Transformers without Tears: Improving the Normalization of Self-Attention

Toan Q. Nguyen[*,1] *Julian Salazar*[*,2]

[1]University of Notre Dame [2]Amazon AWS AI

tnguye28@nd.edu julsal@amazon.com

Abstract

We evaluate three simple, normalization-centric changes to improve Transformer training. First, we show that pre-norm residual connections (PRENORM) and smaller initializations enable warmup-free, validation-based training with large learning rates. Second, we propose ℓ_2 normalization with a single scale parameter (SCALENORM) for faster training and better performance. Finally, we reaffirm the effectiveness of normalizing word embeddings to a fixed length (FIXNORM). On five low-resource translation pairs from TED Talks-based corpora, these changes always converge, giving an average +1.1 BLEU over state-of-the-art bilingual baselines and a new 32.8 BLEU on IWSLT'15 English-Vietnamese. We observe sharper performance curves, more consistent gradient norms, and a linear relationship between activation scaling and decoder depth. Surprisingly, in the high-resource setting (WMT'14 English-German), SCALENORM and FIXNORM remain competitive but PRENORM degrades performance.

1. Introduction

The Transformer [1] has become the dominant architecture for neural machine translation (NMT) due to its train-time parallelism and strong downstream performance. Various modifications have been proposed to improve the efficiency of its multi-head attention and feedforward sublayers [2, 3]. Our work focuses on *layer normalization* (LAYERNORM) [4], which we show has an outsized role in the convergence and performance of the Transformer in two ways:

Placement of normalization. The original Transformer uses *post-norm residual units* (POSTNORM), where layer normalization occurs after the sublayer and residual addition. However, [5] found that *pre-norm residual units* (PRENORM), where layer normalization occurs immediately before the sublayer, were instrumental to their model's performance. [6] compares the two, showing that PRENORM makes backpropagation more efficient over depth and training Transformers with deep, 30-layer encoders.

Our work demonstrates additional consequences in the base ($\leq$6-layer encoder) Transformer regime. We show that PRENORM enables warmup-free, validation-based training with large learning rates even for small batches, in contrast to past work on scaling NMT [7]. We also partly reclaim POSTNORM's stability via smaller initializations, although PRENORM is less sensitive to this magnitude and can improve performance. However, despite PRENORM's recent adoption in many NMT frameworks, we find it degrades base Transformer performance on WMT'14 English-German.

Choice of normalization. [8] shows that batch normalization's effectiveness is not from reducing internal covariate shift, but from smoothing the loss landscape. They achieve similar or better performance with non-variance-based normalizations in image classification. Hence, we propose replacing LAYERNORM with the simpler *scaled ℓ_2 normalization* (SCALENORM), which normalizes activation vectors to a *single* learned length g. This is both inspired by and synergistic with jointly fixing the word embedding lengths (FIXNORM) [9]. These changes improve the training speed and low-resource performance of the Transformer without affecting high-resource performance.

On five low-resource pairs from the TED Talks [10] and IWSLT'15 [11] corpora, we first train state-of-the-art Transformer models (+4.0 BLEU on average over the best published NMT bitext-only numbers). We then apply PRENORM, FIXNORM, and SCALENORM for an average total improvement of +1.1 BLEU, where each addition contributes at least +0.3 BLEU (Section 3), and attain a new 32.8 BLEU on IWSLT'15 English-Vietnamese. We validate our intuitions in Section 4 by showing sharper performance curves (i.e., improvements occur at earlier epochs) and more consistent gradient norms. We also examine the per-sublayer g's learned by SCALENORM, which suggest future study.

2. Background

2.1. Identity mappings for transformers

Residual connections [12] were first introduced to facilitate the training of deep convolutional networks, where the output of the ℓ-th layer F_ℓ is summed with its input:

$$\boldsymbol{x}_{\ell+1} = \boldsymbol{x}_\ell + F_\ell(\boldsymbol{x}_\ell). \tag{1}$$

The identity term $\boldsymbol{x}_\ell$ is crucial to greatly extending the depth of such networks [13]. If one were to scale $\boldsymbol{x}_\ell$ by a scalar λ_ℓ, then the contribution of $\boldsymbol{x}_\ell$ to the final layer F_L is $(\prod_{i=\ell}^{L-1} \lambda_i)\boldsymbol{x}_\ell$. For deep networks with dozens or even hundreds of layers L, the term $\prod_{i=\ell}^{L-1} \lambda_i$ becomes very large if

[*]Equal contribution.

[1]Work was done during an internship at Amazon AWS AI.

$\lambda_i > 1$ or very small if $\lambda_i < 1$, for enough i. When backpropagating from the last layer L back to ℓ, these multiplicative terms can cause exploding or vanishing gradients, respectively. Therefore they fix $\lambda_i = 1$, keeping the total residual path an identity map.

The original Transformer applies LAYERNORM after the sublayer and residual addition:

$$\text{POSTNORM}: \quad \boldsymbol{x}_{\ell+1} = \text{LAYERNORM}(\boldsymbol{x}_\ell + F_\ell(\boldsymbol{x}_\ell)). \quad (2)$$

We conjecture this has caused past convergence failures [14, 15], with LAYERNORMs in the residual path acting similarly to $\lambda_i \neq 1$; furthermore, warmup was needed to let LAYERNORM safely adjust scale during early parts of training. Inspired by [13], we apply LAYERNORM immediately before each sublayer:

$$\text{PRENORM}: \quad \boldsymbol{x}_{\ell+1} = \boldsymbol{x}_\ell + F_\ell(\text{LAYERNORM}(\boldsymbol{x}_\ell)). \quad (3)$$

This is cited as a stabilizer for Transformer training [5, 6] and is already implemented in popular toolkits [16, 17, 18], though not necessarily used by their default recipes. [6] makes a similar argument to motivate the success of PRENORM in training very deep Transformers. Note that one must append an additional normalization after both encoder and decoder so their outputs are appropriately scaled. We compare POSTNORM and PRENORM throughout Section 3.

2.2. Weight initialization

Xavier normal initialization [19] initializes a layer's weights $\boldsymbol{W}_\ell \in \mathbb{R}^{d_{\ell+1} \times d_\ell}$ (d_ℓ is the hidden dimension) with samples from a centered normal distribution with layer-dependent variance:

$$(\boldsymbol{W}_\ell)_{i,j} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{d_\ell + d_{\ell+1}}}\right). \quad (4)$$

Our experiments with this default initializer find that POSTNORM sometimes fails to converge, especially in our low-resource setting, even with a large number of warmup steps. One explanation is that Xavier normal yields initial weights that are too large. In implementations of the Transformer, one scales the word embeddings by a large value (e.g., $\sqrt{d} = 22.6$ for $d = 512$), giving vectors with an expected square norm of d. LAYERNORM's unit scale at initialization preserves this same effect. Since feedforward layers already have their weights initialized to a smaller standard deviation, i.e., $\sqrt{\frac{2}{d+4d}}$, we propose reducing the attention layers' initializations from $\sqrt{\frac{2}{d+d}}$ to $\sqrt{\frac{2}{d+4d}}$ as well (SMALLINIT), as a corresponding mitigation. We evaluate the effect of this on POSTNORM vs. PRENORM in Section 3.1.

2.3. Scaled ℓ_2 normalization and FIXNORM

LAYERNORM is inspired by batch normalization [20], both of which aim to reduce internal covariate shift by fixing the mean and variance of activation distributions. Both have been applied to self-attention [1, 21]. However, [8] shows that batch normalization's success has little to do with covariate shift, but comes instead from smoothing the loss landscape. For example, they divide by the pre-centered ℓ_p norm instead of the variance and achieve similar or better results in image classification.

Hence, we propose replacing LAYERNORM with *scaled ℓ_2 normalization*:

$$\text{SCALENORM}(\boldsymbol{x}; g) = g\frac{\boldsymbol{x}}{||\boldsymbol{x}||}. \quad (5)$$

This can be viewed as projecting d-dimensional vectors onto a $(d-1)$-dimensional hypersphere with learned radius g. This expresses the inductive bias that each sublayer's activations has an ideal "global scale," a notion we empirically validate in Section 4.2. SCALENORM replaces the $2d$ scale and shift parameters of LAYERNORM with a single learned scalar, improving computational and parameter efficiency while potentially regularizing the loss landscape.

This bias has an explicit interpretation at the final layer: large inner products sharpen the output distribution, causing frequent words to disproportionately dominate rare words. This led [9] to introduce $\text{FIXNORM}(\boldsymbol{w}) = g\frac{\boldsymbol{w}}{||\boldsymbol{w}||}$ with fixed g at the last linear layer, to maximize the angular difference of output representations and aid rare word translation. By making g learnable, we can apply SCALENORM and FIXNORM jointly, which means applying the following at the final linear layer:

$$(\text{SCALENORM} + \text{FIXNORM})(\boldsymbol{x}, \boldsymbol{w}; g) = g\frac{\boldsymbol{w} \cdot \boldsymbol{x}}{||\boldsymbol{w}||||\boldsymbol{x}||}. \quad (6)$$

Note that this combination at the last layer is equivalent to cosine normalization [22] with a learned scale.

2.4. Learning rates

Despite using an adaptive optimizer, Adam [23], Transformer training uses a learning rate (LR) schedule with a linear *warmup* and an inverse square root *decay* (INVSQRTDECAY):

$$\text{LR}(n) = \frac{\lambda}{\sqrt{d}} \min\left(\frac{1}{\sqrt{n}}, \frac{n}{n_{\text{warmup}}^{1.5}}\right), \quad (7)$$

where d is the hidden dimension of the self-attention layers, and λ, n_{warmup} are hyperparameters that determine the highest learning rate achieved and the number of steps to reach it, respectively. These two hyperparameters have been the subject of much empirical study [14, 7]. In light of our modifications however, we revisit various aspects of this schedule:

Warmup-free training. We conjectured that warmup is primarily needed when using POSTNORM to gradually learn LAYERNORM parameters without gradient explosion/vanishing (Section 2.1). Hence, we evaluate both PRENORM and POSTNORM without warmup in Section 3.3.

	# egs.	# src. + tgt. tokens	# iters./epoch	max. epoch	# enc./dec. layers	# heads/layer	dropout	# BPE
gl→en	10k	0.37M	100	1000	4	4	0.4	3k
sk→en	61k	2.32M	600	200	6	8	0.3	8k
en→vi	133k	5.99M	1500	200	6	8	0.3	8k
en→he	212k	7.88M	2000	200	6	8	0.3	8k
ar→en	214k	8.09M	2000	200	6	8	0.3	8k

Table 1: Data and model properties for low-resource NMT. *en→vi* is from IWSLT 2015; the rest are from the TED Talks corpus.

Large learning rates. To speed up training, one often explores using larger learning rates. In the context of Transformer, [7, 24] take $\lambda \in \{2, 3\}$ instead of the conventional $\lambda = 1$. [7] showed that one can scale up Adam's learning rate to 10^{-3} with an extremely large batch (400k tokens). However, the improved convergence provided by our modifications could enable higher learning rates with much small batch sizes (4k tokens), as examined in Section 3.3.

Validation-based decay. For similar reasons, one might wish to adopt a classic validation-based decay, i.e., training at a high learning rate for as long as tenable, decaying rapidly when development scores flatline. This has inspired usage of fixed decay schemes upon convergence with INVSQRTDECAY [25, 26]. We revisit VALDECAY under our modifications, where we still perform a linear warmup but then multiply by a scale $\alpha_{\text{decay}} < 1$ when performance on a development set does not improve over $patience$ evaluations.

3. Experiments & results

We train Transformer models for a diverse set of five low-resource translation pairs from the TED Talks [10] and the IWSLT '15 [11] corpora. Details are summarized in Table 1. For more information motivating our choice of pairs and for exact training details, refer to Appendix A.

3.1. Large vs. small initialization

To see the impact of weight initialization, we run training on the *en→vi* dataset using warmup steps of 4k, 8k, 16k (Table 2). With default initialization, POSTNORM fails to converge on this dataset even with a long warmup of 16k steps, only reaching 5.76 BLEU.

Xavier normal		# warmup steps 4k	8k	16k
Baseline	POSTNORM	fail	fail	5.76
	PRENORM	28.52	28.73	28.32
SMALLINIT	POSTNORM	28.17	28.20	28.62
	PRENORM	28.26	28.44	28.33

Table 2: Development BLEU on *en→vi* using Xavier normal initialization (baseline versus SMALLINIT).

The second row shows that taking a smaller standard deviation on the attention weights (SMALLINIT) restores convergence to POSTNORM. Though the $\sqrt{2/5} \approx 0.63$ adjustment used here seems marginal, operations like residual connections and the products between queries and keys can compound differences in scale. Though both models now achieve similar performance, we note that PRENORM works in all setups, suggesting greater stability during training. For all remaining experiments, we use POSTNORM and PRENORM with SMALLINIT. We find this choice does not affect the performance of PRENORM.

3.2. Scaled ℓ_2 normalization and FIXNORM

To compare SCALENORM and LAYERNORM, we take 8k warmup steps for all further experiments. Since we tie the target input word embedding and the last linear layer's weight (Appendix A), FIXNORM is implemented by applying ℓ_2 normalization to the word embedding, with each component initialized uniformly in $[-0.01, 0.01]$. For non-FIXNORM models, word embeddings are initialized with mean 0 and standard deviation $\sqrt{1/d}$ so they sum to unit variance. All g's in SCALENORM are initialized to $\sqrt{d}$.

Table 3 shows our results along with some published baselines. First, note that our Transformer baselines with POSTNORM + LAYERNORM (1) are very strong non-multilingual NMT models on these pairs. They outperform the best published numbers, which are all Transformer models in the past year, by an average margin of +4.0 BLEU. Then, we see that PRENORM (2) achieves comparable or slightly better results than POSTNORM on all tasks. FIXNORM (3) gives an additional gain, especially on ar→en ($p < 0.01$).

Finally, we replace LAYERNORM with SCALENORM (4). SCALENORM significantly improves on LAYERNORM for two very low-resource pairs, gl→en and sk→en. On the other tasks, it performs comparably to LAYERNORM. Upon aggregating all changes, our final model with SCALENORM and FIXNORM improves over our strong baseline with POSTNORM on all tasks by an average of +1.1 BLEU ($p < 0.01$), with each change contributing an average of at least +0.3 BLEU. In Section 4.2 and Appendix B, we further examine where the performance gains of SCALENORM come from.

Moreover, SCALENORM is also faster than LAYERNORM. Recall that for each vector of size d, LAYERNORM

	gl→en	sk→en	*en→vi*	en→he	ar→en	average Δ
POSTNORM + LAYERNORM [27, 28, 24]	16.2	24.0	29.09	23.66	27.84	-4.05
POSTNORM + LAYERNORM (1)	18.47	29.37	31.94	27.85	33.39	+0.00
PRENORM + LAYERNORM (2)	19.09	29.45	31.92	28.13	33.79	+0.27
PRENORM + FIXNORM + LAYERNORM (3)	19.38	29.50	32.45	28.39	$34.35^{\dagger}$	+0.61
PRENORM + FIXNORM + SCALENORM (4)	$20.91^{\ddagger *}$	$30.25^{\ddagger *}$	32.79^{*}	28.44^{*}	34.15^{*}	+1.10

Table 3: Test BLEU using POSTNORM or PRENORM and different normalization techniques. †, ‡ and $*$ indicate significant improvement of (3) over (2), (4) over (3), and (4) over (1), respectively; $p < 0.01$ via bootstrap resampling [29].

	gl→en	sk→en	*en→vi*	en→he	ar→en
NOWARMUP	18.00	28.92	28.91	30.33	35.40
INVSQRTDECAY	22.18	29.08	28.84	30.30	35.33
VALDECAY	21.45	29.46	28.67	30.69	35.46
INVSQRTDECAY + 2×LR	21.92	29.03	28.76	30.50	35.33
VALDECAY + 2×LR	21.63	29.49	28.46	30.13	34.95

Table 4: Development BLEU for PRENORM + FIXNORM + SCALENORM, trained with different learning rate schedulers.

needs to compute mean, standard deviation, scaling, and shifting, which costs $O(7d)$ operations. For SCALENORM, we only need $O(3d)$ operations to perform normalization and global scaling. This does not account for further gains due to reduction in parameters. In our implementation, training with SCALENORM is around 5% faster than with LAYERNORM, similar to the speedups on NMT observed by [30]'s RMSNORM (which can be viewed as SCALENORM with per-unit scales; see Section 4.2).

3.3. Learning rates

We compare the original learning rate schedule in equation 7 (INVSQRTDECAY) with validation-based decay (VALDECAY), possibly with no warmup (NOWARMUP). We use $\lambda = 1$, $n_{warmup} = 8$k for INVSQRTDECAY and VALDECAY. For NOWARMUP, we instead use a learning rate of $3 \cdot 10^{-4}$ for all datasets. For both VALDECAY and NOWARMUP, we take $\alpha_{decay} = 0.8$ and $patience = 3$. For experiments with high learning rate, we use either VALDECAY or INVSQRTDECAY with $\lambda = 2$ (giving a peak learning rate of $\approx 10^{-3}$). All experiments use PRENORM + FIXNORM + SCALENORM.

In Table 4, we see that NOWARMUP performs comparably to INVSQRTDECAY and VALDECAY except on gl→en. We believe that in general, one can do without warmup, though it remains useful in the lowest resource settings. In our 2×LR experiments, we can still attain a maximum learning rate of 10^{-3} without disproportionately overfitting to small datasets like gl→en.

One might hypothesize that VALDECAY converges more quickly to better minima than INVSQRTDECAY by staying at high learning rates for longer. However, both schedulers achieve similar results with or without doubling the learning rate. This may be due to the tail-end behavior of VALDECAY methods, which can involve multiplicative decays in rapid succession. Finally, our 2×LR experiments, while not yielding better performance, show that PRENORM allows us to train the Transformer with a very high learning rate despite small batches (4k tokens).

Since PRENORM can train without warmup, we wonder if POSTNORM can do the same. We run experiments on *en→vi* with NOWARMUP, varying the number of encoder/decoder layers. As seen in Table 5, POSTNORM often fails without warmup even with 5 or 6 layers. Even at 4 layers, one achieves a subpar result compared to PRENORM. This reaffirms Section 3.1 in showing that PRENORM is more stable than POSTNORM under different settings.

	4 layers	5 layers	6 layers
POSTNORM	18.31	fails	fails
PRENORM	28.33	28.13	28.32

Table 5: Development BLEU on *en→vi* using NOWARMUP, as number of encoder/decoder layers increases.

3.4. High-resource setting

Since all preceding experiments were in low-resource settings, we examine if our claims hold in a high-resource setting. We train the Transformer base model on WMT'14 English-German using FAIRSEQ and report tokenized BLEU scores on *newstest2014*. Implementation of our methods in FAIRSEQ can be found in Appendix C.

In Table 6, SCALENORM and FIXNORM achieve equal or better results than LAYERNORM. Since SCALENORM is also faster, we recommend using both as drop-in replace-

ments for LAYERNORM in all settings. Surprisingly, in this task POSTNORM works notably better than PRENORM; one observes similar behavior in [6]. We speculate this is related to identity residual networks acting like shallow ensembles [31] and thus undermining the learning of the longest path; further study is required.

	newstest2014
POSTNORM + LAYERNORM [1]	27.3
PRENORM + LAYERNORM	26.83
PRENORM + FIXNORM + SCALENORM	27.07
POSTNORM + LAYERNORM	27.58
POSTNORM + FIXNORM + SCALENORM	27.57

Table 6: BLEU scores from WMT '14 English-to-German.

4. Analysis

4.1. Performance curves

Figure 1 shows that PRENORM not only learns faster than POSTNORM, but also outperforms it throughout training. Adding FIXNORM also gives faster learning at first, but only achieves close performance to that with PRENORM and no FIXNORM. However, once paired with SCALENORM, we attain a better BLEU score at the end. Because of the slow warmup period, SCALENORM with warmup learns slower than SCALENORM without warmup initially; however, they all converge at about the same rate.

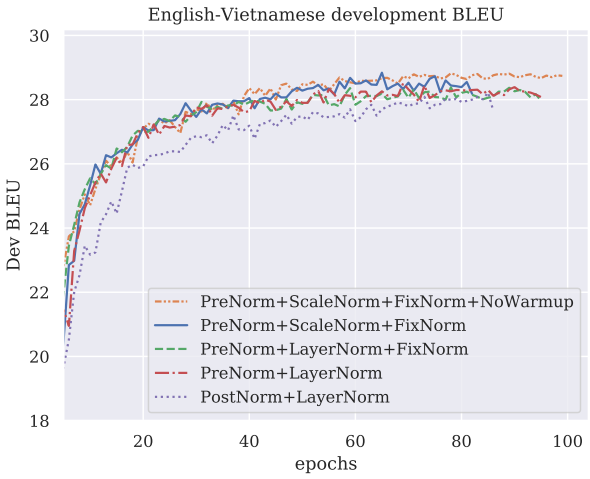

Figure 1: Development BLEU on *en→vi* with POSTNORM or PRENORM, and with LAYERNORM or SCALENORM.

To visualize how PRENORM helps backpropagation, we plot the global gradient norms from our runs in Figure 2. POSTNORM produces noisy gradients with many sharp spikes, even towards the end of training. On the other hand, PRENORM has fewer noisy gradients with smaller sizes, even without warmup. LAYERNORM has lower global norms than SCALENORM + FIXNORM but it has more gradient components corresponding to normalization.

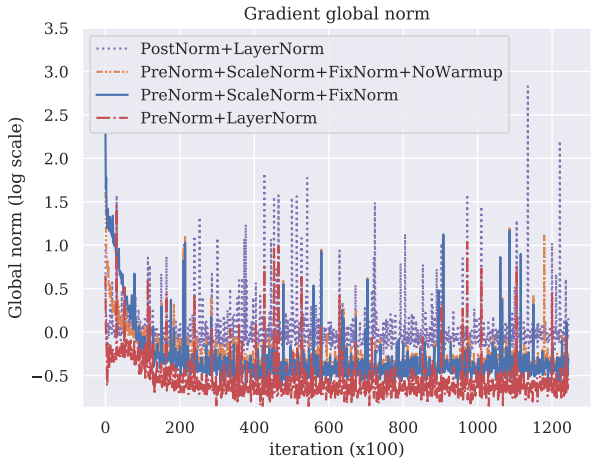

Figure 2: The global norm of gradients when using POSTNORM or PRENORM, and with LAYERNORM, SCALENORM and FIXNORM. Best viewed in color.

4.2. Activation scaling and the role of g

One motivation for SCALENORM was that it expressed a good inductive bias for the global scaling of activations, independent of distributional stability (Section 2.3). In contrast, a contemporaneous work [30] proposes *root mean square layer normalization* (RMSNORM), which still follows layer normalization's motivation but reduces overhead by forgoing additive adjustments, using only a scaling g_i per activation a_i. Despite their differing motives, tying the g_i of RMSNORM and dividing by $\sqrt{d}$ retrieves SCALENORM.

Hence we can frame our comparisons in terms of number of learnable parameters. We rerun our PRENORM experiments with RMSNORM. We also consider fixing $g = \sqrt{d}$ for SCALENORM, where only FIXNORM has learnable g. Table 7 shows that SCALENORM always performs comparably or better than RMSNORM. Surprisingly, the fixed-g model performs comparably to the one with learnable g. However, at higher learning rates (VALDECAY with and without 2×LR), fixed-g models perform much worse on ar→en, en→he and *en→vi*. We conjecture that learning g is required to accommodate layer gradients.

In Figure 3, we plot the learned g values for pairs with 100k+ examples. For all but the decoder-encoder sublayers, we observe a positive correlation between depth and g, giving credence to SCALENORM's inductive bias of global scaling. This trend is clearest in the decoder, where g linearly scales up to the output layer, perhaps in tandem with the discriminativeness of the hidden representations [32]. We also note a negative correlation between the number of training examples and the magnitude of g for attention sublayers, which may reflect overfitting.

	gl→en	sk→en	*en→vi*	en→he	ar→en
RMSNORM + FIXNORM	20.92	30.36	32.54	28.29	33.67
SCALENORM + FIXNORM	20.91	30.25	32.79	28.44	34.15
SCALENORM ($g = \sqrt{d}$) + FIXNORM (learned g)	21.18	30.36	32.66	28.19	34.11
SCALENORM ($g = \sqrt{d}$) + FIXNORM (learned g) + VALDECAY	20.36	30.45	32.83	27.97	33.98
SCALENORM ($g = \sqrt{d}$) + FIXNORM (learned g) + VALDECAY + 2×LR	21.15	30.57	31.81	25.00	28.92

Table 7: Test BLEU of ℓ_2-based normalization techniques with different numbers of learned g: $O(Ld)$ vs. $O(L)$ vs. $O(1)$.

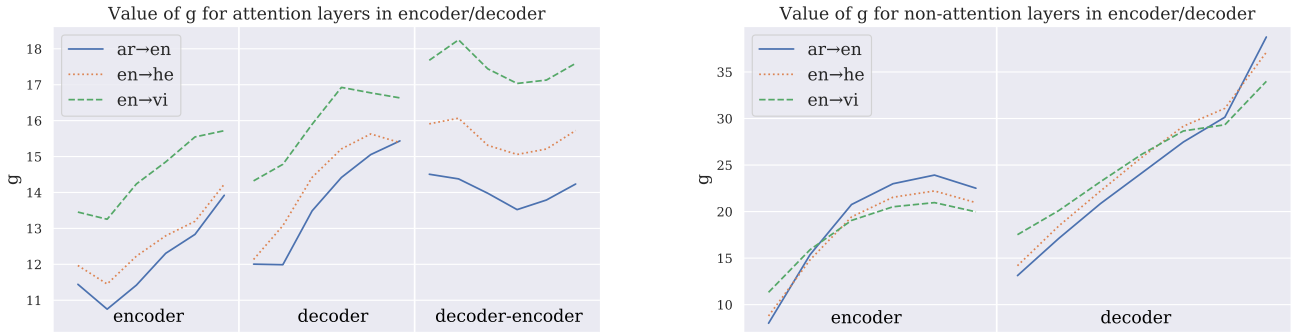

Figure 3: Learned g values for PRENORM + SCALENORM + FIXNORM models, versus depth. **Left:** Attention sublayers (*decoder-encoder* denotes decoder sublayers attending on the encoder). **Right:** Feedforward sublayers and the final linear layer.

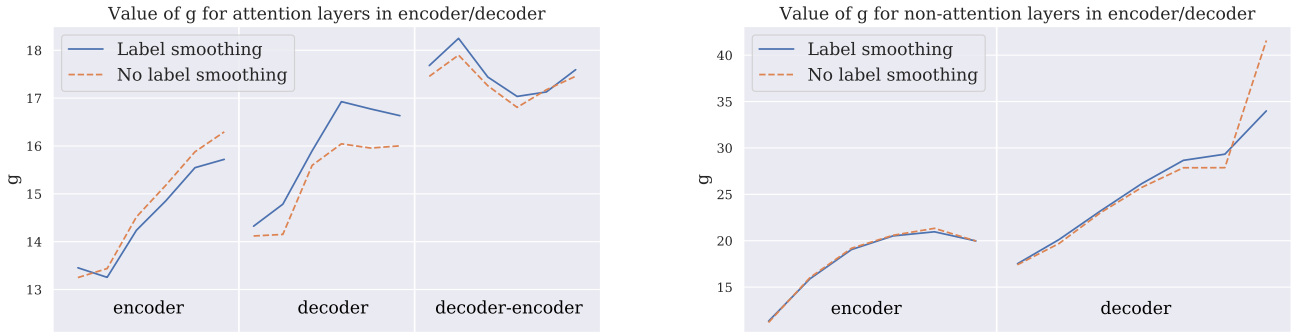

Figure 4: Learned g values for our PRENORM + SCALENORM + FIXNORM *en→vi* model (with and without label smoothing), versus depth. **Left** and **Right** are the same as in Figure 3.

Finally, to affirm our intuition for interpreting g, we plot g values with and without label smoothing (Figure 4). We see a difference in later layers of the decoder; there, removing label smoothing results in lower g values except at the output layer, where g increases sharply. This corresponds to the known overconfidence of translation models' logits, on which label smoothing has a downscaling effect [33].

5. Conclusion

In this work, we presented three simple, normalization-centric changes to the Transformer model, with a focus on NMT. First, we show that while POSTNORM performs better for high-resource NMT in the original base Transformer regime, PRENORM is both more stable and more competent in low-resource settings. Second, we propose replacing LAYERNORM with SCALENORM, a fast and effective *scaled* ℓ_2 *normalization* technique which requires only a single learned parameter. Finally, we reaffirm the effectiveness of fixing the word embedding norm (FIXNORM). Altogether, PRENORM + FIXNORM + SCALENORM significantly improves NMT on low-resource pairs, with the latter two performing comparably in the high-resource setting, but faster.

In the future, we would like to investigate the relationship between POSTNORM and PRENORM when using other optimizers such as RADAM [34], which has been shown to improve Transformer training without warmup. We are also interested in seeing if FIXNORM or SCALENORM at the final linear layer remains effective when paired with an initialization method such as FIXUP [35], which enables the training of deep neural networks without normalization. One could also explore using other ℓ_p norms [8].

6. Acknowledgements

The authors would like to thank David Chiang and Katrin Kirchhoff for their support of this research.

7. References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All you Need," in *NeurIPS*, 2017, pp. 5998–6008.

[2] Q. Guo, X. Qiu, P. Liu, Y. Shao, X. Xue, and Z. Zhang, "Star-Transformer," in *NAACL-HLT*, 2019, pp. 1315–1325.

[3] S. Sukhbaatar, E. Grave, G. Lample, H. Jegou, and A. Joulin, "Augmenting Self-attention with Persistent Memory," *CoRR*, vol. abs/1907.01470, 2019.

[4] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," *CoRR*, vol. abs/1607.06450, 2015.

[5] M. X. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, N. Parmar, N. Shazeer, A. Vaswani, J. Uszkoreit, L. Kaiser, M. Schuster, Z. Chen, Y. Wu, and M. Hughes, "The best of both worlds: Combining recent advances in neural machine translation," in *ACL*, 2018, pp. 76–86.

[6] Q. Wang, B. Li, T. Xiao, J. Zhu, C. Li, D. F. Wong, and L. S. Chao, "Learning Deep Transformer Models for Machine Translation," in *ACL*, 2019, pp. 1810–1822.

[7] M. Ott, S. Edunov, D. Grangier, and M. Auli, "Scaling Neural Machine Translation," in *WMT*, 2018, pp. 1–9.

[8] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" in *NeurIPS*, 2018, pp. 2483–2493.

[9] T. Nguyen and D. Chiang, "Improving Lexical Choice in Neural Machine Translation," in *NAACL-HLT*, 2018, pp. 334–343.

[10] Y. Qi, D. Sachan, M. Felix, S. Padmanabhan, and G. Neubig, "When and Why Are Pre-Trained Word Embeddings Useful for Neural Machine Translation?" in *NAACL-HLT*, 2018, pp. 529–535.

[11] M. Cettolo, J. Niehues, L. Bentivogli, R. Cattoni, and M. Federico, "The IWSLT 2015 Evaluation Campaign," in *IWSLT*, 2015, pp. 3–4.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[13] ——, "Identity Mappings in Deep Residual Networks," in *ECCV*, 2016, pp. 630–645.

[14] M. Popel and O. Bojar, "Training Tips for the Transformer Model," *Prague Bull. Math. Linguistics*, vol. 110, no. 1, pp. 43–70, 2018.

[15] N. Shazeer and M. Stern, "Adafactor: Adaptive Learning Rates with Sublinear Memory Cost," in *ICML*, 2018, pp. 4603–4611.

[16] A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A. N. Gomez, S. Gouws, L. Jones, Ł. Kaiser, N. Kalchbrenner, N. Parmar, R. Sepassi, N. Shazeer, and J. Uszkoreit, "Tensor2Tensor for Neural Machine Translation," in *AMTA*, 2018, pp. 193–199.

[17] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "fairseq: A Fast, Extensible Toolkit for Sequence Modeling," in *NAACL-HLT (Demonstrations)*, 2019, pp. 48–53.

[18] F. Hieber, T. Domhan, M. Denkowski, D. Vilar, A. Sokolov, A. Clifton, and M. Post, "The Sockeye neural machine translation toolkit," in *AMTA*, 2018, pp. 200–207.

[19] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010, pp. 249–256.

[20] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *ICML*, 2015, pp. 448–456.

[21] W. Kool, H. Van Hoof, and M. Welling, "Attention, Learn to Solve Routing Problems!" in *ICLR*, 2019.

[22] C. Luo, J. Zhan, L. Wang, and Q. Yang, "Cosine Normalization: Using Cosine Similarity Instead of Dot Product in Neural Networks," in *ICANN*, 2018, pp. 382–391.

[23] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR*, 2015.

[24] R. Aharoni, M. Johnson, and O. Firat, "Massively Multilingual Neural Machine Translation," in *NAACL-HLT*, 2019, pp. 3874–3884.

[25] L. Dong, S. Xu, and B. Xu, "Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition," in *ICASSP*, 2018, pp. 5884–5888.

[26] J. Salazar, K. Kirchhoff, and Z. Huang, "Self-attention Networks for Connectionist Temporal Classification in Speech Recognition," in *ICASSP*, 2019, pp. 7115–7119.

[27] X. Wang, H. Pham, Z. Dai, and G. Neubig, "Switchout: an efficient data augmentation algorithm for neural machine translation," in *EMNLP*, 2018, pp. 856–861.

[28] G. Neubig and J. Hu, "Rapid Adaptation of Neural Machine Translation to New Languages," in *EMNLP*, 2018, pp. 875–880.

[29] P. Koehn, "Statistical significance tests for machine translation evaluation," in *EMNLP*, 2004, pp. 388–395.

[30] B. Zhang and R. Sennrich, "Root Mean Square Layer Normalization," *NeurIPS*, 2019.

[31] A. Veit, M. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," *NeurIPS*, pp. 550–558, 2016.

[32] D. Liang, Z. Huang, and Z. C. Lipton, "Learning noise-invariant representations for robust speech recognition," in *SLT*, 2018, pp. 56–63.

[33] R. Müller, S. Kornblith, and G. Hinton, "When Does Label Smoothing Help?" in *NeurIPS*, 2019.

[34] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," *CoRR*, vol. abs/1908.03265, 2019.

[35] H. Zhang, Y. N. Dauphin, and T. Ma, "Residual learning without normalization via better initialization," in *ICLR*, 2019.

[36] R. Sennrich, B. Haddow, and A. Birch, "Edinburgh Neural Machine Translation Systems for WMT 16," in *WMT*, 2016, pp. 371–376.

[37] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *CVPR*, 2016, pp. 2818–2826.

[38] G. Pereyra, G. Tucker, J. Chorowski, L. Kaiser, and G. E. Hinton, "Regularizing neural networks by penalizing confident output distributions," in *ICLR (Workshop)*, 2017.

[39] O. Press and L. Wolf, "Using the Output Embedding to Improve Language Models," in *EACL*, 2017, pp. 157–163.

[40] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *ICML*, 2013, pp. 1310–1318.

[41] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *ACL*, 2002, pp. 311–318.

A. Training details

Data and preprocessing. The pairs are English (en) to Hebrew (he), Vietnamese (vi), and Galician (gl), Slovak (sk), Arabic (ar) to English (en). Because the data is already preprocessed, we only apply BPE [36] with `fastBPE`[1]. Depending on the data size, we use different numbers of BPE operations.

We wanted to compare with the latest low-resource works of [28, 24] on the TED Talks corpus [10]. In particular, [24] identified 4 very low-resource pairs (<70k); we took the two (gl→en, sk→en) that were not extremely low (≤6k). They then identified 4 low-resource pairs with 100k-300k examples; we took the top two (ar→en, en→he). To introduce a second English-source pair and to showcase on a well-understood task, we used the *en→vi* pair from IWSLT'15 with an in-between number of examples (133k). In this way, we have examples of different resource levels, language families, writing directions, and English-source versus -target.

Model configuration. We set the hidden dimension of the feedforward sublayer to 2048 and the rest to 512, matching [1]. We use the same dropout rate for output of sublayers, ReLU, and attention weights. Additionally, we also do word-dropout [36] with probability 0.1. However, instead of zeroing the word embeddings, we randomly replace tokens with `UNK`. For all experiments, we use label smoothing of 0.1 [37, 38]. The source and target's input and output embeddings are shared [39], but we mask out words that are not in the target's vocabulary at the final output layer before softmax, by setting their logits to $-\infty$.

Training. We use a batch size of 4096 and optimize using Adam [23] with the default parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. Gradients are clipped when global norm exceeds 1.0 [40]. An epoch is a predefined number of iterations for each pair. We stop training when a maximum number of epochs has been met or the learning rate becomes too small (10^{-6}). We also do early stopping when the development BLEU has not improved for 20 evaluations. For gl→en, this number is 50. When doing validation-based decay, we use $\alpha_{decay} = 0.8$ and $patience = 3$. For complete data and model statistics, please refer to Table 1. The best checkpoint is selected based on the development BLEU score during training.

Evaluation. We report tokenized BLEU [41] with `multi-bleu.perl` to be comparable with previous works. We also measure statistical significance using bootstrap resampling [29]. For WMT'14 English-German, note that one needs to put compounds in ATAT format[2] before calculating BLEU score to be comparable with previous works.

B. Further analysis

We ask if improvements from ScaleNorm on our low-resource tasks are due to improved regularization (a smaller

[1] `https://github.com/glample/fastBPE`

[2] `https://github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/utils/get_ende_bleu.sh`

	LayerNorm		ScaleNorm	
	train	**test**	**train**	**test**
gl→en	11.792	54.300	10.151	45.770
sk→en	14.078	20.460	14.004	19.080
en→vi	15.961	17.950	16.719	17.100
en→he	15.562	14.950	15.906	15.080
ar→en	14.372	13.450	14.165	13.290

Table 8: Label-smoothed train/test perplexities when using LayerNorm and ScaleNorm.

generalization gap) or improved overall performance. We record smoothed train and test perplexities of our PreNorm models in Table 8. We see suggestive results but no conclusive trends. For ar→en, gl→en, and sk→en, train and test drop slightly, with test more so than train. For *en→vi*, train perplexity increases and test perplexity decreases an equivalent amount. For en→he, our smallest change between ScaleNorm and LayerNorm, train perplexity negligibly increased and test perplexity remains the same.

C. Listings

ScaleNorm

```
class ScaleNorm(nn.Module):
  """ScaleNorm"""
  def __init__(self, scale, eps=1e-5):
      super(ScaleNorm, self).__init__()
      self.scale = Parameter(torch.
          ↪ tensor(scale))
      self.eps = eps

  def forward(self, x):
      norm = self.scale / torch.norm(x,
          ↪ dim=-1, keepdim=True).clamp(
          ↪ min=self.eps)
      return x * norm
```

fairseq We follow fairseq's tutorial[3] and train a PostNorm Transformer base model using the following configuration:

```
fairseq-train \
data-bin/wmt16_en_de_bpe32k/ \
--arch transformer_wmt_en_de \
--share-all-embeddings \
--optimizer adam \
--adam-betas '(0.9, 0.98)' \
--clip-norm 1.0 \
--lr 0.001 \
--lr-scheduler inverse_sqrt \
--warmup-updates 4000 \
--warmup-init-lr 1e-07 \
--dropout 0.1 \
--weight-decay 0.0 \
--criterion
    ↪ label_smoothed_cross_entropy \
--label-smoothing 0.1 \
--max-tokens 8192 \
--update-freq 10 \
--attention-dropout 0.1 \
--activation-dropout 0.1 \
--max-epoch 40
```

For PreNorm, simply include the flags `--encoder-normalize-before --decoder-normalize-before`.

For ScaleNorm, we replace all LayerNorms in `fairseq/models/transformer.py` and `fairseq/modules/transformer_layer.py` with ScaleNorm (implemented above). For FixNorm, we change the word embedding initialization to uniform with range $[-0.01, 0.01]$ and normalize with `torch.nn.functional.normalize`.

We note that fairseq uses Xavier uniform initialization, which is big compared to our SmallInit (Section 3.1). We conjecture that fairseq training remains stable thanks to its large batch size, which gives more stable gradients.

[3] https://github.com/pytorch/fairseq/blob/master/examples/scaling_nmt/README.md