Understanding the Difficulty of Training Transformers

kimcando94@gmail.com

Overall

- Not about unbalanced gradients(e.g, gradient vanishing),
- But amplification of small parameter updates(perturbations) due to heavy dependency residual branch
  - Leading to disturbance in model output
  - If with light dependency, lower model potential
- Propose adaptive model initialization

In the previous papers,

BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension

i. encoder와 decoder의 hidden size => 12
ii. batch size => 8,000
iii. train steps => 500,000
iv. tokenizing method => BPE
v. Text Infilling + Sentence Shuffling => masking 30% of token, permute all sentences
vi. train step의 마지막 10% => dropout off
vii. pre-training data => 160Gb (news + books + stories + web text)

In the previous papers,

Content Planning for Neural Story Generation with Aristotelian Rescoring

A.4 Hyper-Parameters for Mixture Weight Tuning

Mixture weights are tuned with a held out validation set of 10,000 samples. The models train for 3 epochs, but all converge in 1 epoch, which takes 24 hours on 1 RTX 2080 GPU (11GB). We use SGD at each step, with learning rates set to 0.001.

In the previous papers,

Spelling Error Correction with Soft-Masked BERT

The pre-trained BERT model utilized in the experiments is the one provided at https://github.com/huggingface/transformers. In fine-tuning of BERT, we kept the default hyper-parameters and only fine-tuned the parameters using Adam. In order to reduce the impact of training tricks, we did not use the dynamic learning rate strategy and maintained a learning rate $2e^{-5}$ in fine-tuning. The size of hidden unit in Bi-GRU is 256 and all models use a batch size of 320.

In the previous papers,

RoBERTa: A Robustly Optimized BERT Pretraining Approach

2.4 Optimization

BERT is optimized with Adam (Kingma and Ba, 2015) using the following parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = $ 1e-6 and L_2 weight decay of 0.01. The learning rate is warmed up over the first 10,000 steps to a peak value of 1e-4, and then linearly decayed. BERT trains with a dropout of 0.1 on all layers and attention weights, and a GELU activation function (Hendrycks and Gimpel, 2016). Models are pretrained for $S =$ 1,000,000 updates, with minibatches containing $B = 256$ sequences of maximum length $T = 512$ tokens.

In the previous papers,

Text Summarization with Pretrained Encoders

We use two Adam optimizers with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for the encoder and the decoder, respectively, each with different warmup-steps and learning rates:

$$lr_{\mathcal{E}} = \tilde{lr}_{\mathcal{E}} \cdot \min(step^{-0.5}, \text{step} \cdot \text{warmup}_{\mathcal{E}}^{-1.5}) \quad (6)$$

$$lr_{\mathcal{D}} = \tilde{lr}_{\mathcal{D}} \cdot \min(step^{-0.5}, \text{step} \cdot \text{warmup}_{\mathcal{D}}^{-1.5}) \quad (7)$$

where $\tilde{lr}_{\mathcal{E}} = 2e^{-3}$, and $\text{warmup}_{\mathcal{E}} = 20,000$ for the encoder and $\tilde{lr}_{\mathcal{D}} = 0.1$, and $\text{warmup}_{\mathcal{D}} = 10,000$ for the decoder. This is based on the assumption that the pretrained encoder should be fine-tuned with a smaller learning rate and smoother decay (so that the encoder can be trained with more accurate gradients when the decoder is becoming stable).

The loss of the model is the binary classification entropy of prediction $\hat{y}_i$ against gold label y_i. Inter-sentence Transformer layers are jointly fine-tuned with BERTSUM. We use the Adam optimizer with $\beta_1 = 0.9$, and $\beta_2 = 0.999$). Our learning rate schedule follows (Vaswani et al., 2017) with warming-up ($\text{warmup} = 10,000$):

$$lr = 2e^{-3} \cdot \min\left(\text{step}^{-0.5}, \text{step} \cdot \text{warmup}^{-1.5}\right)$$

Pre LN , Post LN

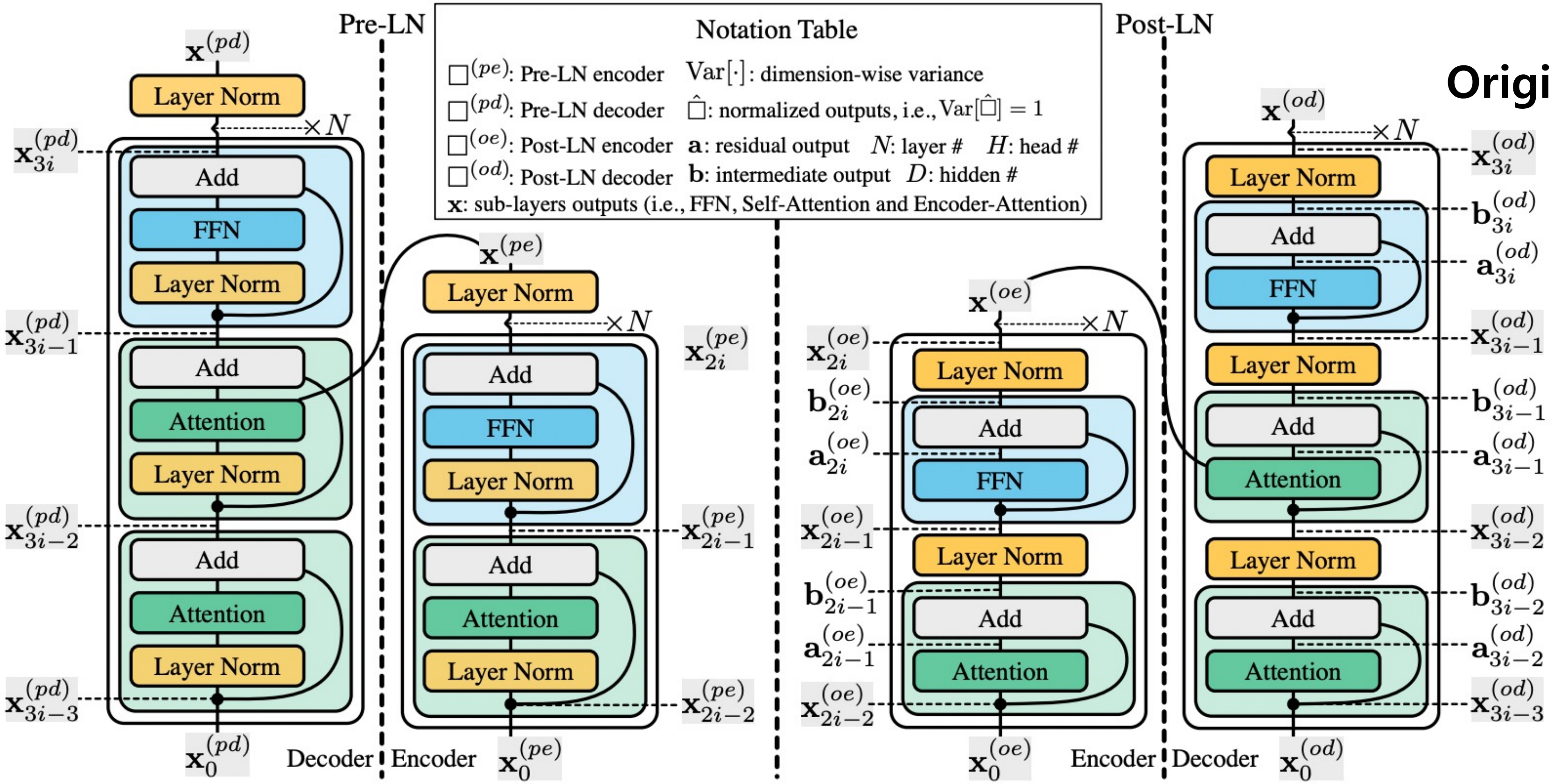
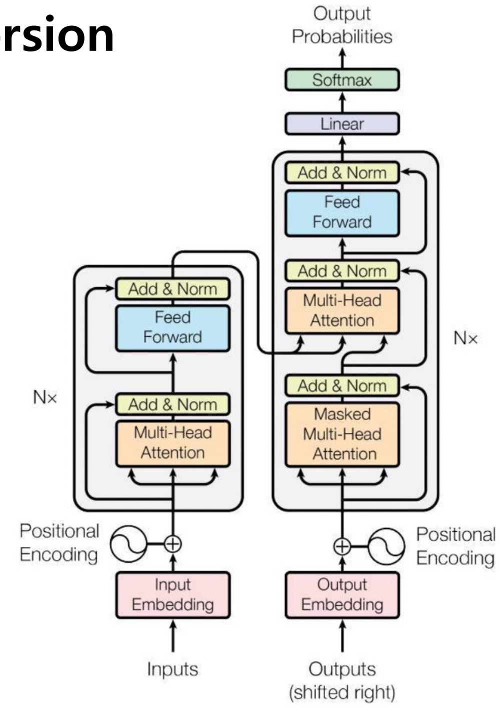
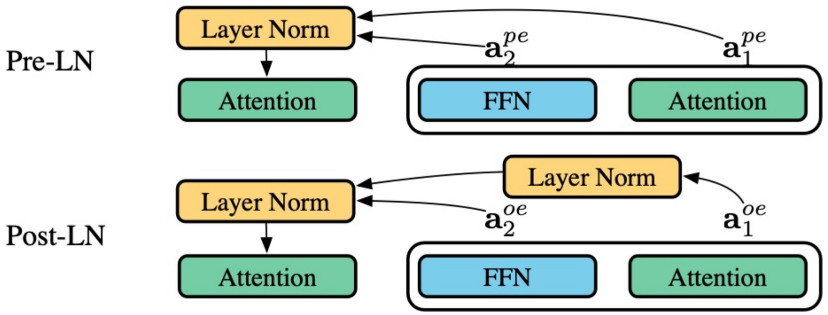


Figure 2: The Architecture and notations of Pre-LN Transformers (Left) and Post-LN Transformers (Right).

Original version



Pre LN, Post LN



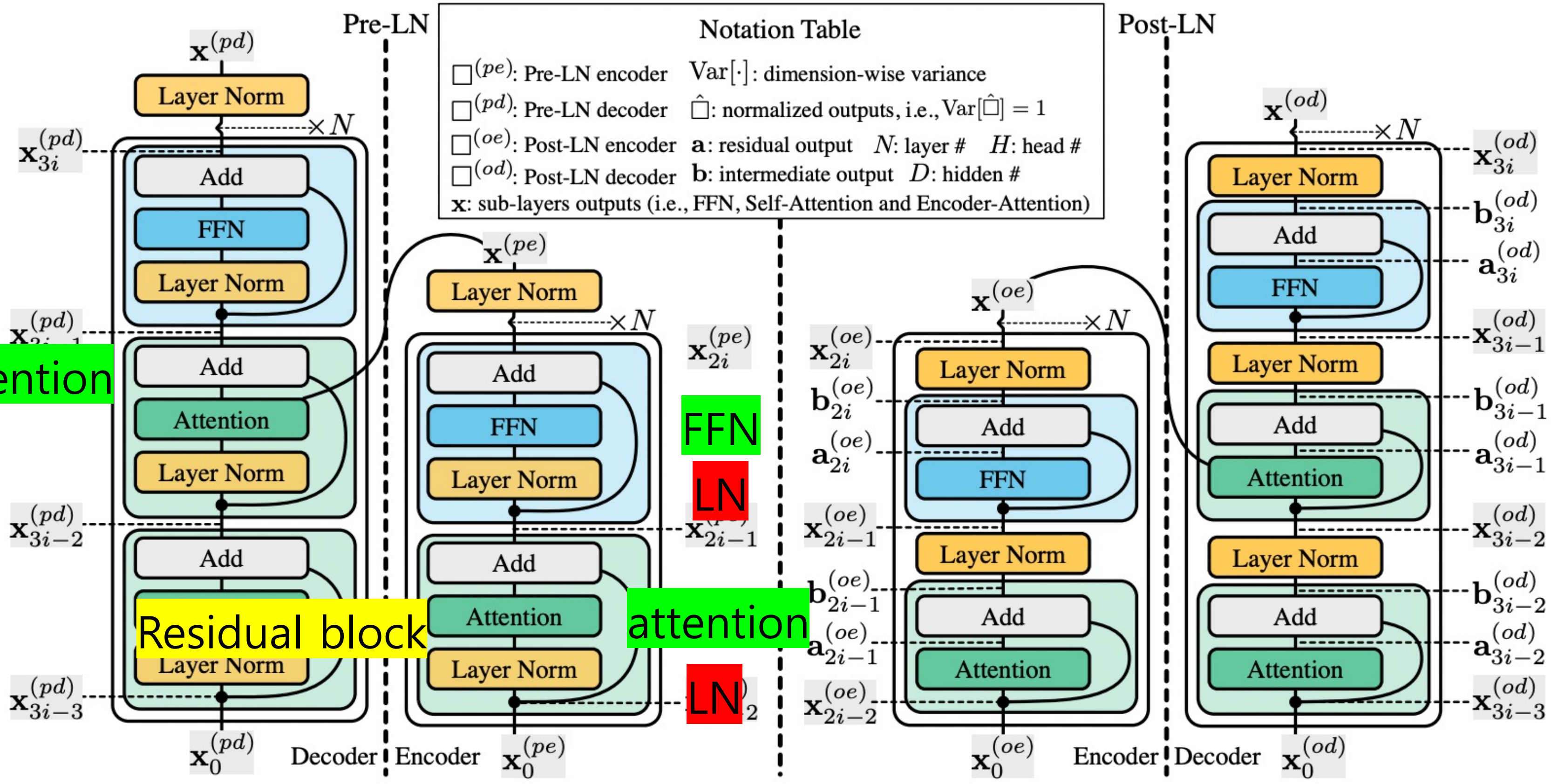
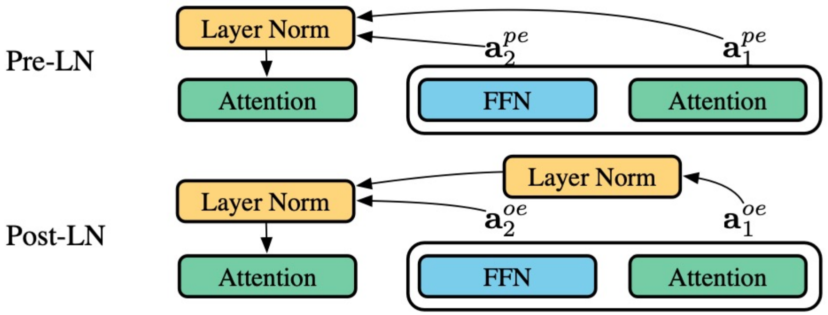


Figure 2: The Architecture and notations of Pre-LN Transformers (Left) and Post-LN Transformers (Right).

Pre LN , Post LN



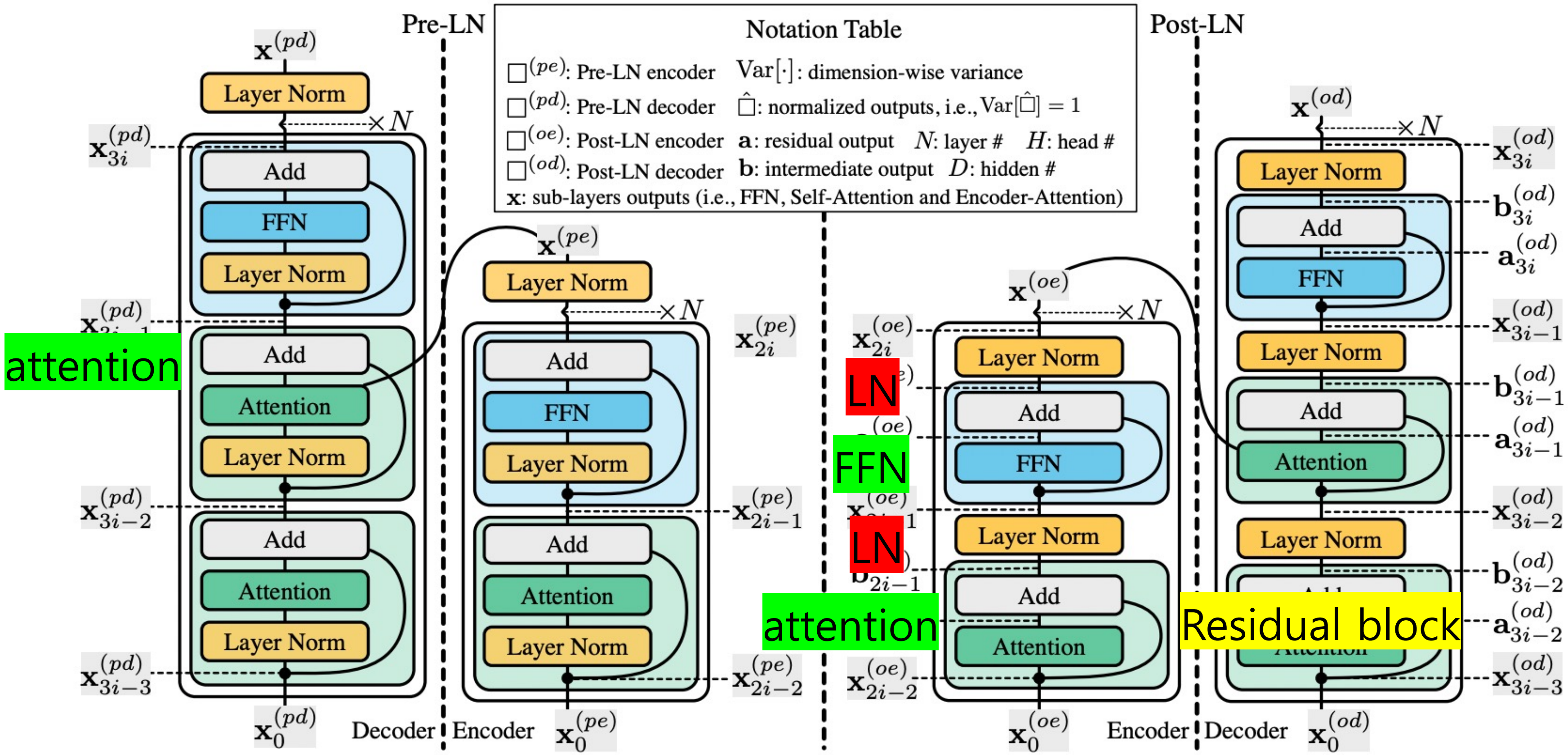
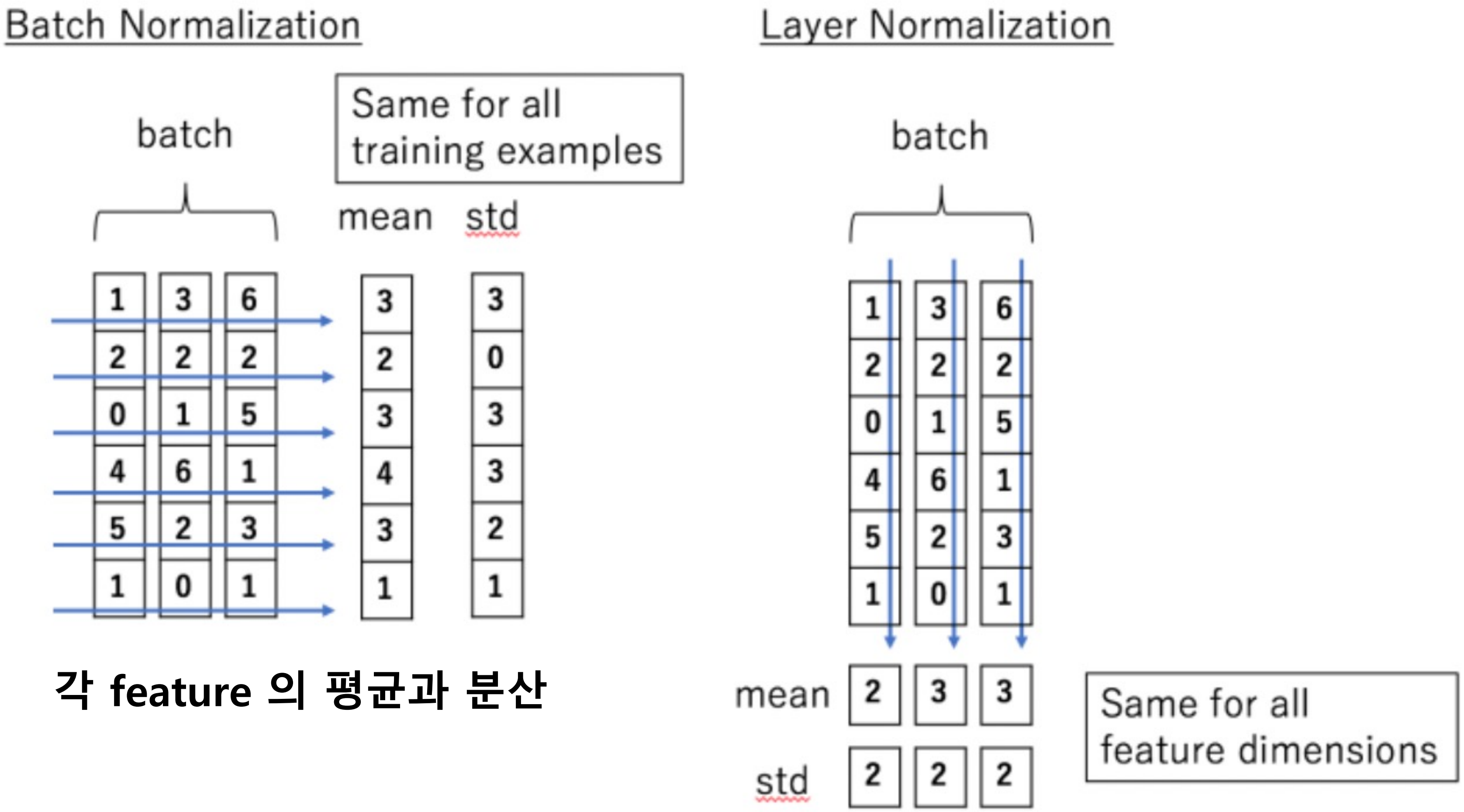


Figure 2: The Architecture and notations of Pre-LN Transformers (Left) and Post-LN Transformers (Right).

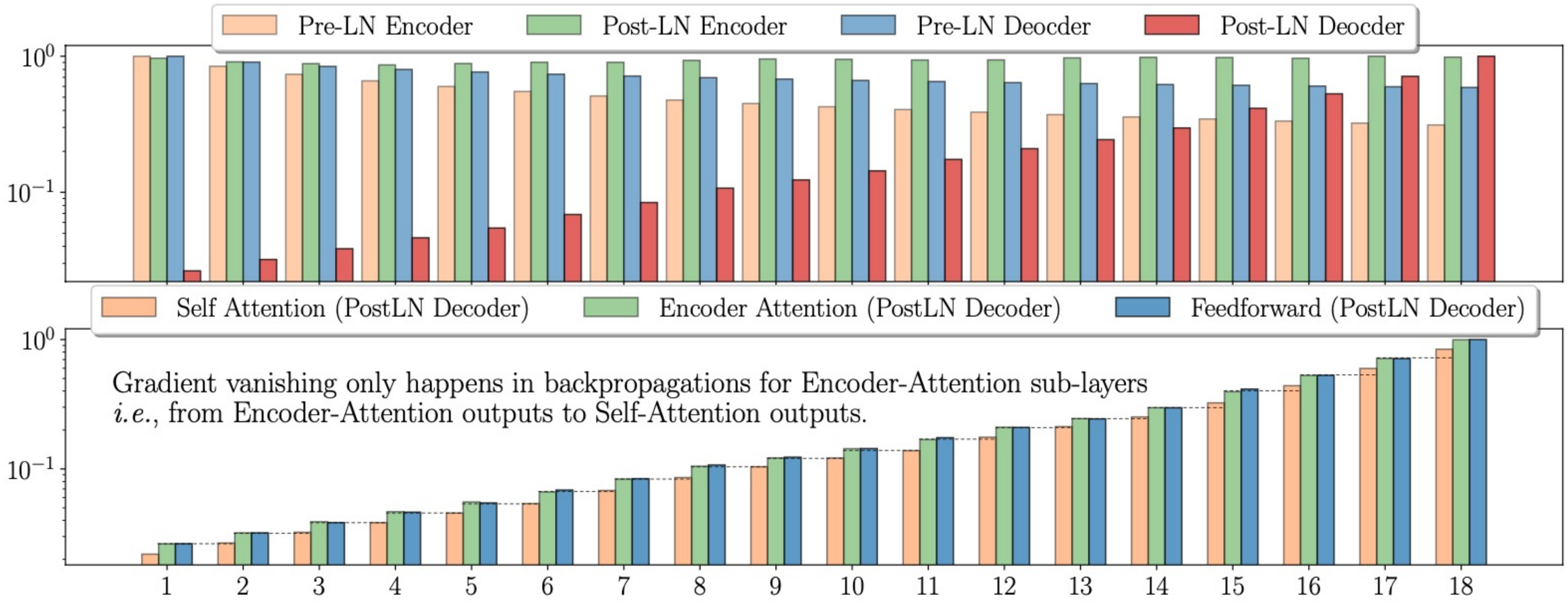
Layer Norm



각 feature 의 평균과 분산

각 input feature 의 평균과 분산

18th layers Transformer



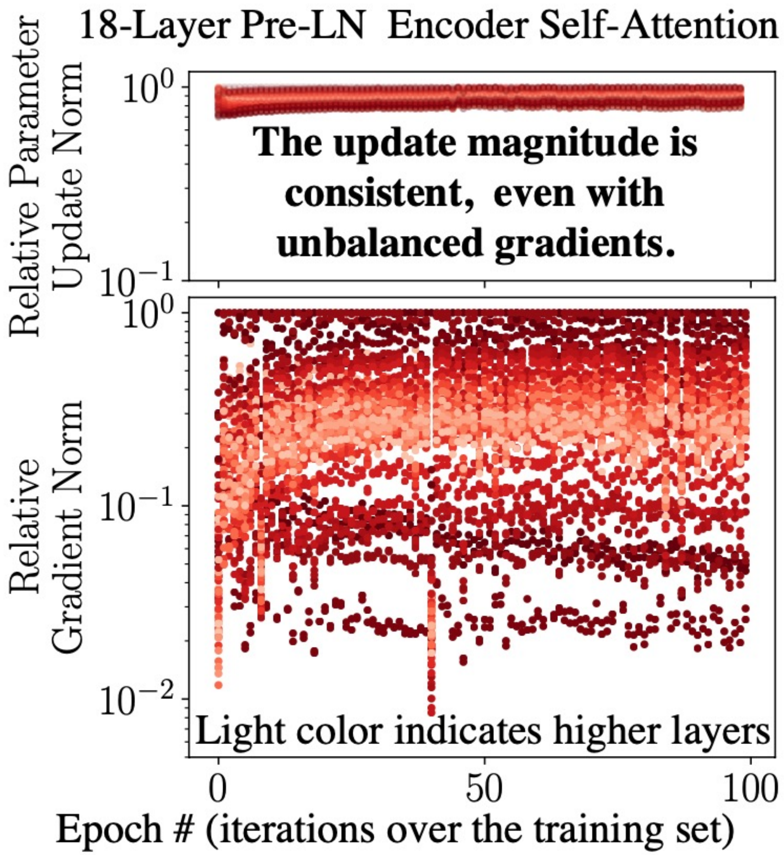
$$\frac{||\Delta \mathbf{x}_i^{(\cdot)}||_2}{\max_j ||\Delta \mathbf{x}_j^{(\cdot)}||_2}$$

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.

Pre LN decoder

Post LN encoder



Relative Gradient Norm $\frac{|\nabla w_i^t|}{\max_j |\nabla w_j^t|}$

Relative Parameter Update Norm $\frac{|w_i^{t+1} - w_i^t|}{\max_j |w_j^{t+1} - w_j^t|}$

업데이트되는 양의 range는 적음

Gradient 의 range 범위는 크나

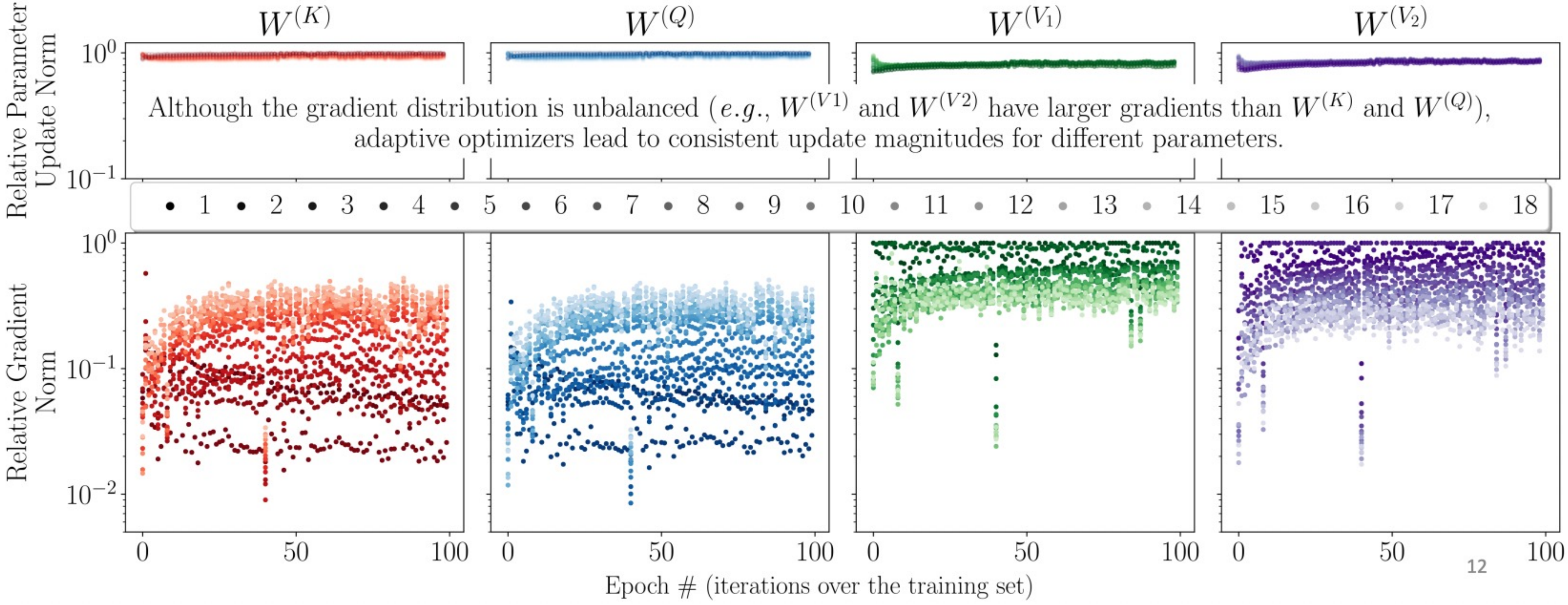
Figure 5: Histogram of relative norm of gradient and $|W_{i+1} - W_i|$ where W_i is the checkpoint saved after training for i epochs.

Gradient가 parameter 마다 unbalance 하나, adaptive optimizer가 제 역할을 하고 있다!
→ SGD가 작동 어려운 이유!
→ 불안정한 학습이 unbalanced gradient만의 문제가 아니지 않을까

Unbalanced gradients are largely handled by adaptive optimizers.

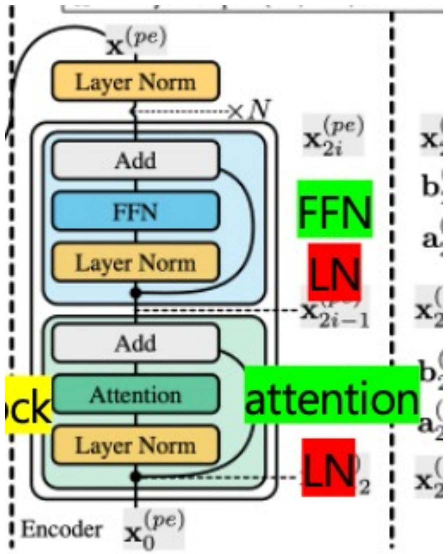
Relative Gradient Norm $\frac{|\nabla \boldsymbol{w}_i^t|}{\max_j |\nabla \boldsymbol{w}_j^t|}$ Relative Parameter Update Norm $\frac{|\boldsymbol{w}_i^{t+1} - \boldsymbol{w}_i^t|}{\max_j |\boldsymbol{w}_j^{t+1} - \boldsymbol{w}_j^t|}$

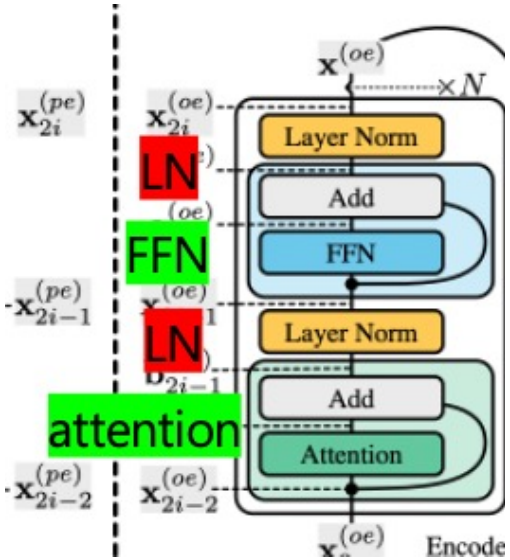
As unbalanced gradients are largely handled by adaptive optimizers, it necessitates the use of adaptive optimizers.



Instability from amplification effect

- Layer dependence

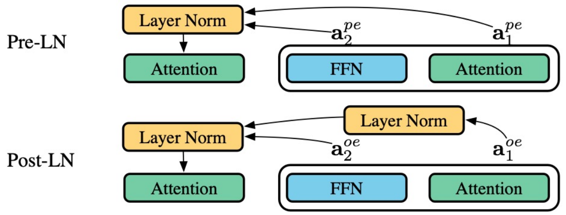




1. Layer norm --> Pre LN은 input을, post LN은 output을 정규화

Instability from amplification effect

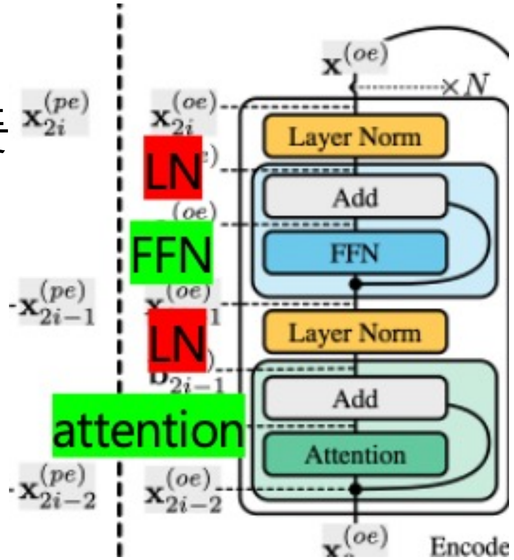
- Layer dependence



residual block의 아웃풋== 새로운 residual의 input

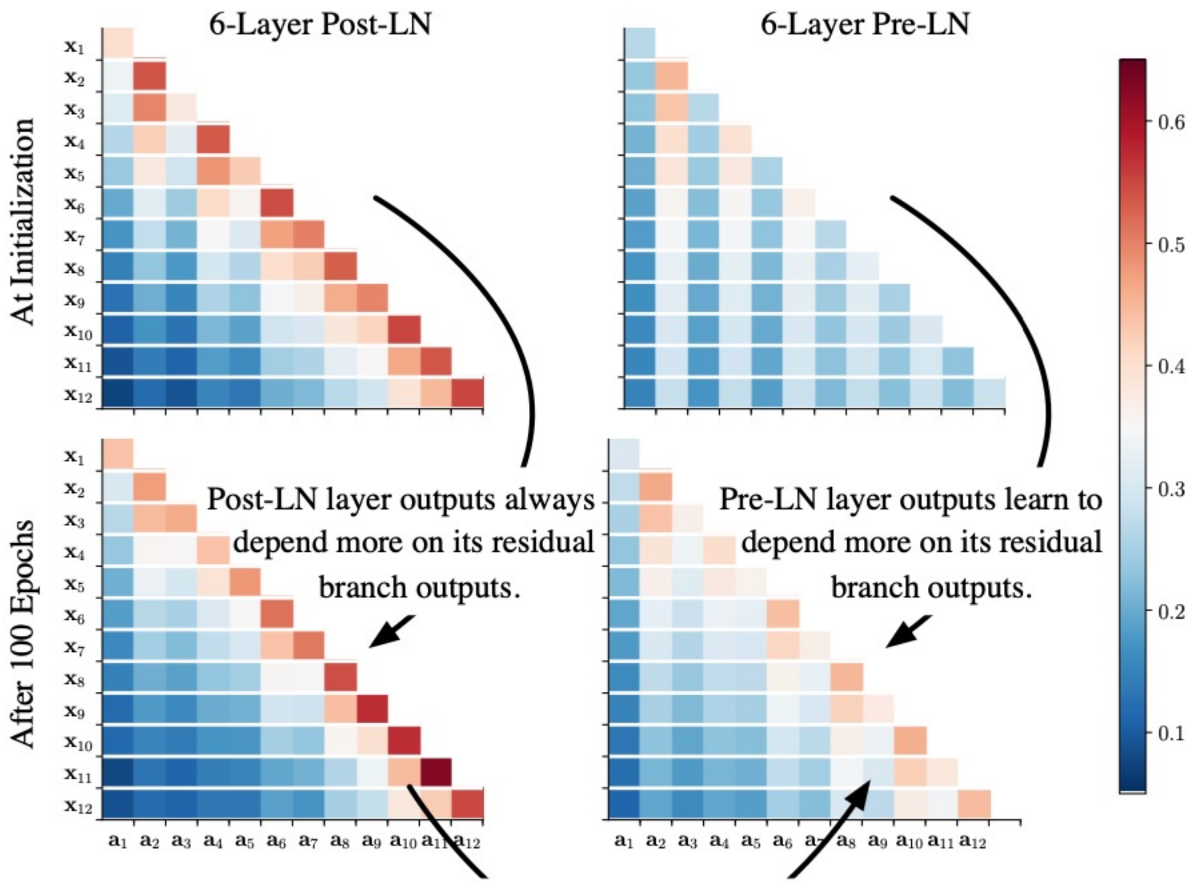
residual block안의 attention, FFN 아웃풋

$$\mathbf{x}_{2i-1}^{(o\cdot)} = \frac{\mathbf{x}_{2i-2}^{(o\cdot)} + \mathbf{a}_{2i-1}^{(o\cdot)}}{\sqrt{\mathrm{Var}[\mathbf{x}_{2i-2}^{(o\cdot)}] + \mathrm{Var}[\mathbf{a}_{2i-1}^{(o\cdot)}]}}$$



- $Var[a_{2i-2}^{O}]$ 에 따라 x_{2i-1}가 영향 받는 정도가 달라짐
- Post LN 은 새로운 레이어의 인풋으로 들어가기전에 2번의 LN을 거치게 됨(레이어가 쌓이면 post LN은 새로운 레이어에 들어가기전에 몇번의 LN을 거치게됨

Instability from amplification effect →



Comparing final models, Post-LN layer has a larger dependency on its residual branch.

$$\hat{x}_i = \sum_{j \le i} \beta_{i,j} \ \hat{a}_j$$

Normalized output $\quad \hat{x}_i = \dfrac{x_i}{\sqrt{Var\ x_i}}$

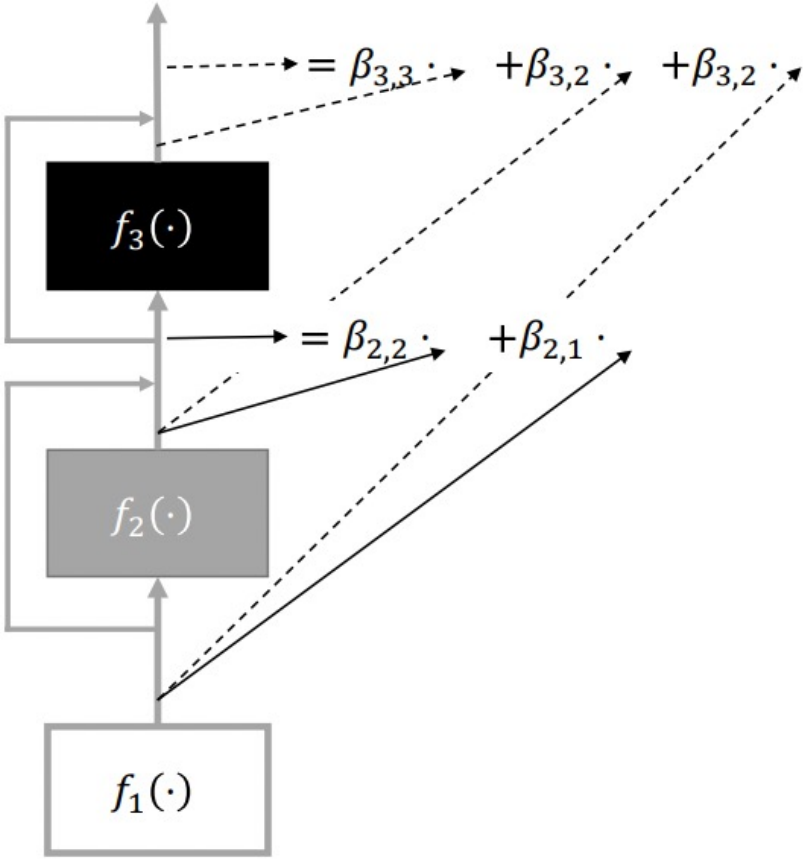
Residual output $\quad \hat{a}_i = \dfrac{a_i}{\sqrt{Var\ a_i}}$

$$\beta_{i,j} = \frac{\sqrt{Var a_j}}{\sqrt{Var\ \sum_{k \le i} a_k}}$$

I-th layer에 대한 J-th residual branch 아웃풋의 비율

~~Q. 이게 왜 layer dependency 를 의미?~~

$\beta_{i,j}$ integrates LNs and captures layer dependency



Refer $\beta_{i,i}$ as the dependency on its own residual branch.

standard deviation of j^th output

For example, $\beta_{i,j} = \frac{\boxed{\mathrm{Std}[\mathbf{a_j}]}}{\boxed{\mathrm{Std}[\sum_{k \leq i} \mathbf{a_k}]}}$ for Pre-LN

standard deviation of the sum of the first i outputs.

→ Results in fluctuation on outputs

Under some conditions, we have: $\mathrm{Var}[\boxed{\mathcal{F}(\mathbf{x_0}, W) - \mathcal{F}(\mathbf{x_0}, W + \delta)}] \approx \sum_{i=1}^{N} \boxed{\beta_{i,i}^2} C$

Model output change.

Dependency on its own residual branch (the weight for i^th residual outputs in i^th layer outputs).

Corollary 1. For Pre-LN, $\mathrm{Var}[\mathcal{F}(\mathbf{x_0}, W) - \mathcal{F}(\mathbf{x_0}, W + \delta)] = O(\log \mathrm{N})$ where N is layer #.

Corollary 2. For Post-LN, $\mathrm{Var}[\mathcal{F}(\mathbf{x_0}, W) - \mathcal{F}(\mathbf{x_0}, W + \delta)] = O(\mathrm{N})$ where N is layer #.

→ Results in fluctuation on outputs

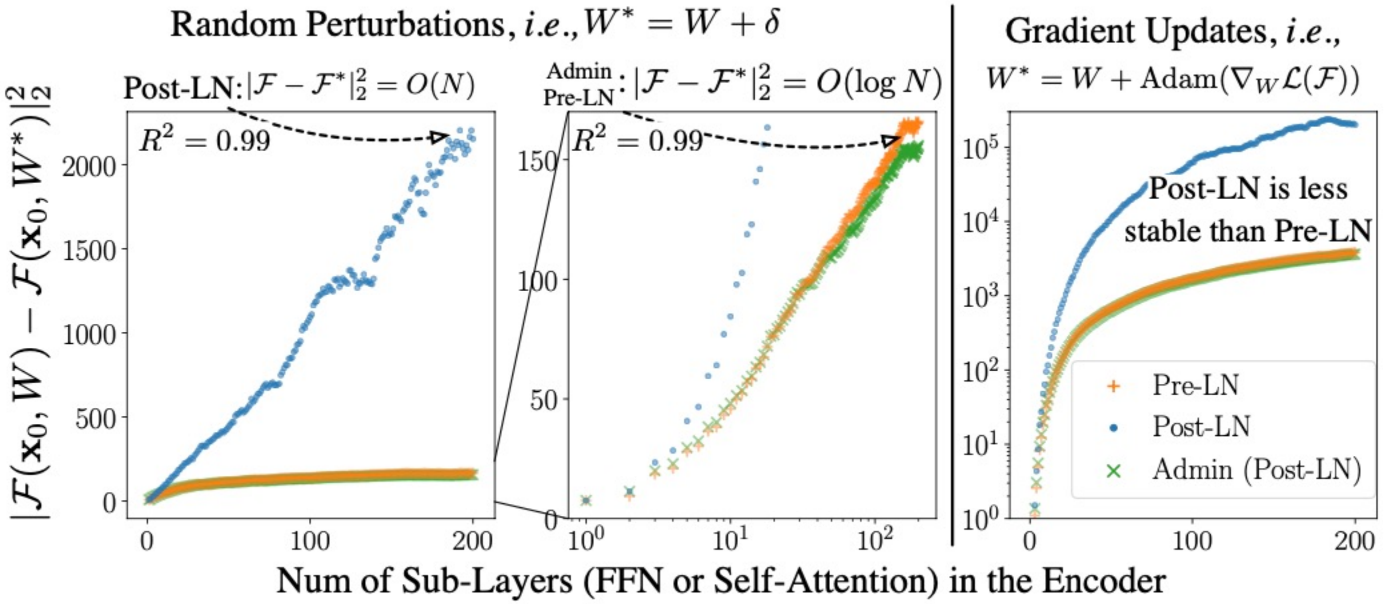


Figure 4: Encoder output changes for parameter changes, *i.e.*, $|\mathcal{F}(\mathbf{x}_0, W) - \mathcal{F}(\mathbf{x}_0, W^*)|_2^2$ where $W^* - W$ is random perturbations (left) or gradient updates (right). Intuitively, very large $|\mathcal{F} - \mathcal{F}^*|$ indicates the training to be ill-conditioned.

Adaptive model initialization

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

Post LN 이 불안정해도 좋은 성능을 가질 수 있는 potential이 있기 때문에,
Post LN을 쓰더라도 output의 fluctuation을 줄일 수 있는 방안 제안

1. Profiling : w_i는 1로 초기화하고 파라미터 업데이트 없이 일부 inference 시켜서 $var[f_i(x_i - 1)]$을 얻음

2. initialization : w_i를 $\sqrt{\sum_{j<i} var[f_i(x_i - 1)]}$로 초기화하고 전체 파라미터를 profiling 했을 때로 돌림 → 특정 parameter들을 rescaling 해줄 수 있는 것으로 볼 수 있음

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 an $O(\log N)$ output change, thus stabilizing training. Model training would become more stable in the late stage (the constant C in Theorem 2 is related to parameter gradients), and each layer has the flexibility to adjust $\boldsymbol{\omega}$ and depends more on its residual branch to calculate the layer outputs. After training finishes, Admin can be reparameterized as the conventional Post-LN structure (*i.e.*, removing $\boldsymbol{\omega}$). More implementation details are elaborated in Appendix 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 $\mathrm{Var}[\mathcal{F}(\mathbf{x}_0, W) - \mathcal{F}(\mathbf{x}_0, W^*)] \approx \sum_{i=1}^N \beta_{i,i}^2 C$.

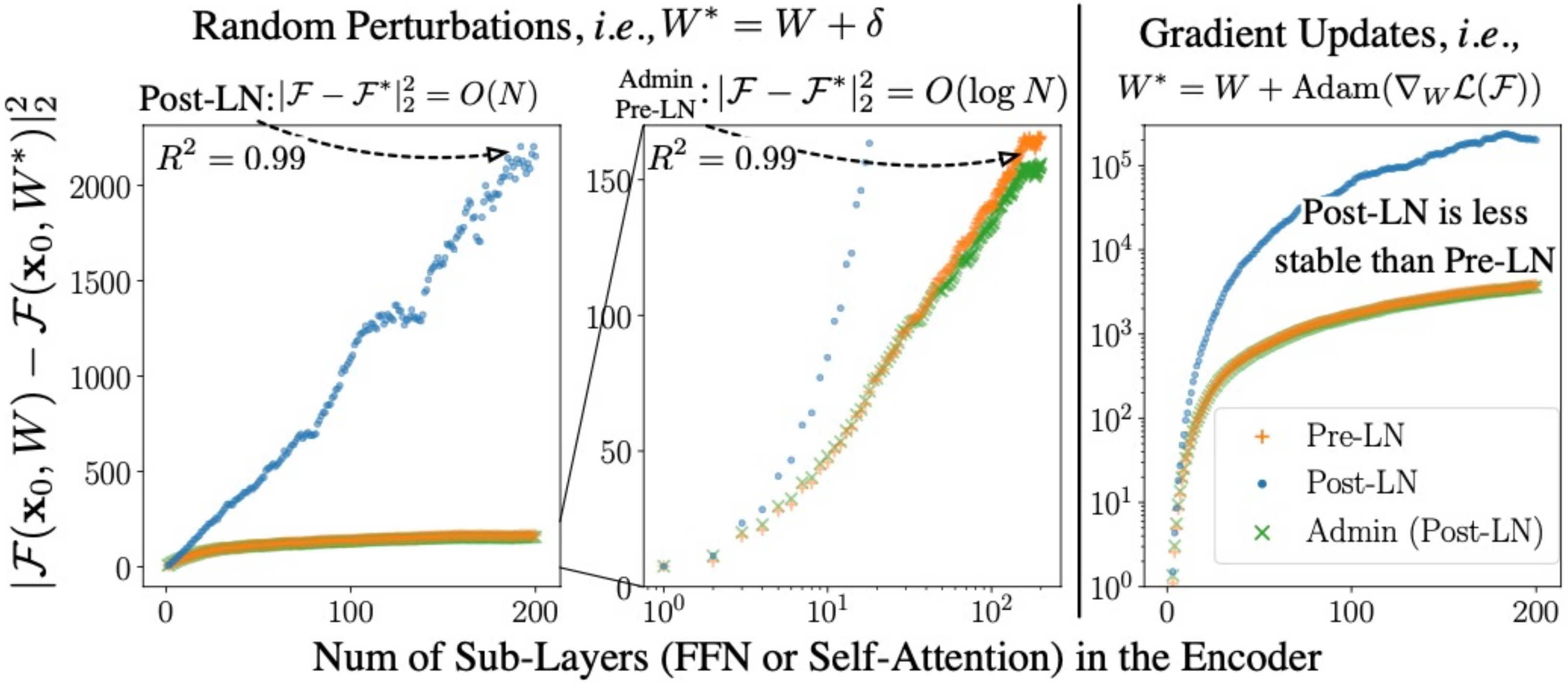
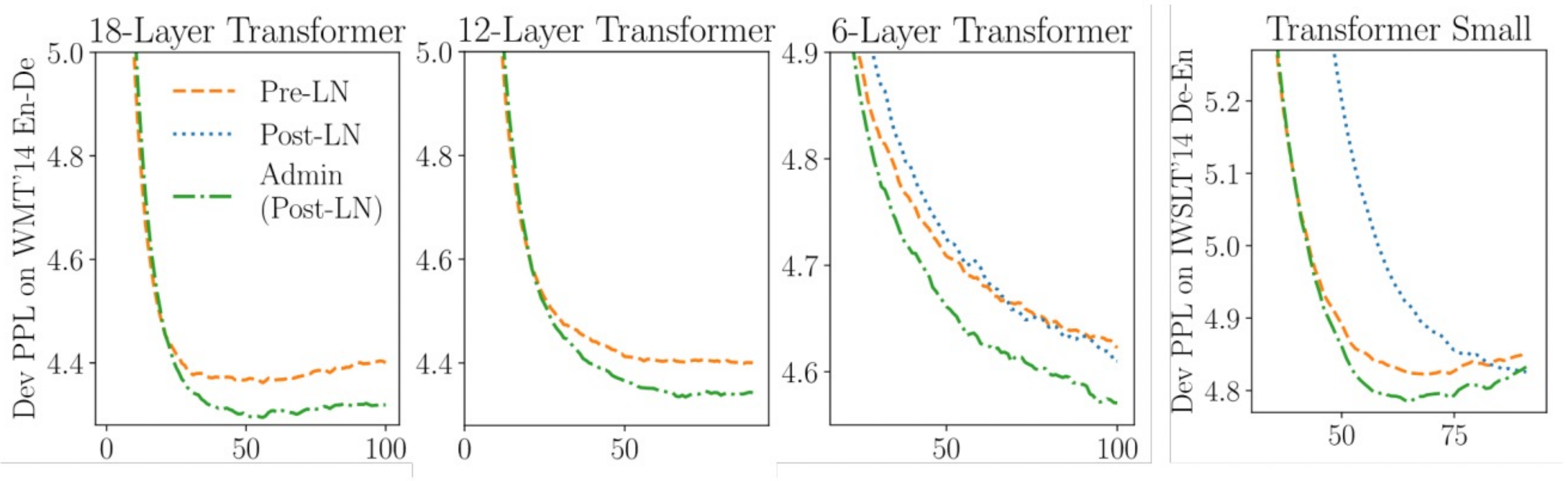


Figure 4: Encoder output changes for parameter changes, *i.e.*, $|\mathcal{F}(\mathbf{x}_0, W) - \mathcal{F}(\mathbf{x}_0, W^*)|_2^2$ where $W^* - W$ is random perturbations (left) or gradient updates (right). Intuitively, very large $|\mathcal{F} - \mathcal{F}^*|$ indicates the training to be ill-conditioned.


18-Layer Transformer
12-Layer Transformer
6-Layer Transformer
Transformer Small
Dev PPL on WMT'14 En-De
Dev PPL on IWSLT'14 De-En
Pre-LN
Post-LN
Admin (Post-LN)
5.0
4.8
4.6
4.4
4.9
4.7
5.2
5.1
0
50
75
100

Dataset	IWSLT'14 De-En	WMT'14 En-Fr		WMT'14 En-De		
Enc #-Dec #	6L-6L (small)	6L-6L	60L-12L	6L-6L	12L-12L	18L-18L
Post-LN	35.64±0.23	41.29	failed	27.80	failed	failed
Pre-LN	35.50±0.04	40.74	43.10	27.27	28.26	28.38
Admin	**35.67±0.15**	**41.47**	**43.80**	**27.90**	**28.58**	**29.03**