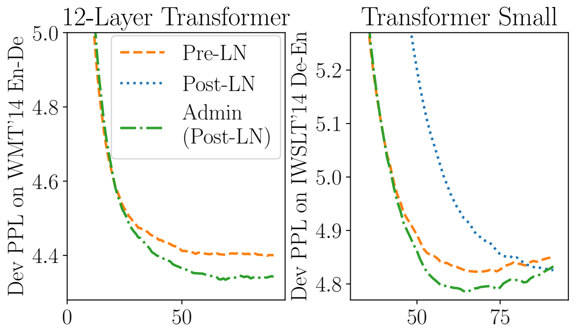

Figure 9: Development PPL on the WMT'14 En-De dataset and the IWLST'14 De-En dataset.

For the IWSLT'14 dataset, we use the Transformer-small model for training. We observe that all methods perform similarly and Admin outperforms the other two by a small margin. Comparing to the WMT'14 results, it verifies that the training stability is related to the number of layers. For shallow networks, the stability difference between Post-LN and Pre-LN is not significant, and all architectures lead to a similar performance. Besides, we find that the attention dropout and the activation dropout have a large impact on the model performance. Specifically, via setting the attention dropout ratio and relu dropout ratio to 0.1, we are able to improve the Post-LN performance from 34.60 (reported) to 35.64 (average of five runs).

5.2 Comparing to other Initializations

We further compare our methods with two initialization methods, *i.e.*, FixUp (Zhang et al., 2019a) and LookLinear (Balduzzi et al., 2017a). Specifically, we conduct experiments with 18-layer Transformers on the WMT'14 De-En dataset. In our experiments, we observe that both FixUp (without using layer normalization) and LookLinear (with Post-LN) leads to divergent training. With further analysis, we find that, the half-precision training and dropout could be the reason destabilizing FixUp, due to the lack of layer normalization.

5.3 Connection to Warmup

Previous work (Liu et al., 2020) establishes that the need of warmup comes from the unstable adaptive learning rates in the early stage. Still, it is observed that removing the warmup phrase results in more serious consequences for Transformers than other architectures. Also, warmup is found to be useful for the vanilla SGD (Xiong et al., 2019).

In Theorem 2, we establish that

$$\mathrm{Var}[\mathcal{F}(\mathbf{x}_0, W) - \mathcal{F}(\mathbf{x}_0, W^*)] \approx \sum_{i=1}^{N} \beta_{i,i}^2 C$$

where $C = \mathrm{Var}[\mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i) - \mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i^*)]$. In the early stage of training, the network has larger parameter gradients and thus larger C. Therefore, same parameter shifts would result in larger output changes in the early stage than in the late stage. Thus, warmup relieves the output change and helps to stabilize training.

To further verify our intuitions, we remove the warmup phrase and conduct a grid search on RAdam hyper-parameters (Liu et al., 2020). Results are visualized in Figure 10. It shows that Post-LN is less robust to the choice of learning rates. Specifically, Post-LN diverges with larger learning rates or smaller β_2 (smaller β_2 use less samples to estimate adaptive learning rates), while Admin and Pre-LN are more robust. At the same time, we extend the warmup phrase from 8 thousand updates to 16, 24, and 32 thousand updates and find the training of 18-layer Post-LN still converges to bad/suspicious local optima. It shows that, the large output shift of Post-LN is not always neutralized by the learning rate warmup. Intuitively, the large output shift not only requires a small learning rate but also unsmoothes the loss surface (Li et al., 2018) and complicates training. Since warmup stabilizes the training without smoothing the loss surface, it fails to train deeper Transformer networks. On the other hand, Admin not only stabilizes training but simplifies the training by initializing from the area with a smooth loss surface, thus leading to better training.

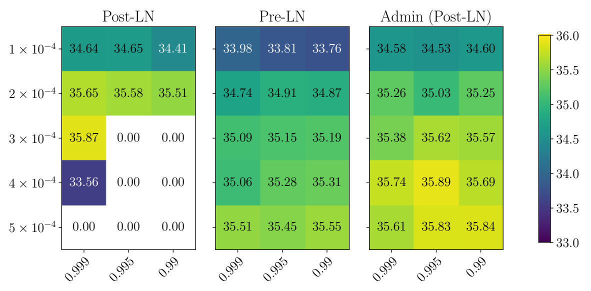

Figure 10: BLEU score of Post-LN, Pre-LN and Admin on the IWSLT'14 De-En dataset (x-axis is the β_2 for adaptive optimizers and y-axis is the learning rate). Pre-LN converges in all settings while Post-LN diverges in 7 out of 15 settings. When Post-LN converges, it outperforms Pre-LN in 7 out of 8 settings. Admin stabilizes Post-LN training and outperforms Pre-LN (its best performance is comparable with Post-LN).

6 Related Work

Transformer. Transformer (Vaswani et al., 2017) has led to a series of breakthroughs in various domains (Devlin et al., 2019; Velickovic et al., 2018; Huang et al., 2019; Parmar et al., 2018; Ramachandran et al., 2019). Liu et al. (2020) shows that comparing to other architectures, removing the warmup phrase is more damaging for Transformers, especially Post-LN. Similarly, it has been found that the original Transformer (referred as Post-LN) is less robust than its Pre-LN variant (Baevski and Auli, 2019; Nguyen and Salazar, 2019; Wang et al., 2019). Our studies go beyond existing literature about gradient vanishing (Xiong et al., 2019) and identify an important factor influencing Transformer training greatly. Our analysis guides us to propose a novel adaptive initialization method and allows us to better understand other empirical observations, *e.g.*, initializing parameters to smaller values helps to stabilize training (Nguyen and Salazar, 2019).

Deep Network Initialization. To handle the gradient vanishing in deep feedforward networks, specific initialization is derived and found to be useful (Glorot and Bengio, 2010). The derivation is further improved for ReLU networks (He et al., 2015). He et al. (2016) find the deep network training is still hard after addressing the gradient vanishing issue and propose ResNet. Balduzzi et al. (2017b) identifies the shattered gradient issue and proposes LookLinear initialization. Recently, the study of dynamical isometry(Xiao et al., 2018; Yang and Schoenholz, 2017; Pennington et al., 2017; Saxe et al., 2013b) provides a new perspective to analyze the network behavior at initialization and focus on simple networks like Deep Linear Network and gradient updates. On the other hand, it has been observed that scaling the residual outputs to smaller values help to stabilize training (Hanin and Rolnick, 2018; Mishkin and Matas, 2015; Zhang et al., 2019a; Bachlechner et al., 2020; Goyal et al., 2017). Here, we focus our study on the Transformer architecture, identify that unbalanced gradients is not the direct cause of the Post-LN instability, recognize the amplification effect of residual dependencies and propose a novel adaptive initialization method.

7 Conclusion

In this paper, we study the difficulties of training Transformer in theoretical and empirical manners. Our study in Section 3 suggests that the gradient vanishing problem is not the root cause of the unstable Transformer training. Also, the unbalanced gradient distribution is largely addressed by adaptive optimizers. In Section 4, we reveal the root cause of the instability to be the strong dependency on residual branches, which amplifies the fluctuation caused by parameter changes and destabilizes model training. In light of our analysis, we propose Admin, an adaptive initialization method to stabilize Transformers training. It controls the dependency in the beginning of training and maintains the flexibility to capture those dependencies once training stabilizes. Extensive experiments on real world datasets verify our intuitions and show that Admin achieves more stable training, faster convergence, and better performance.

Our work opens up new possibilities to not only further push the state-of-the-art but also better understand deep network training. It leads to many interesting future works, including generalizing Theorem 1 to other models, designing new algorithms to automatically adapt deep networks to different training configurations, upgrading the Transformer architecture, and applying our proposed Admin to conduct training in a larger scale.

Acknowledge

We thank Chengyu Dong, Haoming Jiang, Jingbo Shang, Xiaotao Gu, and Zihan Wang for valuable discussions and comments; Jingbo Shang for sharing GPU machines; and Microsoft for setting up GPU machines.

References

Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *ArXiv*, abs/1607.06450.

Thomas C. 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. *ArXiv*, abs/2003.04887.

Alexei Baevski and Michael Auli. 2019. Adaptive input representations for neural language modeling. In *ICLR*.

David Balduzzi, Marcus Frean, Lennox Leary, J. P. Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. 2017a. The shattered gradients problem: If resnets are the answer, then what is the question? In *ICML*.

David Balduzzi, Marcus Frean, Lennox Leary, J P Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. 2017b. The shattered gradients problem: If resnets are the answer, then what is the question? In *ICML*.

Yoshua Bengio, Patrice Y. Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Niki Parmar, Michael Schuster, Zhi-Feng Chen, Yonghui Wu, and Macduff Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. In *ACL*.

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

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*.

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *ArXiv*, abs/1706.02677.

Boris Hanin and David Rolnick. 2018. How to start training: The effect of initialization and architecture. In *NeurIPS*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. 2019. Music transformer: Generating music with long-term structure. In *ICLR*.

Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. 2018. Visualizing the loss landscape of neural nets. In *NeurIPS*.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. 2020. On the variance of the adaptive learning rate and beyond. In *ICLR*.

Yiping Lu, Zhuohan Li, Di He, Zhiqing Sun, Bin Dong, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2020. Understanding and improving transformer from a multi-particle dynamic system point of view. In *ICLR Workshop DeepDiffEq*.

Dmytro Mishkin and Juan E. Sala Matas. 2015. All you need is a good init. In *ICLR*.

Toan Q. Nguyen and Julian Salazar. 2019. Transformers without tears: Improving the normalization of self-attention. In *IWSLT*.

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 NAACL-HLT 2019: Demonstrations*.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. In *ICML*.

Jeffrey Pennington, Samuel S. Schoenholz, and Surya Ganguli. 2017. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In *NIPS*.

Martin Popel and Ondrej Bojar. 2018. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110:43 – 70.

Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. 2019. Stand-alone self-attention in vision models. In *NeurIPS*.

Andrew M Saxe, James L McClelland, and Surya Ganguli. 2013a. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. 2013b. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *CVPR*.

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 *NIPS*.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.

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*.

Felix Wu, Angela Fan, Alexei Baevski, Yann Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. In *ICLR*.

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, and Jeffrey Pennington. 2018. Dynamical isometry and a mean field theory of cnns: How to train 10, 000-layer vanilla convolutional neural networks. In *ICML*.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shu xin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Li-Wei Wang, and Tie-Yan Liu. 2019. On layer normalization in the transformer architecture. *ArXiv*, abs/2002.04745.

Greg Yang and Samuel S. Schoenholz. 2017. Mean field residual networks: On the edge of chaos. In *NIPS*.

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

Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank J. Reddi, Surinder Kumar, and Suvrit Sra. 2019b. Why adam beats sgd for attention models. *ArXiv*, abs/1912.03194.

A Gradients at Initialization

Here, we first reveal that Pre-LN does not suffer from the gradient vanishing. Then we establish that only the Post-LN decoder suffers from the gradient vanishing, but not the Post-LN encoder. For simplicity, we use $\Delta \mathbf{x}$ to denote gradients, *i.e.*, $\Delta \mathbf{x} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}}$ where $\mathcal{L}$ is the training objective. Following the previous study (Bengio et al., 1994; Glorot and Bengio, 2010; He et al., 2015; Saxe et al., 2013a), we analyze the gradient distribution at the very beginning of training, assume that the randomly initialized parameters and the partial derivative with regard to module inputs are independent.

A.1 Pre-LN Analysis

For Pre-LN encoders, we have $\mathbf{x}^{(pe)}_{2i} = \mathbf{x}^{(pe)}_{2i-1} + f_{\text{FFN}}(f_{\text{LN}}(\mathbf{x}^{(pe)}_{2i-1}))$ and $\Delta \mathbf{x}^{(pe)}_{2i-1} = \Delta \mathbf{x}^{(pe)}_{2i}(1 + \frac{\partial f_{\text{FFN}}(f_{\text{LN}}(\mathbf{x}^{(pe)}_{2i-1}))}{\partial \mathbf{x}^{(pe)}_{2i}})$. At initialization, the two terms on the right part are approximately independent and $E[\frac{\partial f_{\text{FFN}}(f_{\text{LN}}(\mathbf{z}^{(pe)}_{2i-1}))}{\partial \mathbf{x}^{(pe)}_{2i}}] = 0$. Therefore we have $\text{Var}[\Delta \mathbf{x}^{(pe)}_{2i-1}] \geq \text{Var}[\Delta \mathbf{x}^{(pe)}_{2i}]$. Similarly, we can get $\text{Var}[\Delta \mathbf{x}^{(pe)}_{2i-2}] \geq \text{Var}[\Delta \mathbf{x}^{(pe)}_{2i-1}]$ thus $\forall i \leq j, \text{Var}[\Delta \mathbf{x}^{(pe)}_{i}] \geq \text{Var}[\Delta \mathbf{x}^{(pe)}_{j}]$. Applying the same analysis to Pre-LN decoders, we can get $\forall i \leq j, \text{Var}[\Delta \mathbf{x}^{(pd)}_{i}] \geq \text{Var}[\Delta \mathbf{x}^{(pd)}_{j}]$. Thus, lower layers have larger gradients than higher layers and gradients do not vanish in the back-propagation.

REMARK 1. — For Pre-LN, if $\forall i, \Delta \mathbf{x}^{(p\cdot)}_{i}$ and the derivatives of modules in the i-th sub-layer are independent, then $\forall i \leq j, \text{Var}[\Delta \mathbf{x}^{(p\cdot)}_{i}] \geq \text{Var}[\Delta \mathbf{x}^{(p\cdot)}_{j}]$.

A.2 Post-LN Encoder Analysis

Different from Pre-LN, $\mathbf{x}^{(oe)}_{i}$ and $\mathbf{x}^{(oe)}_{i-1}$ are associated with not only the residual connection, but the layer normalization, which makes it harder to establish the connection on their gradients. After making assumptions on the model initialization, we find that lower layers in Post-LN encoder also have larger gradients than higher layers and gradients do not vanish in the back-propagation through the encoder.

THEOREM 2. — For Post-LN Encoders, if γ and β in the Layer Norm are initialized as 1 and 0 respectively; all other parameters are initialized by symmetric distributions with zero mean; $\mathbf{x}^{(oe)}_{i}$ and $\Delta \mathbf{x}^{(oe)}_{i}$ are subject to symmetric distributions with zero mean; the variance of $\mathbf{x}^{(oe)}_{i}$ is 1 (*i.e.*, normalized by Layer Norm); $\Delta \mathbf{x}^{(oe)}_{i}$ and the derivatives of modules in i-th sub-layer are independent, we have $\text{Var}[\Delta \mathbf{x}_{i-1}] \geq \text{Var}[\Delta \mathbf{x}_{i}]$.

Proof. We first prove $\text{Var}[\Delta \mathbf{x}^{(oe)}_{2i-1}] \geq \text{Var}[\Delta \mathbf{x}^{(oe)}_{2i}]$, *i.e.*, the backpropagation through FFN sublayers does not suffer from gradient vanishing. In Post-LN encoders, the output of FFN sublayers are calculated as $\mathbf{x}^{(oe)}_{2i} = f_{\text{LN}}(\mathbf{b}^{(oe)}_{2i})$ where $\mathbf{b}^{(oe)}_{2i} = \mathbf{x}^{(oe)}_{2i-1} + \max(0, \mathbf{x}^{(oe)}_{2i-1} W^{(1)}) W^{(2)}$. Since at initialization, $W^{(1)}$ and $W^{(2)}$ are independently randomized by symmetric distributions, we have $\mathbb{E}[\mathbf{b}^{(oe)}_{2i}] = 0$ and

$$\mathbf{x}^{(oe)}_{2i} = \frac{\mathbf{x}^{(oe)}_{2i-1} + \max(\mathbf{x}^{(oe)}_{2i-1} W^{(1)}, 0) W^{(2)}}{\sigma_{b,2i}}$$

where $\sigma^2_{b,2i} = \text{Var}[\mathbf{b}^{(oe)}_{2i}]$. Referring the dimension of $W^{(1)}$ as $D \times D_f$, He et al. (2015) establishes that

$$\text{Var}[\max(\mathbf{x}^{(oe)}_{2i-1} W^{(1)}, 0) W^{(2)}] = \frac{1}{2} D D_f \,\text{Var}[w^{(1)}]\,\text{Var}[w^{(2)}]\,\text{Var}[\mathbf{x}^{(oe)}_{2i-1}].$$

Since in Post-LN, $\mathbf{x}^{(oe)}_{2i-1}$ is the output of layer norm, we have $\text{Var}[\mathbf{x}^{(oe)}_{2i-1}] = 1$. Thus,

$$\begin{aligned}\sigma^2_{b,2i} &= \text{Var}[\mathbf{b}^{(oe)}_{2i}] = \text{Var}[\mathbf{x}^{(oe)}_{2i-1}] + \text{Var}[\max(\mathbf{x}^{(oe)}_{2i-1} W^{(1)}, 0) W^{(2)}] \\ &= 1 + \frac{1}{2} D D_f \,\text{Var}[w^{(1)}]\,\text{Var}[w^{(2)}]. \qquad (1)\end{aligned}$$

Assuming different terms are also independent in the backpropagation, we have

$$\mathrm{Var}[\Delta \mathbf{x}_{2i-1}^{(oe)}] \geq \mathrm{Var}[\frac{1}{\sigma_{b,2i}}(\Delta \mathbf{x}_{2i}^{(oe)} + \Delta \mathbf{x}_{2i}^{(oe)} \frac{\partial \max(\mathbf{x}_{2i-1}^{(oe)} W^{(1)}, 0) W^{(2)}}{\partial \mathbf{x}_{2i-1}^{(oe)}})].$$

At initialization, He et al. (2015) establishes that

$$\mathrm{Var}[\Delta \mathbf{x}_{2i}^{(oe)} \frac{\partial \max(\mathbf{x}_{2i-1}^{(oe)} W^{(1)}, 0) W^{(2)}}{\partial \mathbf{x}_{2i-1}^{(oe)}}] = \frac{1}{2} D D_f \,\mathrm{Var}[w^{(1)}] \,\mathrm{Var}[w^{(2)}] \,\mathrm{Var}[\Delta \mathbf{x}_{2i}^{(oe)}].$$

Therefore, we have

$$\mathrm{Var}[\Delta \mathbf{x}_{2i-1}^{(oe)}] \geq \frac{1}{\sigma_{b,2i}^2}(1 + \frac{1}{2} D D_f \,\mathrm{Var}[w^{(1)}] \,\mathrm{Var}[w^{(2)}]) \,\mathrm{Var}[\Delta \mathbf{x}_{2i}^{(oe)}]. \tag{2}$$

Combining Equation 1 with Equation 2, we have

$$\mathrm{Var}[\Delta \mathbf{x}_{2i-1}^{(oe)}] \geq \mathrm{Var}[\Delta \mathbf{x}_{2i}^{(oe)}] \tag{3}$$

which shows the backpropagation through FFN sublayers does not suffer from gradient vanishing.

Now we proceed to proof that, $\mathrm{Var}[\Delta \mathbf{x}_{2i-2}^{(oe)}] \geq \mathrm{Var}[\Delta \mathbf{x}_{2i-1}^{(oe)}]$, *i.e.*, the backpropagation through Self-Attention sublayers does not suffer from gradient vanishing. In Post-LN encoders, the output of Self-Attention sublayers are calculated as $\mathbf{x}_{2i-1}^{(oe)} = f_{\mathrm{LN}}(\mathbf{b}_{2i-1}^{(oe)})$ where $\mathbf{b}_{2i-1}^{(oe)} = \mathbf{x}_{2i-2}^{(oe)} + \mathbf{a}_{2i-1}^{(oe)}$ and $\mathbf{a}_{2i-1}^{(od)} = \sum_h f_s(\mathbf{x}_{2i-2}^{(oe)} W_h^{(Q)} W_h^{(K)} \mathbf{x}^{T}{}_{2i-2}^{(oe)}) \mathbf{x}_{2i-2}^{(oe)} W_h^{(V_1)} W_h^{(V_2)}$. At initialization, since $W^{(Q)}$, $W^{(K)}$, $W^{(V_1)}$ and $W^{(V_2)}$ are independently randomized by symmetric distributions, we have $\mathbb{E}[\mathbf{b}_{2i-1}^{(od)}] = 0$, thus $\mathbf{x}_{2i-1}^{(oe)} = \frac{\mathbf{b}_{2i-1}^{(oe)}}{\sigma_{b,2i-1}}$, where $\sigma_{b,2i-1}^2 = \mathrm{Var}[\mathbf{b}_{2i-1}^{(oe)}] = \mathrm{Var}[\mathbf{x}_{2i-2}^{(oe)}] + \mathrm{Var}[\mathbf{a}_{2i-1}^{(oe)}]$.

Referring $\mathbb{E}[{f_s}^2(\mathbf{x}_{2i-2}^{(oe)} W_h^{(Q)} W_h^{(K)} \mathbf{x}^{T}{}_{2i-2}^{(oe)})]$ as P_h, we have

$$\mathrm{Var}[\mathbf{a}_{2i-1}^{(od)}] = \mathrm{Var}[\mathbf{x}_{2i-2}^{(oe)} W_h^{(V_1)} W_h^{(V_2)}] H P_h.$$

Similar to He et al. (2015), we have

$$\mathrm{Var}[\mathbf{x}_{2i-2}^{(oe)} W_h^{(V_1)} W_h^{(V_2)}] = \frac{D^2}{H} \mathrm{Var}[\mathbf{x}_{2i-2}^{(oe)}] \,\mathrm{Var}[w^{(V_1)}] \,\mathrm{Var}[w^{(V_2)}].$$

Since $\mathbf{x}_{2i-2}^{(oe)}$ is the output of layer norm, we have $\mathrm{Var}[\mathbf{x}_{2i-2}^{(oe)}] = 1$. Thus,

$$\sigma_{b,2i-1}^2 = 1 + D^2 P_h \,\mathrm{Var}[\mathbf{x}_{2i-2}^{(oe)}] \,\mathrm{Var}[w^{(V_1)}] \,\mathrm{Var}[w^{(V_2)}]. \tag{4}$$

In the backpropagation, we have

$$\begin{aligned}
&\mathrm{Var}[\Delta \mathbf{x}_{2i-2}^{(oe)}] \geq \mathrm{Var}[\frac{1}{\sigma_{b,2i-1}}(\Delta \mathbf{x}_{2i-1}^{(oe)} + \Delta \mathbf{x}_{2i-1}^{(oe)} \sum_h \frac{\partial f_s(\mathbf{x}_{2i-2}^{(oe)} W_h^{(Q)} W_h^{(K)} \mathbf{x}^{T}{}_{2i-2}^{(oe)}) \mathbf{x}_{2i-2}^{(oe)} W_h^{(V_1)} W_h^{(V_2)}}{\partial \mathbf{x}_{2i-2}^{(oe)}})] \\
&\geq \frac{1}{\sigma_{b,2i-1}^2}(\mathrm{Var}[\Delta \mathbf{x}_{2i-1}^{(oe)}] + \mathrm{Var}[\Delta \mathbf{x}_{2i-1}^{(oe)} \sum_h f_s(\mathbf{x}_{2i-2}^{(oe)} W_h^{(Q)} W_h^{(K)} \mathbf{x}^{T}{}_{2i-2}^{(oe)}) \frac{\partial \mathbf{x}_{2i-2}^{(oe)} W_h^{(V_1)} W_h^{(V_2)}}{\partial \mathbf{x}_{2i-2}^{(oe)}}])
\end{aligned}$$

At initialization, we assume $\Delta \mathbf{x}_{2i-1}^{(oe)}$ and model parameters are independent (He et al., 2015), thus

$$\begin{aligned}
&\mathrm{Var}[\Delta \mathbf{x}_{2i-1}^{(oe)} \sum_h f_s(\mathbf{x}_{2i-2}^{(oe)} W_h^{(Q)} W_h^{(K)} \mathbf{x}^{T}{}_{2i-2}^{(oe)}) \frac{\partial \mathbf{x}_{2i-2}^{(oe)} W_h^{(V_1)} W_h^{(V_2)}}{\partial \mathbf{x}_{2i-2}^{(oe)}}] \\
=& D^2 P_h \,\mathrm{Var}[\Delta \mathbf{x}_{2i-1}^{(oe)}] \,\mathrm{Var}[w^{(V_1)}] \,\mathrm{Var}[w^{(V_2)}]
\end{aligned}$$

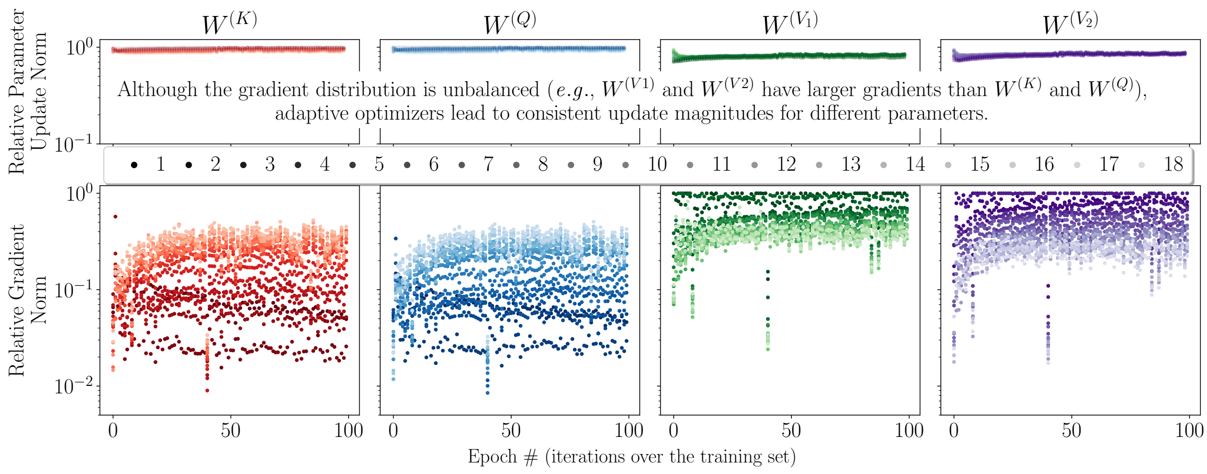

Figure 11: Relative Norm of Gradient (ΔW_i, where W_i is the checkpoint of i-th epoch) and Update ($|W_{i+1} - W_i|$) of Self-Attention Parameters in 12-Layer Pre-LN.

Therefore, we have

$$\mathrm{Var}[\Delta \mathbf{x}_{2i-2}^{(oe)}] \geq \frac{1}{\sigma_{b,2i-1}^2}(1 + D^2 P_h \,\mathrm{Var}[w^{(V_1)}]\,\mathrm{Var}[w^{(V_2)}])\,\mathrm{Var}[\Delta \mathbf{x}_{2i-1}^{(oe)}]. \quad (5)$$

Integrating Equation 4 with Equation 5, we have

$$\mathrm{Var}[\Delta \mathbf{x}_{2i-2}^{(oe)}] \geq \mathrm{Var}[\Delta \mathbf{x}_{2i-1}^{(oe)}]. \quad (6)$$

Combining Equation 3 and Equation 6, we have $\mathrm{Var}[\Delta \mathbf{x}_{i-1}] \geq \mathrm{Var}[\Delta \mathbf{x}_i]$. □

A.3 Post-LN Decoder Analysis

In Post-LN, the Encoder-Attention sub-layer suffers from gradient vanishing. The Encoder-Attention sub-layer calculates outputs as $\mathbf{x}_{3i-1}^{(od)} = f_{\mathrm{LN}}(\mathbf{b}_{3i-1}^{(od)})$ where $\mathbf{b}_{3i-1}^{(od)} = \mathbf{x}_{3i-2}^{(od)} + \mathbf{a}_{3i-1}^{(od)}$ and $\mathbf{a}_{3i-1}^{(od)} = \sum_h f_s(\mathbf{x}_{3i-2}^{(od)} W_h^{(Q)} W_h^{(K)} \mathbf{x}^{T(oe)}) \mathbf{x}^{(oe)} W_h^{(V_1)} W_h^{(V_2)}$. Here $\mathbf{x}^{(oe)}$ is encoder outputs and f_s is the row-wise softmax function. In the backpropagation, $\Delta \mathbf{x}_{3i-2}^{(od)} \approx \frac{\Delta \mathbf{x}_{3i-1}^{(od)}}{\sigma_{b,3i-1}}(1 + \frac{\partial \mathbf{a}_{3i-1}^{(od)}}{\mathbf{x}_{3i-2}^{(od)}})$. All of backpropagations from $\mathbf{a}_{3i-1}^{(od)}$ to $\mathbf{x}_{3i-2}^{(od)}$ went through the softmax function, we have $\mathrm{Var}[\frac{\partial \mathbf{a}_{3i-1}^{(od)}}{\mathbf{x}_{3i-2}^{(od)}}] + 1 \leq \sigma_{b,3i-1}^2$. Thus, those backpropagations suffer from gradient vanishing.

A.4 Gradients of Unbalanced Gradients

As in Figure 3 and Figure 11, even for Pre-LN, the gradient distributions of Attention modules are unbalanced. Specifically, parameters within the softmax function (*i.e.*, $W^{(K)}$ and $W^{(V_1)}$) suffer from gradient vanishing (*i.e.*, $\frac{\partial f_s(x_0, \cdots, x_i, \cdots)}{\partial x_i} \leq 1$) and have smaller gradients than other parameters.

With further analysis, we find it is hard to neutralize the gradient vanishing of softmax. Different from conventional non-linear functions like ReLU or sigmoid, softmax has a dynamic input length (*i.e.*, for sentence with different lengths, inputs of softmax have different dimensions). Although this setting allows Attention modules to handle sequential inputs, it restricts them from having stable and consistent backpropagation. Specifically, let us consider the comparison between softmax and sigmoid. For the sigmoid function, although its derivation is smaller than 1, this damping effect is consistent for all inputs. Thus, sigmoid can be neutralized by a larger initialization (Glorot and Bengio, 2010). For softmax, its damping effect is different for different inputs, thus cannot be neutralized by a static initialization.

Also, we observe that this issue is largely addressed by adaptive optimizers. Specifically, we calculate the norm of parameter change in consequent epochs (*e.g.*, $|W_{t+1}^{(K)} - W_t^{(K)}|$ where $W_t^{(K)}$ is the checkpoint saved after t epochs) and also visualize the relative norm (scaled by the largest value in the same network) in Figure 11. Comparing the relative norm of parameter gradients and parameter updates, we notice that: although the gradient distribution is unbalanced, adaptive optimizers successfully assign different learning rates to different parameters and lead to consistent update magnitudes. This results explains why the vanilla SGD fails for training Transformer (*i.e.*, lacking the ability to handle unbalanced gradient distributions). Also, it implies that the unbalanced gradient distribution (*e.g.*, gradient vanishing) has been largely addressed by adaptive optimizers and may not have a big impact on the training instability.

B Proof of Theorem 1

Here, we elaborate the derivation for Theorem 1, which establishes the relationship between layer number and output fluctuation brought by parameter change.

THEOREM 1. — Consider a N-layer Transformer $\widehat{\mathbf{x}} = \mathcal{F}(\widehat{\mathbf{x}}_0, W)$, where $\widehat{\mathbf{x}}_0$ is the input and W is the parameter. If the layer dependency stays the same after a parameter change (*i.e.*, $\beta_{i,j}$ has the same value after changing W to W^*, where W is randomly initialized and $\delta = W^* - W$ is independent to W), the output change (*i.e.*, $\mathrm{Var}[\mathcal{F}(\mathbf{x}_0, W) - \mathcal{F}(\mathbf{x}_0, W^*)]$) can be estimated as $\sum_{i=1}^{N} \beta_{i,i}^2 C$ where C is a constant.

Proof. We refer the module in i sub-layer as $\mathbf{a}_i = \mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}, W_i)$, where $\widehat{\mathbf{x}}_i = \sum_{j\leq i} \beta_{i,j}\widehat{\mathbf{a}}_j$ is the normalized residual output and $\widehat{\mathbf{a}}_i = \frac{\mathbf{a}_i}{\sqrt{\mathrm{Var}\,\mathbf{a}_i}}$ is the normalized module output. The final output is marked as $\widehat{\mathbf{x}} = \mathcal{F}(\mathbf{x}_0, W) = \sum_{j\leq N} \beta_{N,j}\widehat{\mathbf{a}}_j$. To simplify the notation, we use the superscript $*$ to indicate variables related to W^*, *e.g.*, $\widehat{\mathbf{x}}^* = \mathcal{F}(\mathbf{x}_0, W^*)$ and $\mathbf{a}_i^* = \mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i^*)$.

At initialization, all parameters are initialized independently. Thus $\forall i \neq j$, $\widehat{\mathbf{a}}_i$ and $\widehat{\mathbf{a}}_j$ are independent and $1 = \mathrm{Var}[\sum_{j\leq i} \beta_{i,j}\widehat{\mathbf{a}}_j] = \sum_{j\leq i} \beta_{i,j}^2$. Also, since k-layer and $(k+1)$-layer share the residual connection to previous layers, $\forall i, j \leq k$ we have $\frac{\beta_{i,k}}{\beta_{j,k}} = \frac{\beta_{i,k+1}}{\beta_{j,k+1}}$. Thus $\forall i \leq k$, $\beta_{i,k+1}^2 = (1-\beta_{k,k}^2)\beta_{i,k}^2$ and

$$\begin{aligned}\mathrm{Var}[\widehat{\mathbf{x}}_i - \widehat{\mathbf{x}}_i^*] &= \mathrm{Var}[\sum_{j\leq i} \beta_{i,j}(\widehat{\mathbf{a}}_j - \widehat{\mathbf{a}}_j^*)] = \sum_{j\leq i} \beta_{i,j}^2 \mathrm{Var}[\widehat{\mathbf{a}}_j - \widehat{\mathbf{a}}_j^*] \\ &= \beta_{i,i}^2 \mathrm{Var}[\widehat{\mathbf{a}}_i - \widehat{\mathbf{a}}_i^*] + (1-\beta_{i,i}^2)\mathrm{Var}[\widehat{\mathbf{x}}_i - \widehat{\mathbf{x}}_i^*]. \end{aligned} \tag{7}$$

Now, we proceed to analyze $\mathrm{Var}[\widehat{\mathbf{a}}_i - \widehat{\mathbf{a}}_i^*]$. Specifically, we have

$$\begin{aligned}\mathrm{Var}[\widehat{\mathbf{a}}_i - \widehat{\mathbf{a}}_i^*] &= \mathrm{Var}[\mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}, W_i) - \mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i^*)] \\ &= \mathrm{Var}[\mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}, W_i) - \mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i) + \mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i^*) - \mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i^*)] \\ &= \mathrm{Var}[\mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}, W_i) - \mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i)] + \mathrm{Var}[\mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i) - \mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i^*)]. \end{aligned} \tag{8}$$

Since W is randomly initialized, $\mathrm{Var}[\mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i) - \mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i^*)]$ should have the same value for all layers, thus we use a constant C to refer its value ($C = \mathrm{Var}[\mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i) - \mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i^*)]$ and $C \approx |\delta| \cdot |\nabla \mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i)|$). As to $\mathrm{Var}[\mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}, W_i) - \mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i)]$, since the sub-layer of Transformer are mostly using linear weights with ReLU nonlinearity and $1 = \mathrm{Var}[\mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}, W_i)] = \mathrm{Var}[\widehat{\mathbf{x}}_{i-1}]$, we have $\mathrm{Var}[\mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}, W_i) - \mathcal{G}_i(\widehat{\mathbf{x}}_{i-1}^*, W_i)] \approx \mathrm{Var}[\widehat{\mathbf{x}}_{i-1} - \widehat{\mathbf{x}}_{i-1}^*]$. Thus, we can rewrite Equation 8 and get

$$\mathrm{Var}[\widehat{\mathbf{a}}_i - \widehat{\mathbf{a}}_i^*] \approx \mathrm{Var}[\widehat{\mathbf{x}}_{i-1} - \widehat{\mathbf{x}}_{i-1}^*] + C$$

With Equation 7, we have

$$\begin{aligned}\mathrm{Var}[\widehat{\mathbf{x}}_i - \widehat{\mathbf{x}}_i^*] &= \beta_{i,i}^2 \mathrm{Var}[\widehat{\mathbf{a}}_i - \widehat{\mathbf{a}}_i^*] + (1-\beta_{i,i}^2)\mathrm{Var}[\widehat{\mathbf{x}}_i - \widehat{\mathbf{x}}_i^*] \\ &\approx \beta_{i,i}^2(\mathrm{Var}[\widehat{\mathbf{x}}_{i-1} - \widehat{\mathbf{x}}_{i-1}^*] + C) + (1-\beta_{i,i}^2)\mathrm{Var}[\widehat{\mathbf{x}}_i - \widehat{\mathbf{x}}_i^*] \\ &= \mathrm{Var}[\widehat{\mathbf{x}}_i - \widehat{\mathbf{x}}_i^*] + \beta_{i,i}^2 C \end{aligned}$$

Therefore, we have $\mathrm{Var}[\mathcal{F}(\mathbf{x}_0, W) - \mathcal{F}(\mathbf{x}_0, W^*)] \approx \sum_{i=1}^{N} \beta_{i,i}^2 C$. □

C Admin Implementation Details

As introduced in Section 4.3, we introduces a new set of parameters to re-scale the module outputs. Specifically, we refer these new parameters as ω and construct the Post-LN sub-layer as:

$$\mathbf{x}_i = f_{\text{LN}}(\mathbf{b}_i), \text{where } \mathbf{b}_i = \mathbf{x}_{i-1} \cdot \boldsymbol{\omega}_i + f_i(\mathbf{x}_{i-1})$$

where $\cdot$ is element-wise product.

After training, Admin can be reparameterized as the conventional Post-LN structure (*i.e.*, removing $\boldsymbol{\omega}_i$). Specifically, we refer $\mathbf{x}_i = \frac{\mathbf{b}_i}{\sigma_b}\boldsymbol{\gamma} + \boldsymbol{\nu}$. Then, for feed forward sub-layers, we have

$$\mathbf{b}_i = \mathbf{x}_{i-1} \cdot \omega + \max(0, \mathbf{x}_{i-1} W^{(1)}) W^{(2)}, \text{where } \mathbf{x}_i = \frac{\mathbf{b}_{i-1}}{\sigma_b}\boldsymbol{\gamma} + \boldsymbol{\nu}.$$

It can be reparameterized by changing $\boldsymbol{\gamma}$, $\boldsymbol{\nu}$, $W^{(1)}$ to $\boldsymbol{\gamma}\boldsymbol{\omega}_i$, $\boldsymbol{\nu}\boldsymbol{\omega}_i$, $\frac{1}{\boldsymbol{\omega}_i} W^{(1)}$ respectively, *i.e.*,

$$\mathbf{b}'_i = \mathbf{x}'_{i-1} + \max(0, \mathbf{x}'_{i-1} \frac{1}{\boldsymbol{\omega}_i} W^{(1)}) W^{(2)}, \text{where } \mathbf{x}'_{i-1} = \frac{\mathbf{b}'_{i-1}}{\sigma_b}\boldsymbol{\gamma}\boldsymbol{\omega}_i + \boldsymbol{\nu}\boldsymbol{\omega}_i.$$

For Self-Attention sub-layers, we have

$$\mathbf{b}_i = \mathbf{x}_{i-1} + \sum_h f_s(\mathbf{x}_{i-1} W_h^{(Q)} W_h^{(K)} \mathbf{x}_{i-1}) \mathbf{x}_{i-1} W_h^{(V_1)} W_h^{(V_2)}, \text{where } \mathbf{x}_i = \frac{\mathbf{b}_{i-1}}{\sigma_b}\boldsymbol{\gamma} + \boldsymbol{\nu}.$$

It can be reparameterized by changing $\boldsymbol{\gamma}$, $\boldsymbol{\nu}$, $W_h^{(Q)}$, $W_h^{(K)}$, $W_h^{(V_1)}$ to $\boldsymbol{\gamma}\boldsymbol{\omega}_i$, $\boldsymbol{\nu}\boldsymbol{\omega}_i$, $\frac{1}{\boldsymbol{\omega}_i} W_h^{(Q)}$, $\frac{1}{\boldsymbol{\omega}_i} W_h^{(K)}$ $\frac{1}{\boldsymbol{\omega}_i} W_h^{(V_1)}$ respectively, *i.e.*,

$$\mathbf{b}'_i = \mathbf{x}'_{i-1} + \sum_h f_s(\mathbf{x}'_{i-1} \frac{1}{\boldsymbol{\omega}_i} W_h^{(Q)} W_h^{(K)} \frac{1}{\boldsymbol{\omega}_i} \mathbf{x}'_{i-1}) \mathbf{x}'_{i-1} \frac{1}{\boldsymbol{\omega}_i} W_h^{(V_1)} W_h^{(V_2)}, \text{where } \mathbf{x}'_{i-1} = \frac{\mathbf{b}'_{i-1}}{\sigma_b}\boldsymbol{\gamma}\boldsymbol{\omega}_i + \boldsymbol{\nu}\boldsymbol{\omega}_i.$$

For Encoder-Attention sub-layers, we have

$$\mathbf{b}_i = \mathbf{x}_{i-1} + \sum_h f_s(\mathbf{x}_{i-1} W_h^{(Q)} W_h^{(K)} \mathbf{x}^{\cdot e}) \mathbf{x}^{\cdot e} W_h^{(V_1)} W_h^{(V_2)}, \text{where } \mathbf{x}_i = \frac{\mathbf{b}_{i-1}}{\sigma_b}\boldsymbol{\gamma} + \boldsymbol{\nu}.$$

It can be reparameterized by changing $\boldsymbol{\gamma}$, $\boldsymbol{\nu}$, $W_h^{(Q)}$ to $\boldsymbol{\gamma}\boldsymbol{\omega}_i$, $\boldsymbol{\nu}\boldsymbol{\omega}_i$, $\frac{1}{\boldsymbol{\omega}_i} W_h^{(Q)}$ respectively, *i.e.*,

$$\mathbf{b}'_i = \mathbf{x}'_{i-1} + \sum_h f_s(\mathbf{x}'_{i-1} \frac{1}{\boldsymbol{\omega}_i} W_h^{(Q)} W_h^{(K)} \mathbf{x}^{\cdot e}) \mathbf{x}^{\cdot e} \frac{1}{\boldsymbol{\omega}_i} W_h^{(V_1)} W_h^{(V_2)}, \text{where } \mathbf{x}'_{i-1} = \frac{\mathbf{b}'_{i-1}}{\sigma_b}\boldsymbol{\gamma}\boldsymbol{\omega}_i + \boldsymbol{\nu}\boldsymbol{\omega}_i.$$

It is easy to find $\mathbf{b}'_i = \mathbf{b}_i$ in all three situations.

From the previous analysis, it is easy to find that introducing the additional parameter $\boldsymbol{\omega}_i$ is equivalent to rescale some model parameters. In our experiments on IWSLT14 De-En, we find that directly rescaling parameters at initialization can get roughly the same performance with introducing $\boldsymbol{\omega}_i$, however, it is not very stable when conducting training in the half-precision manner. Accordingly, we choose to use add new parameters $\boldsymbol{\omega}_i$ instead of rescaling parameters.

D Experimental Setup

Our experiments are based on the implementation from the fairseq package (Ott et al., 2019) and follows the pre-processing setting in Lu et al. (2020). Specifically, on the IWSLT'14 De-En dataset, we use word embedding with 512 dimensions and 6-layer encoder/decoder with 4 head and 1024 feedforward dimensions; on the WMT'14 En-De dataset, we use word embedding with 512 dimension and 8-head encoder/decoder with 2048 hidden dimensions. Label smoothed cross entropy is used as the objective function with an uncertainty $= 0.1$ (Szegedy et al., 2016). On the WMT'14 En-De dataset, all dropout ratios (including (activation dropout and attention dropout) are set to 0.1, except two experiments for the FixUp initialization method (Chen et al., 2018). On the IWSLT'14 De-En dataset, we use a larger after-layer dropout (0.3) and use a weight decay of 0.0001.

Most hyper-parameters are adopted from Lu et al. (2020). Specifically, we use the RAdam optimizer (Liu et al., 2020) with $(\beta_1, \beta_2) = (0.9, 0.98)$, inverse sqrt learning rate scheduler with a warmup phrase (8000 steps on the WMT'14 En-De dataset and 6000 steps on the IWSLT'14 De-En dataset). The maximum learning rate is set to $1e^{-3}$ on the WMT14 En-De dataset and $7e^{-4}$ on the IWSLT14 De-En dataset. We conduct training for 100 epochs on the WMT14 En-De dataset and 90 epochs on the IWSLT14 De-En dataset, the last 10 checkpoints are averaged before inference. On the IWSLT'14 dataset, we conduct training on one NVIDIA GeForce GTX 1080 Ti GPU, set maximum batch size as 4000. On the WMT'16 dataset, we conduct training on four NVIDIA Quadro R8000 GPUs and set maximum batch size as 8196. All implementations are available at: `https://github.com/LiyuanLucasLiu/Transforemr-Clinic`