ACL2019

Learning Deep Transformer Models for Machine Translation

Qiang Wang[1], **Bei Li**[1], **Tong Xiao**[1,2]*, **Jingbo Zhu**[1,2], **Changliang Li**[3], **Derek F. Wong**[4], **Lidia S. Chao**[4]
[1]NLP Lab, Northeastern University, Shenyang, China
[2]NiuTrans Co., Ltd., Shenyang, China
[3]Kingsoft AI Lab, Beijing, China
[4]NLP^{2}CT Lab, University of Macau, Macau, China
wangqiangneu@gmail.com, libei_neu@outlook.com,
{xiaotong,zhujingbo}@mail.neu.edu.com,
lichangliang@kingsoft.com, {derekfw,lidiasc}@um.edu.mo

Abstract

Transformer is the state-of-the-art model in recent machine translation evaluations. Two strands of research are promising to improve models of this kind: the first uses wide networks (a.k.a. Transformer-Big) and has been the de facto standard for the development of the Transformer system, and the other uses deeper language representation but faces the difficulty arising from learning deep networks. Here, we continue the line of research on the latter. We claim that a truly deep Transformer model can surpass the Transformer-Big counterpart by 1) proper use of layer normalization and 2) a novel way of passing the combination of previous layers to the next. On WMT'16 English-German, NIST OpenMT'12 Chinese-English and larger WMT'18 Chinese-English tasks, our deep system (30/25-layer encoder) outperforms the shallow Transformer-Big/Base baseline (6-layer encoder) by 0.4∼2.4 BLEU points. As another bonus, the deep model is 1.6X smaller in size and 3X faster in training than Transformer-Big[1].

1 Introduction

Neural machine translation (NMT) models have advanced the previous state-of-the-art by learning mappings between sequences via neural networks and attention mechanisms (Sutskever et al., 2014; Bahdanau et al., 2015). The earliest of these read and generate word sequences using a series of recurrent neural network (RNN) units, and the improvement continues when 4-8 layers are stacked for a deeper model (Luong et al., 2015; Wu et al., 2016). More recently, the system based on multi-layer self-attention (call it Transformer) has shown strong results on several large-scale tasks (Vaswani et al., 2017). In particular, approaches of this kind benefit greatly from a wide network with more hidden states (a.k.a. Transformer-Big), whereas simply deepening the network has not been found to outperform the "shallow" counterpart (Bapna et al., 2018). Do deep models help Transformer? It is still an open question for the discipline.

For vanilla Transformer, learning deeper networks is not easy because there is already a relatively deep model in use[2]. It is well known that such deep networks are difficult to optimize due to the gradient vanishing/exploding problem (Pascanu et al., 2013; Bapna et al., 2018). We note that, despite the significant development effort, simply stacking more layers cannot benefit the system and leads to a disaster of training in some of our experiments.

A promising attempt to address this issue is Bapna et al. (2018)'s work. They trained a 16-layer Transformer encoder by using an enhanced attention model. In this work, we continue the line of research and go towards a much deeper encoder for Transformer. We choose encoders to study because they have a greater impact on performance than decoders and require less computational cost (Domhan, 2018). Our contributions are threefold:

- We show that the proper use of layer normalization is the key to learning deep encoders. The deep network of the encoder can be optimized smoothly by relocating the layer normalization unit. While the location of layer normalization has been discussed in recent systems (Vaswani et al., 2018; Domhan, 2018; Klein et al., 2017), as far as we know, its impact has not been studied in deep Trans-

*Corresponding author.

[1]The source code is available at https://github.com/wangqiangneu/dlcl

[2]For example, a standard Transformer encoder has 6 layers. Each of them consists of two sub-layers. More sub-layers are involved on the decoder side.

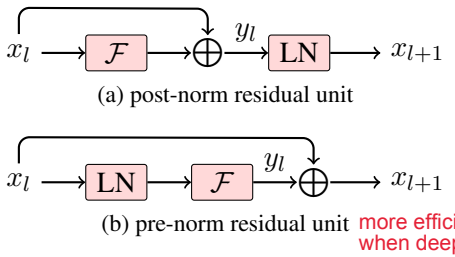


Figure 1: Examples of pre-norm residual unit and post-norm residual unit. $\mathcal{F}$ = sub-layer, and LN = layer normalization.

former.

- Inspired by the linear multi-step method in numerical analysis (Ascher and Petzold, 1998), we propose an approach based on dynamic linear combination of layers (DLCL) to memorizing the features extracted from all preceding layers. This overcomes the problem with the standard residual network where a residual connection just relies on the output of one-layer ahead and may forget the earlier layers.

- We successfully train a 30-layer encoder, far surpassing the deepest encoder reported so far (Bapna et al., 2018). To our best knowledge, this is the deepest encoder used in NMT.

On WMT'16 English-German, NIST OpenMT'12 Chinese-English, and larger WMT'18 Chinese-English translation tasks, we show that our deep system (30/25-layer encoder) yields a BLEU improvement of 1.3∼2.4 points over the base model (Transformer-Base with 6 layers). It even outperforms Transformer-Big by 0.4∼0.6 BLEU points, but requires 1.6X fewer model parameters and 3X less training time. More interestingly, our deep model is 10% faster than Transformer-Big in inference speed.

2 Post-Norm and Pre-Norm Transformer

The Transformer system and its variants follow the standard encoder-decoder paradigm. On the encoder side, there are a number of identical stacked layers. Each of them is composed of a self-attention sub-layer and a feed-forward sub-layer. The attention model used in Transformer is multi-head attention, and its output is fed into a fully connected feed-forward network. Likewise, the decoder has another stack of identical layers. It has an encoder-decoder attention sub-layer in addition to the two sub-layers used in each encoder layer. In general, because the encoder and the decoder share a similar architecture, we can use the same method to improve them. In the section, we discuss a more general case, not limited to the encoder or the decoder.

2.1 Model Layout

For Transformer, it is not easy to train stacked layers on neither the encoder-side nor the decoder-side. Stacking all these sub-layers prevents the efficient information flow through the network, and probably leads to the failure of training. Residual connections and layer normalization are adopted for a solution. Let $\mathcal{F}$ be a sub-layer in encoder or decoder, and θ_l be the parameters of the sub-layer. A residual unit is defined to be (He et al., 2016b):

$$x_{l+1} = f(y_l) \quad (1)$$

$$y_l = x_l + \mathcal{F}(x_l; \theta_l) \quad (2)$$

where x_l and x_{l+1} are the input and output of the l-th sub-layer, and y_l is the intermediate output followed by the post-processing function $f(\cdot)$. In this way, x_l is explicitly exposed to y_l (see Eq. (2)).

Moreover, layer normalization is adopted to reduce the variance of sub-layer output because hidden state dynamics occasionally causes a much longer training time for convergence. There are two ways to incorporate layer normalization into the residual network.

- **Post-Norm**. In early versions of Transformer (Vaswani et al., 2017), layer normalization is placed after the element-wise residual addition (see Figure 1(a)), like this:

$$x_{l+1} = \text{LN}(x_l + \mathcal{F}(x_l; \theta_l)) \quad (3)$$

where $\text{LN}(\cdot)$ is the layer normalization function, whose parameter is dropped for simplicity. It can be seen as a post-processing step of the output (i.e., $f(x) = \text{LN}(x)$).

- **Pre-Norm**. In recent implementations (Klein et al., 2017; Vaswani et al., 2018; Domhan, 2018), layer normalization is applied to the input of every sub-layer (see Figure 1(b)):

$$x_{l+1} = x_l + \mathcal{F}(\text{LN}(x_l); \theta_l) \quad (4)$$

Eq. (4) regards layer normalization as a part of the sub-layer, and does nothing for post-processing of the residual connection (i.e., $f(x) = x$).[3]

Both of these methods are good choices for implementation of Transformer. In our experiments, they show comparable performance in BLEU for a system based on a 6-layer encoder (Section 5.1).

2.2 On the Importance of Pre-Norm for Deep Residual Network

The situation is quite different when we switch to deeper models. More specifically, we find that pre-norm is more efficient for training than post-norm if the model goes deeper. This can be explained by seeing back-propagation which is the core process to obtain gradients for parameter update. Here we take a stack of L sub-layers as an example. Let $\mathcal{E}$ be the loss used to measure how many errors occur in system prediction, and x_L be the output of the topmost sub-layer. For post-norm Transformer, given a sub-layer l, the differential of $\mathcal{E}$ with respect to x_l can be computed by the chain rule, and we have

$$\frac{\partial \mathcal{E}}{\partial x_l} = \frac{\partial \mathcal{E}}{\partial x_L} \times \prod_{k=l}^{L-1} \frac{\partial \mathrm{LN}(y_k)}{\partial y_k} \times \prod_{k=l}^{L-1} \left(1 + \frac{\partial \mathcal{F}(x_k; \theta_k)}{\partial x_k}\right) \quad (5)$$

where $\prod_{k=l}^{L-1} \frac{\partial \mathrm{LN}(y_k)}{\partial y_k}$ means the backward pass of the layer normalization, and $\prod_{k=l}^{L-1}(1 + \frac{\partial \mathcal{F}(x_k;\theta_k)}{\partial x_k})$ means the backward pass of the sub-layer with the residual connection. Likewise, we have the gradient for pre-norm [4]:

$$\frac{\partial \mathcal{E}}{\partial x_l} = \frac{\partial \mathcal{E}}{\partial x_L} \times \left(1 + \sum_{k=l}^{L-1} \frac{\partial \mathcal{F}(\mathrm{LN}(x_k); \theta_k)}{\partial x_l}\right) \quad (6)$$

Obviously, Eq. (6) establishes a direct way to pass error gradient $\frac{\partial \mathcal{E}}{\partial x_L}$ from top to bottom. Its merit lies in that the number of product items on the right side does not depend on the depth of the stack.

In contrast, Eq. (5) is inefficient for passing gradients back because the residual connection is not a bypass of the layer normalization unit (see Figure 1(a)). Instead, gradients have to be passed through $\mathrm{LN}(\cdot)$ of each sub-layer. It in turn introduces term $\prod_{k=l}^{L-1} \frac{\partial \mathrm{LN}(y_k)}{\partial y_k}$ into the right hand side of Eq. (5), and poses a higher risk of gradient vanishing or exploring if L goes larger. This was confirmed by our experiments in which we successfully trained a pre-norm Transformer system with a 20-layer encoder on the WMT English-German task, whereas the post-norm Transformer system failed to train for a deeper encoder (Section 5.1).

3 Dynamic Linear Combination of Layers

The residual network is the most common approach to learning deep networks, and plays an important role in Transformer. In principle, residual networks can be seen as instances of the ordinary differential equation (ODE), behaving like the forward Euler discretization with an initial value (Chang et al., 2018; Chen et al., 2018b). Euler's method is probably the most popular first-order solution to ODE. But it is not yet accurate enough. A possible reason is that only one previous step is used to predict the current value [5](Butcher, 2003). In MT, the single-step property of the residual network makes the model "forget" distant layers (Wang et al., 2018b). As a result, there is no easy access to features extracted from lower-level layers if the model is very deep.

具有初始值的正向欧拉离散化

Here, we describe a model which makes direct links with all previous layers and offers efficient access to lower-level representations in a deep stack. We call it dynamic linear combination of layers (DLCL). The design is inspired by the linear multi-step method (LMM) in numerical ODE (Ascher and Petzold, 1998). Unlike Euler's method, LMM can effectively reuse the information in the previous steps by linear combination to achieve a higher order. Let $\{y_0, ..., y_l\}$ be the output of layers $0 \sim l$. The input of layer $l+1$ is defined to be

$$x_{l+1} = \mathcal{G}(y_0, \ldots, y_l) \quad (7)$$

where $\mathcal{G}(\cdot)$ is a linear function that merges previously generated values $\{y_0, ..., y_l\}$ into a new value. For pre-norm Transformer, we define $\mathcal{G}(\cdot)$

[3] We need to add an additional function of layer normalization to the top layer to prevent the excessively increased value caused by the sum of unnormalized output.

[4] For a detailed derivation, we refer the reader to Appendix A.

[5] Some of the other single-step methods, e.g. the Runge-Kutta method, can obtain a higher order by taking several intermediate steps (Butcher, 2003). Higher order generally means more accurate.

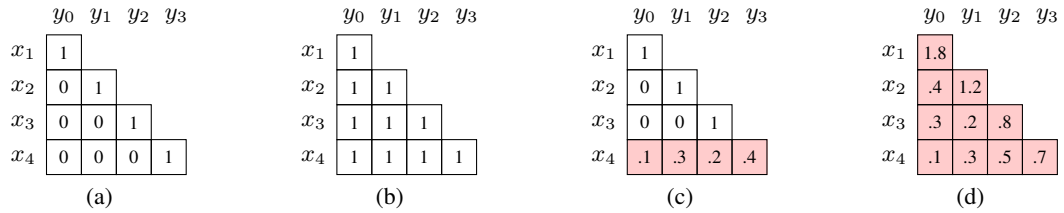


Figure 2: Connection weights for 3-layer encoder: (a) residual connection (He et al., 2016a), (b) dense residual connection (Britz et al., 2017; Dou et al., 2018), (c) multi-layer representation fusion (Wang et al., 2018b)/transparent attention (Bapna et al., 2018) and (d) our approach. y_0 denotes the input embedding. Red denotes the weights are learned by model.

to be

$$\mathcal{G}(y_0, \ldots, y_l) = \sum_{k=0}^{l} W_k^{(l+1)} \mathrm{LN}(y_k) \quad (8)$$

where $W_k^{l+1} \in \mathbb{R}$ is a learnable scalar and weights each incoming layer in a linear manner. Eq. (8) provides a way to learn preference of layers in different levels of the stack. Even for the same incoming layer, its contribution to succeeding layers could be different (e.g. $W_k^i \neq W_k^k$) . Also, the method is applicable to the post-norm Transformer model. For post-norm, $\mathcal{G}(\cdot)$ can be redefined as:

$$\mathcal{G}(y_0, \ldots, y_l) = \mathrm{LN}\Big(\sum_{k=0}^{l} W_k^{(l+1)} y_k\Big) \quad (9)$$

权重可学习；
历史窗口尺寸任意

Comparison to LMM. DLCL differs from LMM in two aspects, though their fundamental model is the same. First, DLCL learns weights in an end-to-end fashion rather than assigning their values deterministically, e.g. by polynomial interpolation. This offers a more flexible way to control the model behavior. Second, DLCL has an arbitrary size of the past history window, while LMM generally takes a limited history into account (Lóczi, 2018). Also, recent work shows successful applications of LMM in computer vision, but only two previous steps are used in their LMM-like system (Lu et al., 2018).

Comparison to existing neural methods. Note that DLCL is a very general approach. For example, the standard residual network is a special case of DLCL, where $W_l^{l+1} = 1$, and $W_k^{l+1} = 0$ for $k < l$. Figure (2) compares different methods of connecting a 3-layer network. We see that the densely residual network is a fully-connected network with a uniform weighting schema (Britz et al., 2017; Dou et al., 2018). Multi-layer representation fusion (Wang et al., 2018b) and transparent attention (call it TA) (Bapna et al., 2018) methods can learn a weighted model to fuse layers but they are applied to the topmost layer only. The DLCL model can cover all these methods. It provides ways of weighting and connecting layers in the entire stack. We emphasize that although the idea of weighting the encoder layers by a learnable scalar is similar to TA, there are two key differences: 1) Our method encourages earlier interactions between layers during the encoding process, while the encoder layers in TA are combined until the standard encoding process is over; 2) For an encoder layer, instead of learning a unique weight for each decoder layer like TA, we make a separate weight for each successive encoder layers. In this way, we can create more connections between layers[6].

4 Experimental Setup

We first evaluated our approach on WMT'16 English-German (En-De) and NIST'12 Chinese-English (Zh-En-Small) benchmarks respectively. To make the results more convincing, we also experimented on a larger WMT'18 Chinese-English dataset (Zh-En-Large) with data augmentation by back-translation (Sennrich et al., 2016a).

4.1 Datasets and Evaluation

For the En-De task, to compare with Vaswani et al. (2017)'s work, we use the same 4.5M preprocessed data [7], which has been tokenized and

[6] Let the encoder depth be M and the decoder depth be N ($M > N$ for a deep encoder model). Then TA newly adds $\mathcal{O}(M \times N)$ connections, which are fewer than ours of $\mathcal{O}(M^2)$

[7] https://drive.google.com/uc?export=download&id=0B_bZck-ksdkpM25jRUN2X2UxMm8

Model		Param.	Batch (×4096)	Updates (×100k)	†Times	BLEU	Δ
Vaswani et al. (2017) (Base)		65M	1	1	reference	27.3	-
Bapna et al. (2018)-deep (Base, 16L)		137M	-	-	-	28.0	-
Vaswani et al. (2017) (Big)		213M	1	3	3x	28.4	-
Chen et al. (2018a) (Big)		379M	16	†0.075	1.2x	28.5	-
He et al. (2018) (Big)		†210M	1	-	-	29.0	-
Shaw et al. (2018) (Big)		†210M	1	3	3x	29.2	-
Dou et al. (2018) (Big)		356M	1	-	-	29.2	-
Ott et al. (2018) (Big)		210M	14	0.25	3.5x	29.3	-
post-norm	Transformer (Base)	62M	1	1	1x	27.5	reference
	Transformer (Big)	211M	1	3	3x	28.8	+1.3
	Transformer-deep (Base, 20L)	106M	2	0.5	1x	failed	failed
	DLCL (Base)	62M	1	1	1x	27.6	+0.1
	DLCL-deep (Base, 25L)	121M	2	0.5	1x	29.2	+1.7
pre-norm	Transformer (Base)	62M	1	1	1x	27.1	reference
	Transformer (Big)	211M	1	3	3x	28.7	+1.6
	Transformer-deep (Base, 20L)	106M	2	0.5	1x	28.9	+1.8
	DLCL (Base)	62M	1	1	1x	27.3	+0.2
	DLCL-deep (Base, 30L)	137M	2	0.5	1x	**29.3**	**+2.2**

Table 1: BLEU scores [%] on English-German translation. `Batch` indicates the corresponding batch size if running on 8 GPUs. `Times` $\propto$ `Batch`×`Updates`, which can be used to approximately measure the required training time. † denotes an estimate value. Note that "-deep" represents the best-achieved result as depth changes.

jointly byte pair encoded (BPE) (Sennrich et al., 2016b) with 32k merge operations using a shared vocabulary [8]. We use *newstest2013* for validation and *newstest2014* for test.

For the Zh-En-Small task, we use parts of the bitext provided within NIST'12 OpenMT[9]. We choose NIST MT06 as the validation set, and MT04, MT05, MT08 as the test sets. All the sentences are word segmented by the tool provided within NiuTrans (Xiao et al., 2012). We remove the sentences longer than 100 and end up with about 1.9M sentence pairs. Then BPE with 32k operations is used for both sides independently, resulting in a 44k Chinese vocabulary and a 33k English vocabulary respectively.

For the Zh-En-Large task, we use exactly the same 16.5M dataset as Wang et al. (2018a), composing of 7.2M-sentence CWMT corpus, 4.2M-sentence UN and News-Commentary combined corpus, and back-translation of 5M-sentence monolingual data from NewsCraw2017. We refer the reader to Wang et al. (2018a) for the details.

For evaluation, we first average the last 5 checkpoints, each of which is saved at the end of an epoch. And then we use beam search with a beam size of 4/6 and length penalty of 0.6/1.0 for En-De/Zh-En tasks respectively. We measure case-sensitive/insensitive tokenized BLEU by *multi-bleu.perl* for En-De and Zh-En-Small respectively, while case-sensitive detokenized BLEU is reported by the official evaluation script *mteval-v13a.pl* for Zh-En-Large. Unless noted otherwise we run each experiment three times with different random seeds and report the mean of the BLEU scores across runs[10].

4.2 Model and Hyperparameters

All experiments run on *fairseq-py*[11] with 8 NVIDIA Titan V GPUs. For the post-norm Transformer baseline, we replicate the model setup of Vaswani et al. (2017). All models are optimized by Adam (Kingma and Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-8}$. In training warmup (`warmup` = 4000 steps), the learning rate linearly increases from 10^{-7} to `lr` $=7\times10^{-4}/5\times10^{-4}$ for

[8] The tokens with frequencies less than 5 are filtered out from the shared vocabulary.

[9] LDC2000T46, LDC2000T47, LDC2000T50, LDC2003E14, LDC2005T10, LDC2002E18, LDC2007T09, LDC2004T08

[10] Due to resource constraints, all experiments on Zh-En-Large task only run once.

[11] https://github.com/pytorch/fairseq

Model (Base, 16L)		BLEU
post-norm	Bapna et al. (2018)	28.0
	Transformer	failed
	DLCL	28.4
pre-norm	Transformer	28.0
	DLCL	28.2

Table 2: Compare with Bapna et al. (2018) on WMT'16 English-German translation under a 16-layer encoder.

Transformer-Base/Big respectively, after which it is decayed proportionally to the inverse square root of the current step. Label smoothing ε_{ls}=0.1 is used as regularization.

For the pre-norm Transformer baseline, we follow the setting as suggested in tensor2tensor[12]. More specifically, the attention dropout P_{att} = 0.1 and feed-forward dropout P_{ff} = 0.1 are additionally added. And some hyper-parameters for optimization are changed accordingly: β_2 = 0.997, `warmup` = 8000 and `lr` = $10^{-3}/7\times10^{-4}$ for Transformer-Base/Big respectively.

For both the post-norm and pre-norm baselines, we batch sentence pairs by approximate length and restrict input and output tokens per batch to `batch` = 4096 per GPU. We set the update steps according to corresponding data sizes. More specifically, the Transformer-Base/Big is updated for 100k/300k steps on the En-De task as Vaswani et al. (2017), 50k/100k steps on the Zh-En-Small task, and 200k/500k steps on the Zh-En-Large task.

In our model, we use the dynamic linear combination of layers for both encoder and decoder. For efficient computation, we only combine the output of a complete layer rather than a sub-layer. It should be noted that for deep models (e.g. $L \geq 20$), it is hard to handle a full batch in a single GPU due to memory size limitation. We solve this issue by accumulating gradients from two small batches (e.g. batch = 2048) before each update (Ott et al., 2018). In our primitive experiments, we observed that training with larger batches and learning rates worked well for deep models. Therefore all the results of deep models are reported with `batch` = 8192, `lr` = 2×10^{-3} and `warmup` = 16,000 unless otherwise stated. For fairness, we only use half of the updates of baseline (e.g. `update` = 50k) to ensure the same amount of data that we actually see in training. We report the details in Appendix B.

[12] https://github.com/tensorflow/tensor2tensor

5 Results

5.1 Results on the En-De Task

In Table 1, we first report results on WMT En-De where we compare to the existing systems based on self-attention. Obviously, while almost all previous results based on Transformer-Big (marked by Big) have higher BLEU than those based on Transformer-Base (marked by Base), larger parameter size and longer training epochs are required.

As for our approach, considering the post-norm case first, we can see that our Transformer baselines are superior to Vaswani et al. (2017) in both `Base` and `Big` cases. When increasing the encoder depth, e.g. L = 20, the vanilla Transformer failed to train, which is consistent with Bapna et al. (2018). We attribute it to the vanishing gradient problem based on the observation that the gradient norm in the low layers (e.g. embedding layer) approaches 0. On the contrary, post-norm DLCL solves this issue and achieves the best result when L = 25.

The situation changes when switching to pre-norm. While it slightly underperforms the post-norm counterpart in shallow networks, pre-norm Transformer benefits more from the increase in encoder depth. More concretely, pre-norm Transformer achieves optimal result when L=20 (see Figure 3(a)), outperforming the 6-layer baseline by 1.8 BLEU points. It indicates that pre-norm is easier to optimize than post-norm in deep networks. Beyond that, we successfully train a 30-layer encoder by our method, resulting in a further improvement of 0.4 BLEU points. This is 0.6 BLEU points higher than the pre-norm Transformer-Big. It should be noted that although our best score of 29.3 is the same as Ott et al. (2018), our approach only requires 3.5X fewer training epochs than theirs.

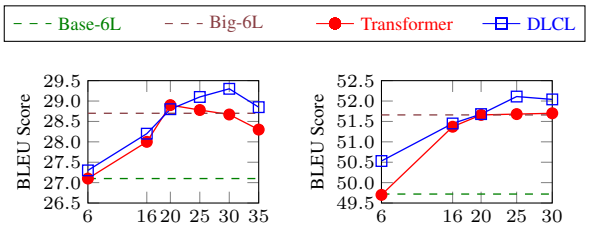
To fairly compare with *transparent attention* (TA) (Bapna et al., 2018), we separately list the results using a 16-layer encoder in Table 2. It can be seen that pre-norm Transformer obtains the same BLEU score as TA without the requirement of complicated attention design. However, DLCL in both post-norm and pre-norm cases outperform TA. It should be worth that TA achieves the best result when encoder depth is 16, while we can fur-

Model (pre-norm)	Param.	Valid.	MT04	MT05	MT08	Average
Transformer (Base)	84M	51.27	54.41	49.43	45.33	49.72
Transformer (Big)	257M	52.30	55.37	52.21	47.40	51.66
Transformer-deep (Base, 25L)	144M	52.50	55.80	51.98	47.26	51.68
DLCL (Base)	84M	51.61	54.91	50.58	46.11	50.53
DLCL-deep (Base, 25L)	144M	**53.57**	**55.91**	**52.30**	**48.12**	**52.11**

Table 3: BLEU scores [%] on NIST'12 Chinese-English translation.

Model	Param.	newstest17	newstest18	$\Delta_{avg.}$
Wang et al. (2018a) (post-norm, Base)	102.1M	25.9	-	-
pre-norm Transformer (Base)	102.1M	25.8	25.9	reference
pre-norm Transformer (Big)	292.4M	26.4	27.0	+0.9
pre-norm DLCL-deep (Base, 25L)	161.5M	26.7	27.1	+1.0
pre-norm DLCL-deep (Base, 30L)	177.2M	**26.9**	**27.4**	**+1.3**

Table 4: BLEU scores [%] on WMT'18 Chinese-English translation.



(a) WMT En-De (b) NIST Zh-En

Figure 3: BLEU scores [%] against the encoder depth for pre-norm Transformer and pre-norm DLCL on English-German and Chinese-English tasks.

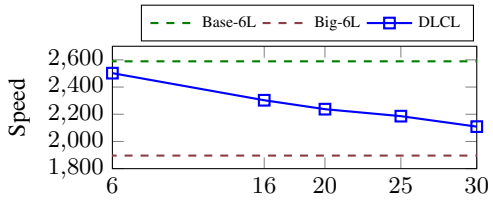


Figure 4: GPU generation speed (target tokens/sec.) against the depth of encoder for pre-norm DLCL on English-German task (batch size = 32, beam size = 4).

ther improve performance by training deeper encoders.

5.2 Results on the Zh-En-Small Task

Seen from the En-De task, pre-norm is more effective than the post-norm counterpart in deep networks. Therefore we evaluate our method in the case of pre-norm on the Zh-En task. As shown in Table 3, firstly DLCL is superior to the baseline when the network's depth is shallow. Interestingly, both Transformer and DLCL achieve the best results when we use a 25-layer encoder. The 25-layer Transformer can approach the performance of Transformer-Big, while our deep model outperforms it by about 0.5 BLEU points under the equivalent parameter size. It confirms that our approach is a good alternative to Transformer no matter how deep it is.

5.3 Results on the Zh-En-Large Task

While deep Transformer models, in particular the deep pre-norm DLCL, show better results than Transformer-Big on En-De and Zh-En-Small tasks, both data sets are relatively small, and the improved performance over Transformer-Big might be partially due to over-fitting in the wider model. For a more challenging task , we report the results on Zh-En-Large task in Table 4. We can see that the 25-layer pre-norm DLCL slightly surpassed Transformer-Big, and the superiority is bigger when using a 30-layer encoder. This result indicates that the claiming of the deep network defeating Transformer-Big is established and is not affected by the size of the data set.

6 Analysis

6.1 Effect of Encoder Depth

In Figure 3, we plot BLEU score as a function of encoder depth for pre-norm Transformer and DLCL on En-De and Zh-En-Small tasks. First of all, both methods benefit from an increase in encoder depth at the beginning. Remarkably, when the encoder depth reaches 20, both of the two deep models can achieve comparable performance to Transformer-Big, and even exceed it when the en-

coder depth is further increased in DLCL. Note that pre-norm Transformer degenerates earlier and is less robust than DLCL when the depth is beyond 20. However, a deeper network (>30 layers) does not bring more benefits. Worse still, deeper networks consume a lot of memory, making it impossible to train efficiently.

We also report the inference speed on GPU in Figure 4. As expected, the speed decreases linearly with the number of encoder layers. Nevertheless, our system with a 30-layer encoder is still faster than Transformer-Big, because the encoding process is independent of beam size, and runs only once. In contrast, the decoder suffers from severe autoregressive problems.

6.2 Effect of Decoder Depth

Enc. Depth	Dec. Depth	BLEU	Speed
6	4	27.12	3088.3
6	6	27.33	2589.2
6	8	27.42	2109.6

Table 5: Tokenized BLEU scores [%] and GPU generation speed (target tokens per second) in pre-norm Transformer (Base) on the test set of WMT English-German (batch size = 32, beam size = 4).

Table 5 shows the effects of decoder depth on BLEU and inference speed on GPU. Different from encoder, increasing the depth of decoder only yields a slight BLEU improvement, but the cost is high: for every two layers added, the translation speed drops by approximate 500 tokens evenly. It indicates that exploring deep encoders may be more promising than deep decoders for NMT.

6.3 Ablation Study

We report the ablation study results in Table 6. We first observe a modest decrease when removing the introduced layer normalization in Eq. (8). Then we try two methods to replace learnable weights with constant weights: *All-One* ($W_j^i = 1$) and *Average* ($W_j^i = 1/(i+1)$). We can see that these two methods consistently hurt performance, in particular in the case of All-One. It indicates that making the weights learnable is important for our model. Moreover, removing the added layer normalization in the Average model makes BLEU score drop by 0.28, which suggests that adding layer normalization helps more if we use the constant weights. In addition, we did two interesting experiments on big models. The first one is to replace the base encoder with a big encoder in pre-norm Transformer-Base. The other one is to use DLCL to train a deep-and-wide Transformer (12 layers). Although both of them benefit from the increased network capacity, the gain is less than the "thin" counterpart in terms of BLEU, parameter size, and training efficiency.

Model	BLEU
pre-norm DLCL-20L	28.80
- layer norm.	28.67
- learnable weight (fix 1)	28.22
- learnable weight (fix 1/N)	28.51
- layer norm.	28.23
pre-norm Transformer-Base	27.11
+ big encoder	27.59
pre-norm Transformer-Big	28.72
+ 12-layer encoder (DLCL)	29.17

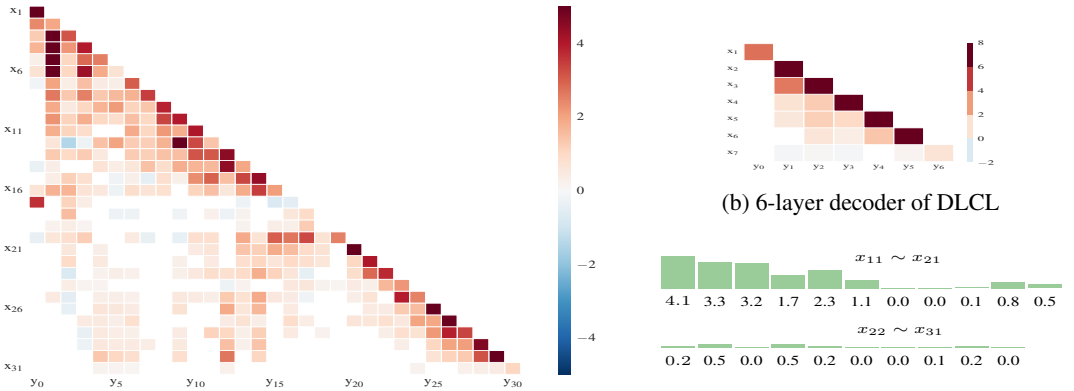
Table 6: Ablation results by tokenized BLEU [%] on the test set of WMT English-German translation.

6.4 Visualization on Learned Weights

We visually present the learned weights matrices of the 30-layer encoder (Figure 5(a)) and its 6-layer decoder (Figure 5(b)) in our pre-norm DLCL-30L model on En-De task. For a clearer contrast, we mask out the points with an absolute value of less than 0.1 or 5% of the maximum per row. We can see that the connections in the early layers are dense, but become sparse as the depth increases. It indicates that making full use of earlier layers is necessary due to insufficient information at the beginning of the network. Also, we find that most of the large weight values concentrate on the right of the matrix, which indicates that the impact of the incoming layer is usually related to the distance between the outgoing layer. Moreover, for a fixed layer's output y_i, it is obvious that its contribution to successive layers changes dynamically (one column). To be clear, we extract the weights of y_{10} in Figure 5(c). In contrast, in most previous paradigms of dense residual connection, the output of each layer remains fixed for subsequent layers.

7 Related Work

Deep Models. Deep models have been explored in the context of neural machine translation since the emergence of RNN-based models. To ease optimization, researchers tried to reduce the number of non-linear transitions (Zhou et al.,



(a) 30-layer encoder of DLCL

(b) 6-layer decoder of DLCL

(c) Weight distribution of y_{10} in the encoder

Figure 5: A visualization example of learned weights in our 30-layer pre-norm DLCL model.

2016; Wang et al., 2017). But these attempts are limited to the RNN architecture and may not be straightforwardly applicable to the current Transformer model. Perhaps, the most relevant work to what is doing here is Bapna et al. (2018)'s work. They pointed out that vanilla Transformer was hard to train if the depth of the encoder was beyond 12. They successfully trained a 16-layer Transformer encoder by attending the combination of all encoder layers to the decoder. In their approach, the encoder layers are combined just after the encoding is completed, but not during the encoding process. In contrast, our approach allows the encoder layers to interact earlier, which has been proven to be effective in machine translation (He et al., 2018) and text match (Lu and Li, 2013). In addition to machine translation, deep Transformer encoders are also used for language modeling (Devlin et al., 2018; Al-Rfou et al., 2018). For example, Al-Rfou et al. (2018) trained a character language model with a 64-layer Transformer encoder by resorting to auxiliary losses in intermediate layers. This method is orthogonal to our DLCL method, though it is used for language modeling, which is not a very heavy task.

Densely Residual Connections. Densely residual connections are not new in NMT. They have been studied for different architectures, e.g., RNN (Britz et al., 2017) and Transformer (Dou et al., 2018). Some of the previous studies fix the weight of each layer to a constant, while others learn a weight distribution by using either the self-attention model (Wang et al., 2018b) or a softmax-normalized learnable vector (Peters et al., 2018). They focus more on learning connections from lower-level layers to the topmost layer. Instead, we introduce additional connectivity into the network and learn more densely connections for each layer in an end-to-end fashion.

8 Conclusion

We have studied deep encoders in Transformer. We have shown that the deep Transformer models can be easily optimized by proper use of layer normalization, and have explained the reason behind it. Moreover, we proposed an approach based on a dynamic linear combination of layers and successfully trained a 30-layer Transformer system. It is the deepest encoder used in NMT so far. Experimental results show that our thin-but-deep encoder can match or surpass the performance of Transformer-Big. Also, its model size is 1.6X smaller. In addition, it requires 3X fewer training epochs and is 10% faster for inference.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China (Grant Nos. 61876035, 61732005, 61432013 and 61672555), the Fundamental Research Funds for the Central Universities (Grant No. N181602013), the Joint Project of FDCT-NSFC (Grant No. 045/2017/AFJ), the MYRG from the University of Macau (Grant No. MYRG2017-00087-FST).

References

Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. 2018. Character-level lan-

guage modeling with deeper self-attention. *arXiv preprint arXiv:1808.04444*.

Uri M Ascher and Linda R Petzold. 1998. *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. Siam.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *In Proceedings of the 3rd International Conference on Learning Representations*.

Ankur Bapna, Mia Chen, Orhan Firat, Yuan Cao, and Yonghui Wu. 2018. Training deeper neural machine translation models with transparent attention. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3028–3033.

Denny Britz, Anna Goldie, Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*.

J C Butcher. 2003. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, New York, NY.

Bo Chang, Lili Meng, Eldad Haber, Frederick Tung, and David Begert. 2018. Multi-level residual networks from dynamical systems view. In *International Conference on Learning Representations*.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Niki Parmar, Mike Schuster, Zhifeng Chen, et al. 2018a. The best of both worlds: Combining recent advances in neural machine translation. *arXiv preprint arXiv:1804.09849*.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. 2018b. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6572–6583. Curran Associates, Inc.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Tobias Domhan. 2018. How much attention do you need? a granular analysis of neural machine translation architectures. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1799–1808.

Zi-Yi Dou, Zhaopeng Tu, Xing Wang, Shuming Shi, and Tong Zhang. 2018. Exploiting deep representations for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4253–4262.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016b. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer.

Tianyu He, Xu Tan, Yingce Xia, Di He, Tao Qin, Zhibo Chen, and Tie-Yan Liu. 2018. Layer-wise coordination between encoder and decoder for neural machine translation. In *Advances in Neural Information Processing Systems*, pages 7955–7965.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. *Proceedings of ACL 2017, System Demonstrations*, pages 67–72.

Lajos Lóczi. 2018. Exact optimal values of stepsize coefficients for boundedness of linear multistep methods. *Numerical Algorithms*, 77(4):1093–1116.

Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. 2018. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3282–3291, Stockholmsmssan, Stockholm Sweden. PMLR.

Zhengdong Lu and Hang Li. 2013. A deep architecture for matching short texts. In *Advances in Neural Information Processing Systems*, pages 1367–1375.

Thang Luong, Hieu Pham, and D. Christopher Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *WMT*, pages 1–9. Association for Computational Linguistics.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association*

for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), volume 1, pages 2227–2237.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 86–96.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, volume 2, pages 464–468.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, et al. 2018. Tensor2tensor for neural machine translation. *Vol. 1: MT Researchers Track*, page 193.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

Mingxuan Wang, Zhengdong Lu, Jie Zhou, and Qun Liu. 2017. Deep neural machine translation with linear associative unit. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 136–145.

Qiang Wang, Bei Li, Jiqiang Liu, Bojian Jiang, Zheyang Zhang, Yinqiao Li, Ye Lin, Tong Xiao, and Jingbo Zhu. 2018a. The niutrans machine translation system for wmt18. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 528–534.

Qiang Wang, Fuxue Li, Tong Xiao, Yanyang Li, Yinqiao Li, and Jingbo Zhu. 2018b. Multi-layer representation fusion for neural machine translation. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3015–3026.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Tong Xiao, Jingbo Zhu, Hao Zhang, and Qiang Li. 2012. Niutrans: an open source toolkit for phrase-based and syntax-based machine translation. In *Proceedings of the ACL 2012 System Demonstrations*, pages 19–24. Association for Computational Linguistics.

Jie Zhou, Ying Cao, Xuguang Wang, Peng Li, and Wei Xu. 2016. Deep recurrent models with fast-forward connections for neural machine translation. *Transactions of the Association of Computational Linguistics*, 4(1):371–383.

A Derivations of Post-Norm Transformer and Pre-Norm Transformer

A general residual unit can be expressed by:

$$y_l = x_l + \mathcal{F}(x_l; \theta_l), \tag{10}$$

$$x_{l+1} = f(y_l), \tag{11}$$

where x_l and x_{l+1} are the input and output of the l-th sub-layer, and y_l is the intermediate output followed by the post-processing function $f(\cdot)$.

We have known that the post-norm Transformer incorporates layer normalization ($\mathrm{LN}(\cdot)$) by:

$$\begin{aligned} x_{l+1} &= \mathrm{LN}\big(x_l + \mathcal{F}(x_l; \theta_l)\big) \\ &= \mathrm{LN}\big(x_l + \mathcal{F}_{post}(x_l; \theta_l)\big) \end{aligned} \tag{12}$$

where $\mathcal{F}_{post}(\cdot) = \mathcal{F}(\cdot)$. Note that we omit the parameter in LN for clarity. Similarly, the pre-norm Transformer can be described by:

$$\begin{aligned} x_{l+1} &= x_l + \mathcal{F}\big(\mathrm{LN}(x_l); \theta_l\big) \\ &= x_l + \mathcal{F}_{pre}(x_l; \theta_l) \end{aligned} \tag{13}$$

where $\mathcal{F}_{pre}(\cdot) = \mathcal{F}(\mathrm{LN}(\cdot))$. In this way, we can see that both post-norm and pre-norm are special cases of the general residual unit. Specifically, the post-norm Transformer is the special case when:

$$f_{post}(x) = \mathrm{LN}(x), \tag{14}$$

while for pre-norm Transformer, it is:

$$f_{pre}(x) = x. \tag{15}$$

Here we take a stack of L sub-layers as an example. Let $\mathcal{E}$ be the loss used to measure how

many errors occur in system prediction, and x_L be the output of the top-most sub-layer. Then from the chain rule of back propagation we obtain:

$$\frac{\partial \mathcal{E}}{\partial x_l} = \frac{\partial \mathcal{E}}{\partial x_L}\frac{\partial x_L}{\partial x_l} \tag{16}$$

To analyze it, we can directly decompose $\frac{\partial x_L}{\partial x_l}$ layer by layer:

$$\frac{\partial x_L}{\partial x_l} = \frac{\partial x_L}{\partial x_{L-1}}\frac{\partial x_{L-1}}{\partial x_{L-2}}\cdots\frac{\partial x_{l+1}}{\partial x_l}. \tag{17}$$

Consider two adjacent layers as Eq.10 and Eq. 11, we have:

$$\begin{aligned}\frac{\partial x_{l+1}}{\partial x_l} &= \frac{\partial x_{l+1}}{\partial y_l}\frac{\partial y_l}{\partial x_l} \\ &= \frac{\partial f(y_l)}{\partial y_l}\Big(1 + \frac{\partial \mathcal{F}(x_l;\theta_l)}{\partial x_l}\Big)\end{aligned} \tag{18}$$

For post-norm Transformer, it is easy to know $\frac{\partial f_{post}(y_l)}{\partial y_l} = \frac{\partial \mathrm{LN}(y_l)}{\partial y_l}$ according to Eq.(14). Then put Eq.(17) and (18) into Eq.(16) and we can obtain the differential $\mathcal{L}$ w.r.t. x_l:

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial x_l} \;=\; & \frac{\partial \mathcal{E}}{\partial x_L} \times \prod_{k=l}^{L-1}\frac{\partial \mathrm{LN}(y_k)}{\partial y_k} \times \\ & \prod_{k=l}^{L-1}\Big(1 + \frac{\partial \mathcal{F}(x_k;\theta_k)}{\partial x_k}\Big)\end{aligned} \tag{19}$$

Eq.(19) indicates that the number of product terms grows linearly with L, resulting in prone to gradient vanishing or explosion.

However, for pre-norm Transformer, instead of decomposing the gradient layer by layer in Eq. (17), we can use the good nature that $x_L = x_l + \sum_{k=l}^{L-1}\mathcal{F}_{pre}(x_k;\theta_k)$ by recursively using Eq. (13):

$$\begin{aligned}x_L &= x_{L-1} + \mathcal{F}_{pre}(x_{L-1};\theta_{L-1}) \\ &= x_{L-2} + \mathcal{F}_{pre}(x_{L-2};\theta_{L-2}) + \mathcal{F}_{pre}(x_{L-1};\theta_{L-1}) \\ &\cdots \\ &= x_l + \sum_{k=l}^{L-1}\mathcal{F}_{pre}(x_k;\theta_k)\end{aligned} \tag{20}$$

In this way, we can simplify Eq.(17) as:

$$\frac{\partial x_L}{\partial x_l} = 1 + \sum_{k=l}^{L-1}\frac{\partial \mathcal{F}_{pre}(x_k;\theta_k)}{\partial x_l} \tag{21}$$

Due to $\frac{\partial f_{pre}(y_l)}{\partial y_l} = 1$, we can put Eq. (21) into Eq. (16) and obtain:

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial x_l} &= \frac{\partial \mathcal{E}}{\partial x_L} \times \Big(1 + \sum_{k=l}^{L-1}\frac{\partial \mathcal{F}_{pre}(x_k;\theta_k)}{\partial x_l}\Big) \\ &= \frac{\partial \mathcal{E}}{\partial x_L} \times \Big(1 + \sum_{k=l}^{L-1}\frac{\partial \mathcal{F}(\mathrm{LN}(x_k);\theta_k)}{\partial x_l}\Big)\end{aligned} \tag{22}$$

B Training Hyper-parameters for Deep Models

Model	Batch	Upd.	Lr	Wu.	PPL
post	4096	100k	$7e^{-4}$	4k	4.85
post	8192	50k	$2e^{-3}$	16k	*
post-20L	4096	100k	$7e^{-4}$	4k	*
post-20L	8192	50k	$2e^{-3}$	16k	*
pre	4096	100k	$1e^{-3}$	8k	4.88
pre	8192	50k	$2e^{-3}$	16k	4.86
pre-20L	4096	100k	$1e^{-3}$	8k	4.68
pre-20L	8192	50k	$2e^{-3}$	16k	4.60

Table 7: Hyper-parameter selection for shallow and deep models based on perplexity on validation set for English-German translation. "post-20L" is short for post-norm Transformer with a 20-layer encoder. Similarly, "pre-20L" denotes the pre-norm Transformer case. * indicates that the model failed to train.

We select hyper-parameters by measuring perplexity on the validation set of WMT En-De task. We compare the effects of hyper-parameters in both shallow networks (6 layers) and deep networks (20 layers). We use the standard hyper-parameters for both models as the baselines. More concretely, for post-norm Transformer-Base, we set *batch/update/lr/warmup* to 4096/100k/7×10^{-4}/4k as the original Transformer, while for pre-norm Transformer-Base, the configuration is 4096/100k/10^{-3}/8k as suggested in *tensor2tensor*. As for deep models, we uniformly use the setting of 8192/50k/2×10^{-3}/16k. Note that while we use a 2X larger batch size for deep models, we reduce a half of the number of updates. In this way, the amount of seen training data keeps the same in all experiments. A larger learning rate is used to speed up convergence when we use large batch. In addition, we found simultaneously increasing the learning rate and warmup steps worked best.

Table 7 report the results. First of all, we can see that post-norm Transformer failed to train when

the network goes deeper. Worse still, the shallow network also failed to converge when switching to the setting of deep networks. We attribute it to post-norm Transformer being more sensitive to the large learning rate. On the contrary, in the case of either a 6-layer encoder or a 20-layer encoder, the pre-norm Transformer benefits from the larger batch and learning rate. However, the gain under deep networks is larger than that under shallow networks.