Bag of Tricks for Optimizing Transformer Efficiency

Ye Lin[1]*, Yanyang Li[2]*, Tong Xiao[1,3]†, Jingbo Zhu[1,3]
[1]NLP Lab, School of Computer Science and Engineering,
Northeastern University, Shenyang, China
[2]The Chinese University of Hong Kong, Hong Kong, China
[3]NiuTrans Research, Shenyang, China
{linye2015,blamedrlee}@outlook.com
{xiaotong,zhujingbo}@mail.neu.edu.cn

Abstract

Improving Transformer efficiency has become increasingly attractive recently. A wide range of methods has been proposed, e.g., pruning, quantization, new architectures and etc. But these methods are either sophisticated in implementation or dependent on hardware. In this paper, we show that the efficiency of Transformer can be improved by combining some simple and hardware-agnostic methods, including tuning hyper-parameters, better design choices and training strategies. On the WMT news translation tasks, we improve the inference efficiency of a strong Transformer system by 3.80× on CPU and 2.52× on GPU. The code is publicly available at https://github.com/Lollipop321/mini-decoder-network.

1 Introduction

Standard implementation of Transformer (Vaswani et al., 2017) is not efficient for inference. Researchers have explored more efficient architectures (Zhang et al., 2018; Xiao et al., 2019; Li et al., 2021) or break the auto-regressive constraint in sequence generation (Gu et al., 2017). But most of these require significant updates of the model or hardware-dependent designs. It is still natural to ask whether the Transformer system can be optimized in a simple way (Hsu et al., 2020; Kasai et al., 2020; Kim et al., 2019; Wang and Tu, 2020).

In this paper we show that Transformer can be optimized for efficiency by a bag of techniques. These techniques are easy to implement and some of them have been tested in related studies. Here we focus on using them in combination for Transformer speedup which has not been well investigated. In particular, our work is based on the following facts:

Module		Time (s)	
		Baseline	MDN
Encoder	Attention	5.81	16.93
	FFN	5.53	18.79
Decoder	Attention	223.55	8.74
	FFN	38.26	0.00
	Output	48.51	8.61

Table 1: Profiling results of the Transformer baseline and our model on WMT14 En-De (FFN: the feedforward network, Output: the output projection).

- The default Byte-Pair Encoding (BPE) setting (Sennrich et al., 2016) has a great impact on efficiency but is generally not optimal.
- A shallow decoder (with a deeper encoder) is preferred for a fast system (Kasai et al., 2020).
- The attention model does not need to be multi-headed in some cases (Behnke and Heafield, 2020).
- The feedforward network sub-layer is removable (Hsu et al., 2020).
- Knowledge Distillation (Hinton et al., 2015) is crucial to squeeze out the last potential. Removing some regularization measures like label smoothing (Szegedy et al., 2016) also helps when training such models.

All these methods are compatible with popular Transformer codebases. In this work, we implement them on the decoder side because it occupies the inference time in many sequence generation tasks (Hsu et al., 2020; Kim et al., 2019). The end result is a simplified and fast Transformer decoder (see Table 1) - *Mini-Decoder Network* (MDN). Experiments on the WMT14 En-De, WMT14 En-Fr and NIST12 Zh-En machine translation (MT) benchmarks demonstrate that the improved system achieves a 3.80× speedup on CPU and a 2.52×

*Authors contributed equally.
†Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2021, pages 4227–4233
November 7–11, 2021.

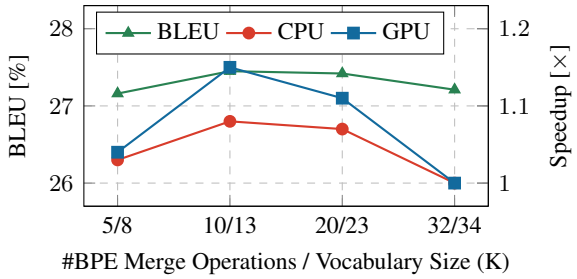

Figure 1: The number of BPE merge operations vs. BLEU and speedup on WMT14 En-De (Detailed setup could be found in Section 3.1).

speedup on GPU with performance on par with the baseline. The speedup obtained is available on most modern hardware, as it does not depend on specific hardware or library, e.g., quantization (Chung et al., 2020) and unstructured pruning (Hoefler et al., 2021) require the support of the latest hardware-dependent and acceleration libraries.

2 Methods

2.1 Byte-Pair Encoding

Byte-Pair Encoding (BPE) (Sennrich et al., 2016) breaks words into subword units. It starts from the alphabet and merges characters into the most frequent subword units, then segments words in sentences by these merged subword units. BPE reduces the risk of out-of-vocabulary words with a small vocabulary, but comes with the cost of longer sentences. If more merge operations are employed, the resulting sentences will be shorter, yet the vocabulary is larger as more subword units are presented. This leads to a tradeoff between sentence length and vocabulary size, and both of them have an impact on the efficiency. In Fig. 1, the green line denotes the BLEU referencing the left axis. The red line denotes the speed change on CPU, the blue line denotes the speed change on GPU, they are both referencing the right axis. As shown in Fig. 1, tuning this hyper-parameter provides a considerable speedup without loss in performance, though most previous work simply adopts the default setting (32K). In our experiments we choose 10K because it is sufficient for good performance.

2.2 Model Structure Updates

Inspired by the observations in Table 1, the Transformer decoder can be improved for each of its components. In this section, we describe how to simplify Transformer in a systematic way as shown in Fig. 2. Table 2 summarizes the contributions of each adopted method. We choose Baseline1 in Table 2 to analyze how each method in "Model Structure Updates" influences the model performance and inference speed after applying techniques from "Byte-Pair Encoding".

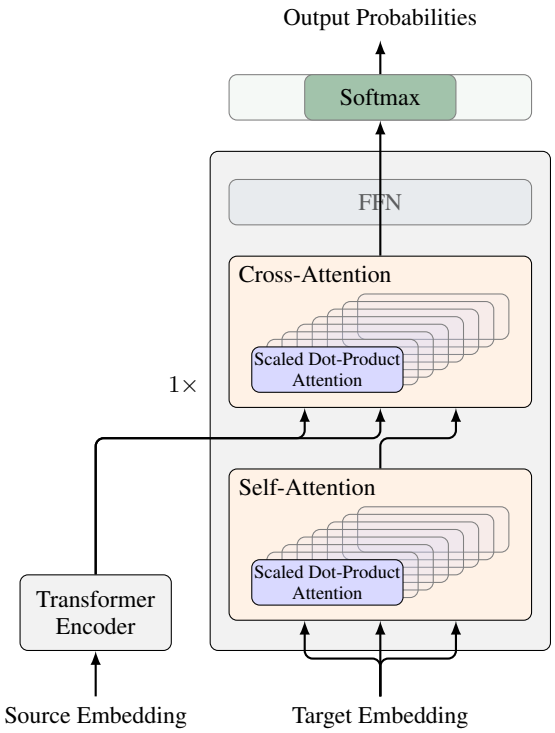

Figure 2: The model structure of our method (MDN).

Shallow Decoder. Recent work has shown that the deep encoder and shallow decoder architecture is promising in system speedup (Kasai et al., 2020; Li et al., 2021). In this work we follow the same idea by restricting the decoder to a 1-layer network and stacking more encoder layers until the total number of parameters matches the baseline.

Pruning Heads. Researchers have found that most heads could be safely pruned and leaving the performance intact (Voita et al., 2019; Michel et al., 2019). So we retain only one head in decoder attentions.

Dropping FFN. Hsu et al. (2020) suggests that FFN is the least important component in the decoder. So we drop all FFNs in the decoder. After dropping FFN, there are only attentions and no other non-linearity except layer normalization in the model.

Factorizing Output. The weight matrix W

System	BLEU	Speed (sent./s)	
		CPU	GPU
Baseline1	27.45	7.72	149.01
+ Shallow Decoder	26.46	19.44	247.04
+ Pruning Heads	26.62	12.91	172.25
+ Dropping FFN	26.91	8.58	189.86
+ Factorizing Output	27.14	7.92	168.21

Table 2: Results of adding each trick from Section 2.2 independently on WMT14 En-De (sent./s: translated sentences per second. Baseline1 is the baseline with 10K BPE merge operations).

System	BLEU	Speed (sent./s)	
		CPU	GPU
Baseline2	22.83	23.19	291.00
+ Deep Configuration	23.78	24.21	305.03
+ WD	26.97	24.56	352.09
- Decoder Dropout	23.25	24.51	277.38
- Label Smoothing	23.22	24.42	286.54

Table 3: Results of adding each trick from Section 2.3 independently on WMT14 En-De (sent./s: translated sentences per second. Baseline2 is the baseline with 10K BPE merge operations and tricks from Section 2.2).

used in the output projection is significantly over-parameterized (Grave et al., 2017), especially when other components are compressed. To address this, we employ the low-rank approximation (Lan et al., 2020) for this matrix $W = AB^T$ to help to reduce the computation cost, where $A \in \mathbb{R}^{V \times E}$, $B \in \mathbb{R}^{H \times E}$, V is the vocabulary size, E is the desired rank and H is the hidden size. We choose $E = 64$ in our experiments.

2.3 Training Strategies

In our work, the methods presented in Section 2.2 can make an extremely small decoder that contains only 0.3% of the overall parameters. But we find that the model learned from scratch using the standard setting is much worse than the baseline (see Table 2). For better training, some methods are necessary. Table 3 illustrates how each proposed strategy contributes to reaching performance on par with the baseline. We choose Baseline2 in Table 3 to analyze how each method in "Training Strategies" influences the results after we have changed the BPE merge operations and simplified the model structure.

Deep Configuration. Because our model is deep, we follow the deep model training setup provided in Wang et al. (2019).

Weight Distillation. We also adopt a simplified version of weight distillation (WD) for training (Lin et al., 2020). This method initializes the student model with the corresponding weights from the teacher model, e.g., the first layer in the teacher encoder is reused in the first layer in the student encoder. Then it trains the student as in standard knowledge distillation (Hinton et al., 2015). Since our encoder is much deeper than the baseline, we initialize it in a round-robin manner. For the decoder, we randomly select one head from the teacher model for initialization, and the low-rank approximation of output projection is initialized by the SVD result of the teacher output projection (Golub and Reinsch, 2007).

Weak Regularization. Because our decoder is small, we do not need to impose a strong regularization on it. We remove the dropout in the decoder and label smoothing. Dropout and label smoothing indeed do not have impacts on the inference speed. But changing them will train different models, which are unlikely to have exactly the same behavior. So some deviations in the inference speed are expected.

3 Experiments

3.1 Setup

We evaluate our methods on the WMT14 En-De, WMT14 En-Fr and NIST12 Zh-En machine translation tasks. We tokenize every sentence using a script from Moses and segment every word into subword units using BPE (Sennrich et al., 2016). The number of the BPE merge operations is set to 32K in the baseline and 10K for the target language in our model. In addition, we remove sentences with more than 250 subword units (Xiao et al., 2012).

We choose Transformer-base (Vaswani et al., 2017) as our baseline. The hyper-parameters of the Mini-Decoder Network (MDN) are the same as the baseline except for those mentioned in Section 2.3. To produce consistent results for distillation, we choose the baseline with 10K BPE merges as the teacher model, which has the same vocabulary as MDN. We also compare our system with some recent proposed fast Transformer variants, e.g., AAN (Zhang et al., 2018), SAN (Xiao et al., 2019) and CAN (Li et al., 2021). Their settings are

	System	Test	Valid	Speed (CPU)	Speedup	Speed (GPU)	Speedup	#Params
En-De	Baseline	27.21	25.53	7.17 sent./s	1.00×	129.02 sent./s	1.00×	96M
	+ BPE 10K	27.45	25.60	7.72 sent./s	1.08×	149.01 sent./s	1.15×	76M
	AAN	27.05	25.11	9.82 sent./s	1.37×	147.11 sent./s	1.14×	96M
	SAN	26.88	24.99	9.10 sent./s	1.27×	179.49 sent./s	1.39×	80M
	CAN	27.20	25.23	8.00 sent./s	1.12×	279.39 sent./s	2.17×	100M
	MDN	27.23	25.37	23.52 sent./s	3.28×	326.78 sent./s	2.53×	96M
En-Fr	Baseline	40.70	46.74	5.80 sent./s	1.00×	130.56 sent./s	1.00×	111M
	+ BPE 10K	40.52	46.51	6.50 sent./s	1.12×	143.21 sent./s	1.10×	81M
	AAN	40.44	46.43	8.00 sent./s	1.38×	140.66 sent./s	1.08×	96M
	SAN	40.70	46.44	7.48 sent./s	1.29×	157.50 sent./s	1.21×	80M
	CAN	40.16	46.40	6.40 sent./s	1.10×	194.67 sent./s	1.49×	100M
	MDN	40.58	46.43	22.86 sent./s	3.94×	352.79 sent./s	2.70×	100M
Zh-En	Baseline	45.84	50.98	4.09 sent./s	1.00×	90.87 sent./s	1.00×	102M
	+ BPE 10K	45.34	51.41	4.35 sent./s	1.06×	91.79 sent./s	1.01×	79M
	AAN	44.87	51.26	5.00 sent./s	1.22×	91.30 sent./s	1.00×	102M
	SAN	44.82	50.73	4.99 sent./s	1.22×	123.18 sent./s	1.36×	102M
	CAN	40.11	46.25	6.09 sent./s	1.49×	168.89 sent./s	1.86×	107M
	MDN	44.51	51.43	17.07 sent./s	4.17×	211.31 sent./s	2.33×	99M

Table 4: Results on the WMT14 En-De and En-Fr tasks (sent./s: translated sentences per second).

followed from their papers.

We report case-sensitive tokenized BLEU scores. For all experiments, we test on the model ensemble by averaging the last 5 checkpoints. For inference, we use a batch size of 64 and a beam width of 4. All models are evaluated on the NVIDIA TITAN V GPU and Intel(R) Xeon(R) Gold 5118 CPU.

3.2 Results

Table 4 shows the results of various systems. In both tasks, our method (MDN) has nearly the same performance as the baseline, but its speed is 3.80× and 2.52× faster on average for CPU and GPU. We find the baseline with 10K BPE merges is about 1.09× faster than the original baseline but with a similar performance, which suggests this BPE hyper-parameter is far from optimal for the baseline.

As for the recent work, i.e., AAN, SAN and CAN, all of them achieve performance similar to the baseline and are faster than the baseline (1.24×∼1.32× speedup on CPU) as reported in their papers. But our method outperforms these methods and runs consistently faster (3.80× speedup on CPU). Although the acceleration of our method in GPU (2.52× speedup) is not as obvious as it in CPU (3.80× speedup), it still outperforms CAN by 1.40×, which is highly optimized for GPU.

System	BLEU	Speed (sent./s)	
		CPU	GPU
Baseline	27.21	7.17	129.02
+ Merge Operations	27.45	7.72	149.01
+ Shallow Decoder	26.46	19.44	247.04
+ Pruning Heads	24.47	21.99	243.42
+ Dropping FFN	23.26	22.36	268.01
+ Output Factorization	22.83	23.19	291.00
+ Deep Configuration	23.78	24.21	305.03
+ WD	27.09	22.96	344.21
- Decoder Dropout	27.18	23.53	334.40
- Label Smoothing	27.23	23.52	326.78

Table 5: Ablation study on WMT14 En-De. The colors refer to Byte-Pair Encoding, Model Structure Updates and Training Strategies (sent./s: translated sentences per second).

4 Analysis

4.1 Ablation Study

Table 5 summarizes and compares the contributions of each proposed tricks described in Section 2. Each row of Table 5 is the result of applying the current trick to the system obtained in the previous row. This way helps to illustrate the compound effect of these tricks.

We observe that using any structure simplification trick brings a significant performance degra-

	System	Test	Valid	Speed(CPU)	Speedup	Speed(GPU)	Speedup
deep	Baseline	29.43	27.82	6.02 sent./s	1.00×	121.57 sent./s	1.00×
	+ BPE 10K	29.67	27.68	6.19 sent./s	1.03×	126.06 sent./s	1.04×
	MDN	29.02	27.64	15.00 sent./s	2.49×	254.25 sent./s	2.09×

Table 6: Results of Transformer-deep on WMT14 En-De (sent./s: translated sentences per second).

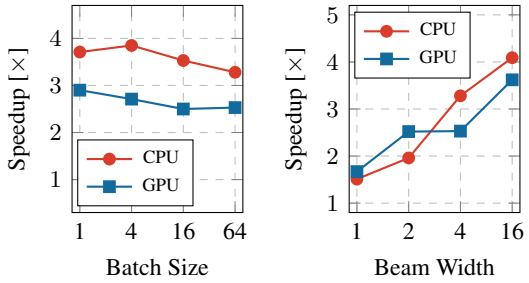

Figure 3: Batch size and beam width vs. speedup of MDN on WMT14 En-De.

System	BLEU	Speed (sent./s)	
		CPU	GPU
AAN	27.05	9.82	147.11
+ Ours	27.21	19.32	259.10
CAN	27.20	8.00	279.39
+ Ours	27.25	10.62	293.43

Table 7: Combining the proposed tricks with AAN and CAN on WMT14 En-De (sent./s: translated sentences per second).

dation but a considerable speedup. However, our training strategies can make up for the performance loss. Among them, WD is most effective and has a boost of more than 3 BLEU points. Interestingly, we find that pruning attention heads does not have any speedup on GPU. This is because the parallelism of GPU decouples the computation cost of attention heads from the head number.

4.2 Experiments on Larger Networks

Table 6 shows the results of the Transformer-deep model with a 48-layer encoder. The phenomenon here is similar to that in Table 4. The acceleration on Transformer-deep is less obvious than on Transformer-base, as a deeper encoder consumes more inference time. Moreover, compared with such a strong Transformer-deep teacher, MDN can still obtain a 2.49× speedup on CPU and a 2.09× speedup on GPU.

4.3 Sensitivity Analysis

We study the impact of two commonly tuned hyper-parameters at inference on our method, i.e., the batch size and the beam size. The left part of Fig. 3 shows that the speedup over the baseline decreases as the batch size increases, especially for GPU. This is because our shallow decoder trick exploits the parallelism of the encoder for speedup, which is not available if the batch size is large. This phenomenon is also observed by Li et al. (2021). The right part of Fig. 3 shows that the speedup is more obvious with a larger beam width. The reason is that the encoder occupies a larger portion of the inference cost at a small beam width and our work only save the cost of the decoder.

4.4 Combining with Other Models

Our method is a bag of generic tricks and can be applied to other models. We choose AAN and CAN for testing, because AAN runs the fastest on CPU and CAN runs the fastest on GPU according to Table 4. Table 7 shows that both AAN and CAN benefit from techniques presented in this paper. Without loss in performance, AAN obtains a 1.97× speedup on CPU and CAN obtains a 1.05× speedup on GPU. It shows that AAN and CAN eventually have a similar BLEU score and speed as MDN, indicating that a highly optimized Transformer baseline already a strong candidate by itself.

5 Conclusion

In this work, we present a bag of tricks to optimize the efficiency of the standard Transformer. The resulting model achieves a 3.61× speedup on CPU and a 2.62× speedup on GPU without loss in performance.

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), and the Ministry of Science and Technology of the PRC (Nos. 2019YFF0303002 and 2020AAA0107900). The authors would like to thank anonymous reviewers for their comments.

References

Maximiliana Behnke and Kenneth Heafield. 2020. Losing heads in the lottery: Pruning transformer attention in neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 2664–2674. Association for Computational Linguistics.

Insoo Chung, Byeongwook Kim, Yoonjung Choi, Se Jung Kwon, Yongkweon Jeon, Baeseong Park, Sangha Kim, and Dongsoo Lee. 2020. Extremely low bit transformer quantization for on-device neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, Online Event, 16-20 November 2020*, pages 4812–4826. Association for Computational Linguistics.

Gene H. Golub and Christian H. Reinsch. 2007. Singular value decomposition and least squares solutions. In Raymond Hon-Fu Chan, Chen Greif, and Dianne P. O'Leary, editors, *Milestones in Matrix Computation - Selected Works of Gene H. Golub, with Commentaries*, pages 160–180. Oxford University Press.

Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. 2017. Efficient softmax approximation for gpus. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1302–1310. PMLR.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2017. Non-autoregressive neural machine translation. *CoRR*, abs/1711.02281.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *CoRR*, abs/2102.00554.

Yi-Te Hsu, Sarthak Garg, Yi-Hsiu Liao, and Ilya Chatsviorkin. 2020. Efficient inference for neural machine translation. *CoRR*, abs/2010.02416.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. 2020. Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation. *CoRR*, abs/2006.10369.

Young Jin Kim, Marcin Junczys-Dowmunt, Hany Hassan, Alham Fikri Aji, Kenneth Heafield, Roman Grundkiewicz, and Nikolay Bogoychev. 2019. From research to production and back: Ludicrously fast neural machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation@EMNLP-IJCNLP 2019, Hong Kong, November 4, 2019*, pages 280–288. Association for Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Yanyang Li, Ye Lin, Tong Xiao, and Jingbo Zhu. 2021. An efficient transformer decoder with compressed sub-layers. *CoRR*, abs/2101.00542.

Ye Lin, Yanyang Li, Ziyang Wang, Bei Li, Quan Du, Tong Xiao, and Jingbo Zhu. 2020. Weight distillation: Transferring the knowledge in neural network parameters. *CoRR*, abs/2009.09152.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14014–14024.

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, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826. IEEE Computer Society.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 5797–5808. Association for Computational Linguistics.

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 Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 1810–1822. Association for Computational Linguistics.

Wenxuan Wang and Zhaopeng Tu. 2020. Rethinking the value of transformer components. In *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pages 6019–6029. International Committee on Computational Linguistics.

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.

Tong Xiao, Jingbo Zhu, Hao Zhang, and Qiang Li. 2012. Niutrans: An open source toolkit for phrase-based and syntax-based machine translation. In *The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the System Demonstrations, July 10, 2012, Jeju Island, Korea*, pages 19–24. The Association for Computer Linguistics.

Biao Zhang, Deyi Xiong, and Jinsong Su. 2018. Accelerating neural transformer via an average attention network. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 1789–1798. Association for Computational Linguistics.