Shallow-to-Deep Training for Neural Machine Translation

Bei Li[1], **Ziyang Wang**[1], **Hui Liu**[1], **Yufan Jiang**[1],
Quan Du[1,2], **Tong Xiao**[1,2]*, **Huizhen Wang**[1,2] **and Jingbo Zhu**[1,2]
[1]NLP Lab, School of Computer Science and Engineering
Northeastern University, Shenyang, China
[2]NiuTrans Research, Shenyang, China
{libei_neu,jiangyufan2018,duquanneu}@outlook.com
{wangziyang,huiliu}@stumail.neu.edu.cn
{xiaotong,wanghuizhen,zhujingbo}@mail.neu.edu.cn

Abstract

Deep encoders have been proven to be effective in improving neural machine translation (NMT) systems, but training an extremely deep encoder is time consuming. Moreover, why deep models help NMT is an open question. In this paper, we investigate the behavior of a well-tuned deep Transformer system. We find that stacking layers is helpful in improving the representation ability of NMT models and adjacent layers perform similarly. This inspires us to develop a shallow-to-deep training method that learns deep models by stacking shallow models. In this way, we successfully train a Transformer system with a 54-layer encoder. Experimental results on WMT'16 English-German and WMT'14 English-French translation tasks show that it is 1.4 $\times$ faster than training from scratch, and achieves a BLEU score of 30.33 and 43.29 on two tasks. The code is publicly available at https://github.com/libeineu/SDT-Training.

1 Introduction

In recent years, neural models have led to state-of-the-art results in machine translation (MT) (Bahdanau et al., 2015; Sutskever et al., 2014). Many of these systems can broadly be characterized as following a multi-layer encoder-decoder neural network design: both the encoder and decoder learn representations of word sequences by a stack of layers (Vaswani et al., 2017; Wu et al., 2016; Gehring et al., 2017), building on an interesting line of work in improving such models. The simplest of these increases the model capacity by widening the network, whereas more recent work shows benefits from stacking more layers on the encoder side. For example, for the popular Transformer model (Vaswani et al., 2017), deep systems have shown promising BLEU improvements by either easing the information flow through the network (Bapna et al., 2018) or constraining the gradient norm across layers (Zhang et al., 2019; Xu et al., 2020; Liu et al., 2020). An improved system can even learn a 35-layer encoder, which is $5\times$ deeper than that of vanilla Transformer (Wang et al., 2019).

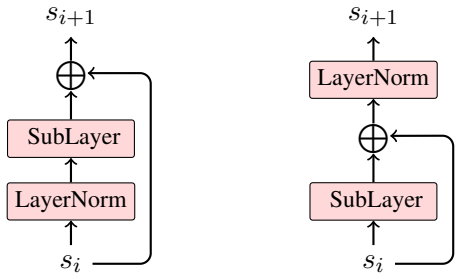
Although these methods have enabled training deep neural MT (NMT) models, questions remain as to the nature of the problem. The main question here is: *why and how deep networks help in NMT*. Note that previous work evaluates these systems in a black-box manner (i.e., BLEU score). It is thus natural to study how much a deep NMT system is able to learn that is different from the shallow counterpart. Beyond this, training an extremely deep model is expensive although a narrow-and-deep network can speed up training (Wang et al., 2019). For example, it takes us $3\times$ longer time to train the model when we deepen the network from 6 layers to 48 layers. This might prevent us from exploiting deeper models in large-scale systems.

In this paper, we explore why deep architectures work to render learning NMT models more effectively. By investigating the change of the hidden states in different layers, we find that new representations are learned by continually stacking layers on top of the base model. More stacked layers lead to a stronger model of representing the sentence. This particularly makes sense in the deep NMT scenario because it has been proven that deep models can benefit from an enriched representation (Wang et al., 2019; Wu et al., 2019b; Wei et al., 2020).

In addition, the finding here inspires us to develop a simple yet efficient method to train a deep NMT encoder: we train model parameters from shallow to deep, rather than training the entire model from scratch. To stabilize training, we design a sparse linear combination method of connecting lower-level layers to the top. It makes efficient pass

*Corresponding author.

Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pages 995–1005,
November 16–20, 2020.



(a) Pre-norm (b) Post-norm

Figure 1: Pre-norm and Post-norm sub-layer architectures.

of information through the deep network but does not require large memory footprint as in dense networks. We experiment with the method in a state-of-the-art deep Transformer system. Our encoder consists of 48-54 layers, which is almost the deepest Transformer model used in NMT. On WMT En-De and En-Fr tasks, it yields a $1.4\times$ speedup of training, matching the state-of-the-art on the WMT'16 En-De task.

2 Background

We start with a description of deep Transformer. In Transformer (Vaswani et al., 2017), the encoder takes a sequence of words $\{x_1, ..., x_n\}$ as input. The input is first transformed into a sequence of embeddings $\{w_1 + p_1, ..., w_n + p_n\}$, where w_k is a word embedding and p_k is a positional embedding. Then, the embedding sequence is fed into a stack of N identical layers. Each layer consists of two stacked sub-layers: a multi-head self-attention sub-layer and a feed-forward sub-layer. The decoder shares a similar architecture as the encoder but possesses an encoder-decoder attention sub-layer to capture the mapping between two languages.

For a deep model, layer normalization networks and layer-wise connections are needed, following the previous work of Bapna et al. (2018) and Wang et al. (2019).

- **Pre-Norm Residual Networks**. We make a residual connection (He et al., 2016) and a layer normalization unit (Lei Ba et al., 2016) at the input of each sub-layer. The output of the sub-layer is defined to be:

$$s_{i+1} = s_i + \text{SubLayer}(\text{LayerNorm}(s_i))$$

where s_i and s_{i+1} are the output of sub-layers i and $i+1$. See Figure 1 (a) for the architecture of a pre-norm sub-layer. Pre-norm residual network has been found to be more efficient for back-propagation over a large number of layers than the post-norm architecture (Wang et al., 2019; Li et al., 2019).

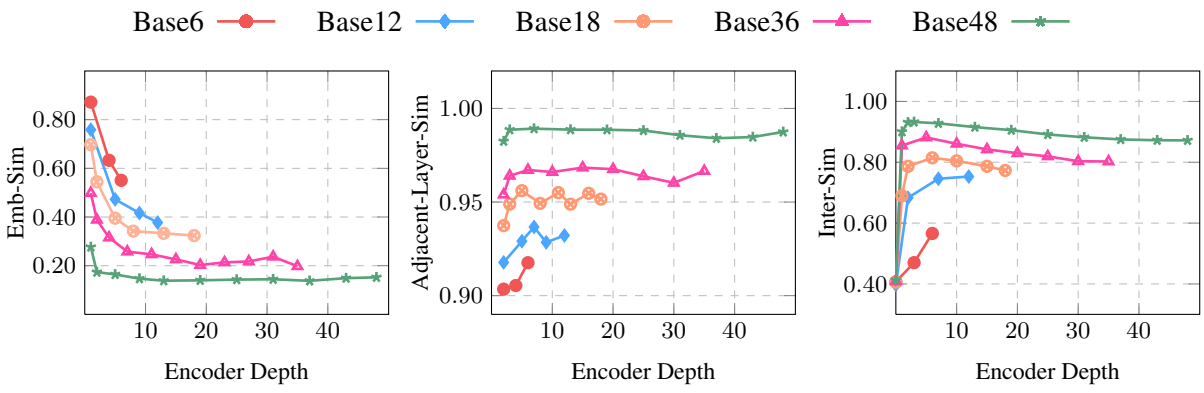
- **Dense Connections**. Direct layer connections can make easy access to distant layers in the stack (Wang et al., 2018; Bapna et al., 2018). Let $\{y_1, ..., y_N\}$ be the output of the stacked layers. We define a network $G(y_1, ..., y_{j-1})$ that reads all layer output vectors prior to layer j and generates a new vector. Then, $G(y_1, ..., y_{j-1})$ is regarded as a part of the input of layer j. In this way, we create direct connections from layers $\{1, ..., j-1\}$ to layer j. For $G(\cdot)$, we choose a linear model as in (Wang et al., 2019).

3 Why do Deep Models Help?

The Transformer encoder (or decoder) is essentially a representation model (Vaswani et al., 2017). Given a word sequence, a layer generates a distributed representation (i.e., y_j) for each position of the sequence. The representation is a mixture of word+position embedding and context embedding. For a simple implementation, only the top-most representation (i.e., y_N) is used for downstream components of the system. Nevertheless, the dense connections can make the lower-level representations directly accessible to the top layers. Hence, the representation model is actually encoded by the set of layer outputs $\{y_1, ..., y_N\}$.

For a stronger model, enlarging the size of each layer can fit the objective function with enough capacity. For example, Transformer-Big doubles the layer size of the base model and shows consistent improvements on several MT tasks. But the number of parameters increases quadratically with network width, which poses new difficulties in training such systems and the risk of overfitting. Alternatively, one can stack layers to strengthen the model because more layers offer more representations for the input sentence. Moreover, top-level layers can generate refined representations (Greff et al., 2017) by passing the input vectors through more linear and non-linear transformations in different layers.

To study how each layer behaves, we evaluate the change of the representation vector in the stack.



(a) Emb-Similarity (b) Adjacent-Layer-Similarity (c) Inter-Similarity

Figure 2: (a) Similarity of layer i and the input embedding, (b) Similarity of layer i and layer $i-1$, and (c) Inter-Similarity over the validation sequences. Note that 0 represents the embedding layer.

To do this, we compute the similarity between the outputs of two layers, as below,

$$\text{sim}(i,j) = \frac{1}{n}\sum_{l=1}^{n} \text{cosine}(y_i(l), y_j(l))$$

where $y_i(l)$ (or $y_j(l)$) is the output of layer i (or j) for position l of the sequence. $\text{sim}(i,j)$ measures the degree of how close the representation vector of layer i is to that of layer j.

Figure 2(a) plots $\text{sim}(i,0)$ curves for WMT En-De systems of different encoder depths. Here $\text{sim}(i,0)$ measures how similar the output of layer j is to the input embedding of the encoder. We see that the similarity keeps going down when we stack more layers. It indicates that the model can learn new representations by using more stacked layers. But the similarity does not converge for shallow models (e.g., 6-layer and 12-layer encoders). It somehow reflects the fact that the shallow models "want" more layers to learn something new. More interestingly, deeper models (e.g., encoders of 18 layers or more) make the similarity converge as the depth increases, showing the effect of rendering the need of representation learning fulfilled.

Also, we investigate the similarity between adjacent layers for different systems. Figure 2(b) plots $\text{sim}(i,i-1)$ as a function of layer number, which begins with $\text{sim}(2,1)$ rather than $\text{sim}(1,0)$[1] for better visualization. The results show that adjacent layers in converged systems have a high similarity. This agrees with previous work on the similarity of attention weights among layers (Xiao et al., 2019) though we study a different issue here. The deeper the models, the higher the similarity between adjacent layers. A natural question is whether we can initialize the higher layers by reusing the parameters of previous layers during training procedure?

Note that the Transformer model is doing something like encoding both contextual information and word information for each position of the sequence. Here, we design the Inter-Sim over the sequence to see how much the representation of a position holds to encode the sequence. For layer i, we have

$$\begin{aligned} \text{sim}_{\text{in}}(i) &= \frac{1}{n}\sum_{l=1}^{n} \text{cosine}(y_i(l), \bar{y}_i) \\ \bar{y}_i &= \frac{1}{n}\sum_{l=1}^{n} y_i(l) \end{aligned}$$

where $\bar{y}_i$ is the mean vector of sequence $\{y_i(1), ..., y_i(n)\}$ and can be seen as the global representation of the entire sequence. $\text{sim}_{\text{in}}(i)$ is an indicator of the distance between an individual representation and the global representation. Figure 2(c) shows that the representation of a position tends to be close to the global representation for higher-level layers. This can be seen as smoothing the representations over different positions. A smoothed representation makes the model more robust and is less sensitive to noisy input. This result constitutes evidence that deep models share more global information over different positions of the encoder. Hence, it is easier to access the global representation of the source sequence for decoder and to generate the translation using a global context.

[1]Note that $(\text{sim}(1,0))$ of Figure 2(b) are the same with those in Figure 2(a).

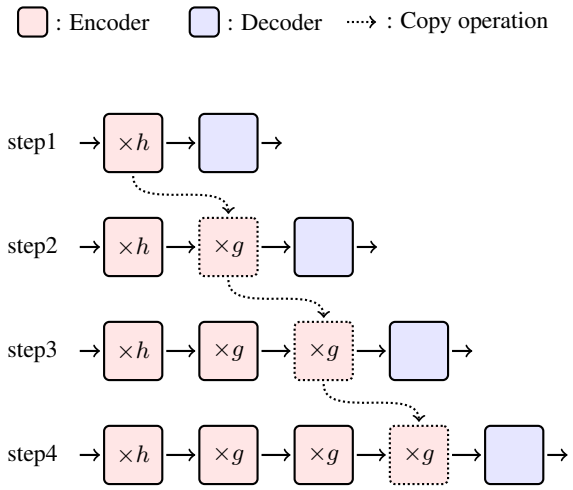


Figure 3: Shallow-to-deep training process.

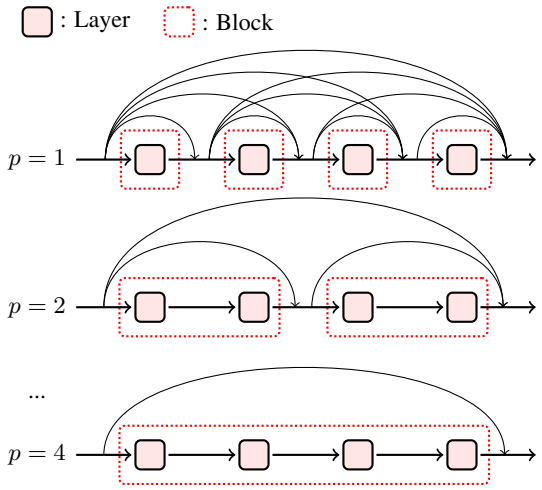


Figure 4: Sparse connections between layers.

4 Shallow-to-Deep Training

Although we are able to train deep models with the standard methods (Wang et al., 2019), it is obviously a time consuming task to learn a model with too many layers. For example, training a 48-layer system takes us 3× longer time than the 6-layer baseline. As stated in Section 3, adjacent layers in a deep network are likely to behave in a similar fashion. This observation inspires us to train upper-level layers by reusing the learned parameters of lower-level layers. We call this method shallow-to-deep training (SDT) because we start with training a shallow model and then train a deeper model on top of it.

4.1 The Method

Assume that we have an initial model A with h layers that have already been trained. Now we need to train a new model B with $h + g$ layers ($h \geq g$). Unlike previous work, we do not train all $h + g$ layers from scratch. Instead,

- We copy the parameters of the first h layers from A to B.
- We then copy the parameters of the g top-most layers from A to B.

Model A can be seen as a good starting point of model B. After initializing of the model, we continue training model B as usual. The training can converge faster because the model initialization tends to place the parameters in regions of the parameter space that generalize well. See Figure 3 for an illustration of the method. In this work, we use the same step to learn from shallow to deep. For example, we train a 6-layer model, and then a 12-layer model, and then a 18-layer model, and so on.

4.2 Sparse Connections between Layers

The efficient pass of information plays an important role in training deep models. To this end, one can create direct connections between layers by dense networks. They are found to be necessary to learn strong Transformer systems (Bapna et al., 2018; Wang et al., 2019; Wu et al., 2019b). However, an extremely deep model in general results in a large number of such connections and of course a heavy system. For example, a 48-layer system runs 1.87× slower than the system with no use of dense connections. We cannot even train it using a batch of 2048 tokens on TITAN V GPUs due to large memory footprint. Instead, we develop a method that resembles the merits of layer-wise connections but is lighter. The idea is pretty simple: we group every p layers to form a layer block and make connections between layer blocks. The connections between blocks are created in the standard way as used in dense networks (see Section 2). Here p is a parameter to control the connection density. For example, $p = 1$ means dense networks, and $p = \infty$ means networks with no layer-wise connections [2]. See Figure 4 for example networks with block/layer-wise connections.

4.3 Learning Rate Restart

The design of the learning rate schema is one of the keys to the success of Transformer. For example, Vaswani et al. (2017) designed a method to warm

[2] Residual connections are used by default in this work.

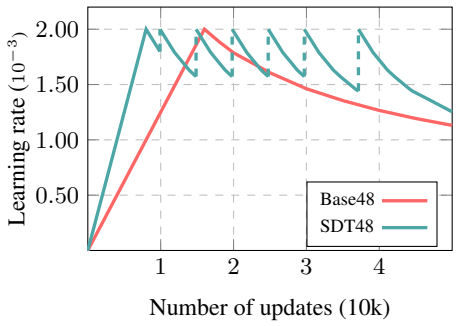


Figure 5: The learning rate schedule of each stacking.

up training for a number of training steps and then decrease the learning rate. In this work, shallow-to-deep training breaks the training process because we need to switch to a deeper model with initialization at some training steps. We found in our experiments that deep models could not be trained efficiently in the standard way because we had a small learning rate for a newly stacked model in late training steps.

We develop a new method to ensure that the model can be trained using a proper learning rate at every point of switching to a deeper model. We divide the training into a number of stages. Each of them is associated with a deeper model.

- For the first ω stages, the model is trained with a linear-warmup learning rate (lr) as described in (Vaswani et al., 2017).

$$lr = d_{\text{model}}^{-0.5} \cdot step\_num \cdot warmup\_steps^{-1.5}$$

- For each of the following stages, the learning rate of a newly stacked model declines from the max-point with an inverse squared root of the current step.

$$lr = d_{\text{model}}^{-0.5} \cdot step\_num^{-0.5}$$

  At the beginning of the stage, we reset the number of training steps.

Here $step\_num$ and $warmup\_steps$ are the current training step number and the warmup-step number. d_{model} is the size of the layer output. See Figure 5 for a comparison of different learning schemas.

5 Experiments

We report the experimental results on two widely used benchmarks - WMT'16 English-German (En-De) and WMT'14 English-French (En-Fr).

5.1 Data

For the En-De task, we used the same preprocessed data with (Vaswani et al., 2017; Ott et al., 2019; Wang et al., 2019), consisting of approximate 4.5M tokenized sentence pairs. All sentences were segmented into sequences of sub-word units (Sennrich et al., 2016) with 32K merge operations using a vocabulary shared by source and target sides. We selected *newstest2012+newstest2013* as validation data and *newstest2014* as test data.

For the En-Fr task, we replicated the setup of Vaswani et al. (2017) with 36M training sentence pairs from WMT14. We validated the En-Fr system on the union set of *newstest2012* and *newstest2013*, and tested it on *newstest2014*. We filtered out sentences of more than 200 words and generated a shared vocabulary with 40K merge operations on both source and target side.

We re-merged sub-word units to form complete words in the final output. For comparable results with previous work (Wu et al., 2016; Gehring et al., 2017; Vaswani et al., 2017), we also adopted compound split for En→De. We reported case-sensitive tokenized BLEU scores for both En-De and En-Fr tasks, and sacrebleu[3] scores for both En-De and En-Fr tasks. The results were the mean of three times run with different random seeds.

5.2 Model Settings

Our implementation was based on Fairseq (Ott et al., 2019). For training, we used Adam optimizer (Kingma and Ba, 2015) with $\beta_1 = 0.9$, $\beta_2 = 0.997$, and $\epsilon = 10^{-8}$. We adopted the same learning rate schedule as the latest implementation of Tensor2Tensor[4]. For deep models, the learning rate (lr) first increased linearly for $warmup = 8,000$ steps from $1e^{-7}$ to $2e^{-3}$. After warmup, the learning rate decayed proportionally to the inverse square root of the current step. For our SDT method presented in Section 4, we set $h = g = p = 6$ on both the WMT En-De and En-Fr tasks. For a stronger system, we employed relative position representation (RPR) to strengthen the position embedding model (Shaw et al., 2018). We only used the relative key in each layer.

We batched sentence pairs by approximate length, and limited input/output tokens per batch to $4,096$/GPU. Following the method of (Wang et al.,

[3]BLEU+case.mixed+numrefs.1+smooth.exp+tok.13a +version.1.2.12

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

Systems	WMT En-De					WMT En-Fr				
	Params	Time	Speedup	BLEU	Sacrebleu	Params	Time	Speedup	BLEU	Sacrebleu
Vaswani et al. (2017) (Big)	213M	N/A	N/A	28.40	N/A	222M	N/A	N/A	41.00	N/A
Shaw et al. (2018) (Big)	210M	N/A	N/A	29.20	N/A	222M	N/A	N/A	41.30	N/A
Ott et al. (2018) (Big)	210M	N/A	N/A	29.30	28.6	222M	N/A	N/A	43.20	41.4
Wu et al. (2019b) (Big)	270M	N/A	N/A	29.92	N/A	281M	N/A	N/A	43.27	N/A
Wang et al. (2019) (Deep)	137M	N/A	N/A	29.30	N/A	N/A	N/A	N/A	N/A	N/A
Wei et al. (2020) (Deep)	272M	N/A	N/A	30.19	N/A	N/A	N/A	N/A	N/A	N/A
Wei et al. (2020) (Big+Deep)	512M	N/A	N/A	30.56	N/A	N/A	N/A	N/A	N/A	N/A
Base (Pre-Norm)	63M	4.79	N/A	27.05	26.0	67M	27.11	N/A	41.00	39.2
Big (Pre-Norm)	210M	36.05	N/A	28.79	27.7	222M	97.51	N/A	42.40	40.6
Deep-24L	118M	8.66	0	28.95	27.8	124M	48.43	0	42.40	40.6
SDT-24L	118M	6.16	28.92%	29.02	27.9	124M	33.81	30.10%	42.42	40.6
Deep-RPR-24L	118M	9.80	0	29.39	28.3	124M	55.32	0	42.67	40.9
SDT-RPR-24L	118M	6.71	31.53%	29.39	28.3	124M	37.59	32.05%	42.69	40.9
Deep-48L	194M	16.38	0	29.44	28.3	199M	90.85	0	42.75	41.0
SDT-48L	194M	10.65	35.02%	29.60	28.5	199M	55.35	39.08%	42.82	41.0
Deep-RPR-48L	194M	19.58	0	30.03	28.8	199M	116.92	0	43.08	41.3
SDT-RPR-48L	194M	11.75	**39.98%**	**30.21**	**29.0**	199M	64.46	**44.90%**	**43.29**	**41.5**
Deep-24L (Big)	437M	37.41	0	29.90	28.7	N/A	N/A	N/A	N/A	N/A
SDT-24L (Big)	437M	18.31	47.41%	29.93	28.7	N/A	N/A	N/A	N/A	N/A
Deep-RPR-24L (Big)	437M	38.80	0	30.40	29.2	N/A	N/A	N/A	N/A	N/A
SDT-RPR-24L (Big)	437M	18.51	**52.30%**	**30.46**	**29.3**	N/A	N/A	N/A	N/A	N/A

Table 1: Results of deep models on WMT14 En-De and WMT14 En-Fr tasks by the model parameters [million], training costs [hours], acceleration rates [%], BLEU scores [%], △ BLEU [%] and Sacrebleu scores [%].

2019), we accumulated every two steps for a better batching. This resulted in approximately $56,000$ source and $56,000$ target tokens per training batch. The deep models were updated for 50k steps on the En-De task and 150k steps on the En-Fr task. All models were trained on 8 NVIDIA TITAN V GPUs with mix-precision accelerating. For fair comparison, we trained the deep Pre-Norm Transformer with the same settings reported in Wang et al. (2019). And all results are the average of three times running with different random seeds. We chose different hyper-parameter settings for the models.

- Base/Deep Model. The hidden layer size of self-attention was 512, and the size of feed forward inner-layer was $2,048$. Also, we used 8 heads for attention. For training, we set all dropout to 0.1, including residual dropout, attention dropout, relu dropout. Label smoothing $\epsilon_{ls} = 0.1$ was applied to enhance the generation ability of the model.

- Big Model. We used the same architecture as Transformer-Base but with a larger hidden layer size $1,024$, more attention heads (16), and a larger feed forward inner-layer $(4,096$ dimensions). The residual dropout was set to 0.3 for the En-De task and 0.1 for the En-Fr task. Additionally, the same depth (6) on both encoder and decoder side with Base model. For deeper Big model, we only change the encoder depth.

For evaluation, we averaged the last 5 consecutive checkpoints which were saved per training epoch on all WMT models. For all datasets, the length penalty was set to 0.6 and the beam size was set to 4.

5.3 Results

Table 1 summarizes the training cost and the translation quality on the WMT En-De and En-Fr tasks. First, we compare deep Transformer systems (Pre-Norm) with previously reported systems. Deep Transformer brings substantial improvements than big counterparts within the same experiment settings[5]. In addition, our SDT method enables efficient training for deeper networks with no loss in BLEU. As we can see from Table 1, the systems trained with the SDT method achieve comparable or even higher BLEU scores with their baselines, and the training costs are much less.

Another finding is that systems encoding relative position representation in the same encoder depth outperform their baselines by $0.44 - 0.59$ BLEU points on the En-De task, indicating that relative

[5]The models of Ott et al. (2018) were trained on 128 GPUS. And Wu et al. (2019b) trained their networks for $800,000$ steps.

Reset-lr	Copy-Initialization	BLEU
×	×	27.33
✓	×	27.98
×	✓	29.93
✓	✓	**30.21**

Table 2: Effect of learning rate schema and copy-initialization strategy.

Copy-Initialization	Interval	BLEU
Top only	$g = 6$	29.20
Interpolation	$g = 6$	29.46
g Top-most	$g = 6$	**30.21**

Table 3: The comparison of our method with other copy initialization strategies.

Strategy	Interv.	Speedup	BLEU
$g = 3$	1	36.7%	29.38
$g = 6$	2	39.9%	**30.21**
$g = 9$	4	42.1%	29.47
$g = 6, 9, 12, 15$	4	**51.8%**	29.78

Table 4: BLEU scores [%] vs. speedup of different stacking strategies during training.

System	Speedup	BLEU	Sacrebleu
Deep-RPR-30L	N/A	29.52	28.4
DLCL-RPR-30L	ref	30.01	28.9
Sparse-RPR-30L	8.9%	29.90	28.8
SDT-RPR-30L	**42.1%**	29.95	28.9
SDT-RPR-54L	N/A	**30.33**	**29.2**
SDT-RPR-60L	N/A	30.28	29.1

Table 5: Comparison of DLCL and our work.

position representation can further strengthen deep Transformer. Similarly, it is observed that SDT can also speed up the RPR enhanced systems with no loss of translation quality. The speedup is larger for deeper models that the SDT method speeds up the training of 48-layer systems by $35.02\% - 39.98\%$. Surprisingly, both SDT-48L and SDT-RPR-48L achieve modest BLEU improvements compared with the baseline. This indicates that the benefit from enlarging encoder depth gradually decreases and our method alleviates the overfitting problem when the model is extremely deep.

To further validate the effectiveness of SDT method, we experimented on 24-layer Big models. Through Table 1 we see that, it achieves up to 52.30% speedup and match the performance with learning from scratch. Note that the training time of Deep-24L (Big) and Deep-RPR-24L (Big) are 37.41 and 38.80 hours respectively because the models were only optimized by 50k steps and they converged on the validation set. The finding here is similar with Wang et al. (2019)'s work that deep encoders can speed up the training process with a large learning rate. In addition, the BLEU score of SDT-RPR-24L (Big) is 30.46, which matches with the state-of-the-art within less parameters and training cost. Another finding here is the speedup of SDT training may gets larger when the model architecture gets more complex.

The similar phenomenon is observed in the WMT En-Fr task, a much larger dataset than that of the En-De task. The results in Table 1 show the effectiveness of our SDT method. It accelerates the training procedure by 44.90% with nearly 0.2 BLEU improvement on a 48-layer RPR system. The larger the dataset is, the greater the speedup will be. Note that our 48-layer deep system matches the state-of-the-art reported in (Ott et al., 2018; Wu et al., 2019a,b) within 8 GPUs training. We will furthermore verify whether there is an another improvement when we switch to a much larger batching schema. However, due to the large size of En-Fr dataset, optimizing a 24-layer Big model is quite time consuming, thus we have not finished the training yet. The experimental results can be found in our codebase soon. In addition, another benifit brought by SDT is that we can efficiently build the ensemble system given an already optimized model, we will show more details in our codebase.

6 Analysis

6.1 Ablation Study

In Table 2, we summarize the effect of resetting the learning rate and copy-initialization strategy in our SDT method. We choose SDT-RPR-48L as our baseline due to its strong performance. We see, first of all, that the copy-initialization plays an important role in our shallow-to-deep training process. For example, the BLEU score decreases dramatically if we stack the model from 6 layers to 12 layers and initialize the new block by a uniform distribution. This can be explained by the fact that every top-most layers are not sufficiently converged in stacking. Another observation is that the reset learning rate can also facilitate the training, enabling the model to learn fast and bringing nearly a +0.3 BLEU improvement. Moreover, we

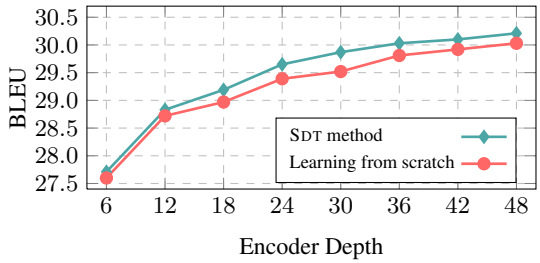


Figure 6: The comparison between learning from scratch and SDT against different encoder depth.

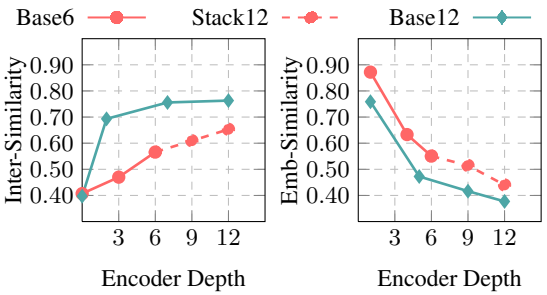


Figure 7: Inter and Emb Similarity of Stack-12L.

compare different copy-initialization strategies, including initializing the new group by copying the g top-most layers, copying the top-most layer for g times and inserting the layer right after each existing layer in the group. From Table 3, we observe that our copying g top-most strategy achieves best performance.

Also, we investigate the impact of different stacking strategies on translation quality and speedup. Table 4 shows results of the models trained with different settings of g and training intervals[6]. Row 4 denotes the case that we stack the shallow model in an incremental way. We find that the stacking strategy and its training interval make great impacts on both translation quality and speedup. We need a trade-off to select the "best" system in different situations. For example, our default strategy (line 2) obtains the best performance, and the incremental stacking achieves the biggest speedup.

6.2 Comparison with Previous Work

Next, we compare the system Transformer-DLCL (Wang et al., 2019) with our SDT system. Table 5 shows the BLEU scores of the models trained with DLCL, sparse connection and SDT based on RPR, respectively. We see that Sparse-RPR-30L can achieve comparable performance with DLCL-RPR-30L using much fewer connections across encoder layers. More interestingly, our SDT-RPR-30L has a comparable BLEU score with DLCL-RPR-30L, but is 42.1% faster. In addition, we find that SDT-RPR-54L outperforms SDT-RPR-30L by 0.38 BLEU scores, but much deeper models cannot gain more benefits. This result indicates that deeper representation models might suffer from the overfitting problem.

Another benefit brought by SDT method is that we can train a deep Transformer model from a pre-trained model instead of training from scratch. For example, we can begin training from a pre-trained 24-layer system to progressively obtain a 48-layer system. The experimental results in Figure 6 verify our conjecture. Except the advantage of accelerating the training, the models trained through the SDT method can even slightly outperform those training from scratch at almost all encoder depths. This enables us to quickly obtain single systems in different depth from an already trained system, which is efficient to build ensemble systems, especially when the training data is extremely large.

[6]Training interval means the training epoch of each newly stacking model

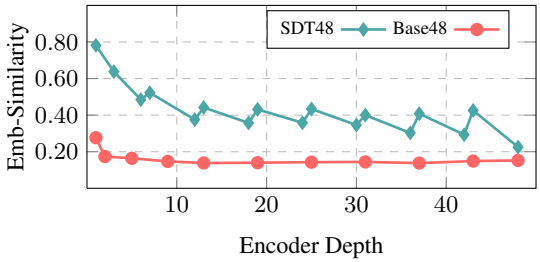


Figure 8: The comparison of Emb Similarity between Base48 and SDT-48L systems.

System	BLEU	△BLEU
Pre-Norm-6L	27.05	0
Pre-Norm-12L	28.33	⇑ 1.28
Stack-12L	28.04	⇑ 0.99
Reg-6L	27.45	⇑ 0.40

Table 6: BLEU scores [%] of several systems.

6.3 Similarity of Layers

We show that the inter-similarity and emb-similarity of shallow models fail to converge in Section 3. Here, we further study the model behavior for different systems. We fix the parameters of a well-trained Pre-Norm-6L baseline and stack it into a 12-layer system by copying the top-6 layers. The dash lines in Figure 7 denote the new stacked system Stack-12L. We observe that

both inter-similarity and emb-similarity continue to rise and decline respectively, which exhibit similar phenomenon with Pre-Norm-12L which is trained from scratch. And it also outperforms the baseline by 0.99 BLEU points and is slightly inferior to Pre-Norm-12L (Table 6). Motivated by this, we design a regularization (according to Inter-sim) to constrain each layer to learn more global information on Pre-Norm-6L. Experimental results show that Reg-6L outperforms the baseline by 0.4 BLEU scores, indicating that a stronger global representation substantially improves the NMT model. This confirms our hypothesis in Section 3.

Figure 8 plots the emb-similarity of Base48 and SDT-48L. The model trained from shallow to deep behaves similarly with learning from scratch. The emb-similarity shows the same trend of decreasing in terms of similarity, and tends to coverage after layer 48. The results also indicate that our method can enbale the deep models to learn efficiently.

7 Related Work

In this section, we discuss the related work from two aspects as follows:

7.1 Deep Network Modeling

In recent years, researchers gradually concentrate on building deep networks for Transformer (Vaswani et al., 2017). Pham et al. (2019) developed a 48-layer Transformer for speech recognition and adopted the stochastic residual connection to alleviate gradient vanishing/exploding problem. Bapna et al. (2018) demonstrated the challenge when training deep encoder models with vanilla Transformer on NMT task, due to the gradient vanishing or exploding. They also proposed a transparent attention mechanism to alleviate the problem. Wang et al. (2019) demonstrated the essential of layer-normalization in each layer and proposed the dynamic linear combination method to ease the information flow. Homochronously, (Wu et al., 2019b) trained a 8-layer Transformer-Big with three specially designed components. More recently, Wei et al. (2020) further enhanced the Transformer-Big up to 18 layers through a multi-scale collaborative framework. In general, shortening the path from bottom to top can obtain consistent improvements in the aforementioned studies. On the other hand, researchers observed that proper initialization strategies without any structure adjustment can also ease the optimization of Post-Norm Transformer, which highlighted the importance of careful parameter-initialization (Zhang et al., 2019; Xu et al., 2020; Huang et al., 2020).

7.2 Efficient Training Methods

When the model goes deeper, a challenge is the long training time for model convergence and the huge GPU cost. To alleviate this issue, several attempts have been made. Chang et al. (2018) proposed a multi-level training method by interpolating a residual block right after each existing block to accelerate the training of ResNets in computer version. Similarly, Gong et al. (2019) adopted a progressive stacking strategy to transfer the knowledge from a shallow model to a deep model, thus successfully trained a large-scale pre-training model BERT (Devlin et al., 2019) at a faster rate with comparable performance on downstream tasks. Unlike previous work, we only copy parameters of the g top-most layers and employ sparse connections across each stacking block in our shallow to deep training method, which has not been discussed yet in learning deep MT models.

8 Conclusions

We have investigated the behaviour of the well-trained deep Transformer models and found that stacking more layers could improve the representation ability of NMT systems. Higher layers share more global information over different positions and adjacent layers behave similarly. Also, we have developed a shallow-to-deep training strategy and employ sparse connections across blocks to ease the optimization. With the help of learning rate restart and appropriate initialization we successfully train a 48-layer RPR model by progressive stacking and achieve a 40% speedup on both WMT'16 English-German and WMT'14 English-French tasks. Furthermore, our SDT-RPR-24L (Big) achieves a BLEU score of 30.46 on WMT'16 English-German task, and speeds up the training by $1.5\times$.

Acknowledgments

This work was supported in part by the National Science Foundation of China (Nos. 61876035 and 61732005), the National Key R&D Program of China (No. 2019QY1801). The authors would like to thank anonymous reviewers for their valuable comments. And thank Qiang Wang for the helpful advice to improve the paper.

References

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, Brussels, Belgium. Association for Computational Linguistics.

Bo Chang, Lili Meng, Eldad Haber, Frederick Tung, and David Begert. 2018. Multi-level residual networks from dynamical systems view. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. 2017. Convolutional sequence to sequence learning. In *ICML*.

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2019. Efficient training of BERT by progressively stacking. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 2337–2346.

Klaus Greff, Rupesh Kumar Srivastava, and Jürgen Schmidhuber. 2017. Highway and residual networks learn unrolled iterative estimation. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

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

Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. 2020. Improving transformer optimization through better initialization. In *Proceedings of Machine Learning and Systems 2020*, pages 9868–9876.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

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

Bei Li, Yinqiao Li, Chen Xu, Ye Lin, Jiqiang Liu, Hui Liu, Ziyang Wang, Yuhao Zhang, Nuo Xu, Zeyang Wang, Kai Feng, Hexuan Chen, Tengbo Liu, Yanyang Li, Qiang Wang, Tong Xiao, and Jingbo Zhu. 2019. The NiuTrans machine translation systems for WMT19. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 257–266, Florence, Italy. Association for Computational Linguistics.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. 2020. Understanding the difficulty of training transformers. *arXiv preprint arXiv:2004.08249*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation, Volume 1: Research Papers*, pages 1–9, Belgium, Brussels. Association for Computational Linguistics.

Ngoc-Quan Pham, Thai-Son Nguyen, Jan Niehues, Markus Müller, and Alex Waibel. 2019. Very deep self-attention networks for end-to-end speech recognition. In *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*, pages 66–70. ISCA.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

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)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

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, 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.

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 *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1810–1822, Florence, Italy. Association for Computational Linguistics.

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

Xiangpeng Wei, Heng Yu, Yue Hu, Yue Zhang, Rongxiang Weng, and Weihua Luo. 2020. Multiscale collaborative deep models for neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 414–426, Online. Association for Computational Linguistics.

Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. 2019a. Pay less attention with lightweight and dynamic convolutions. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Lijun Wu, Yiren Wang, Yingce Xia, Fei Tian, Fei Gao, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2019b. Depth growing for neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5558–5563, Florence, Italy. Association for Computational Linguistics.

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, Yinqiao Li, Jingbo Zhu, Zhengtao Yu, and Tongran Liu. 2019. Sharing attention weights for fast transformer. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 5292–5298. ijcai.org.

Hongfei Xu, Qiuhui Liu, Josef van Genabith, Deyi Xiong, and Jingyi Zhang. 2020. Lipschitz constrained parameter initialization for deep transformers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 397–402, Online. Association for Computational Linguistics.

Biao Zhang, Ivan Titov, and Rico Sennrich. 2019. Improving deep transformer with depth-scaled initialization and merged attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 897–908, Hong Kong, China. Association for Computational Linguistics.