PowerNorm: Rethinking Batch Normalization in Transformers

Sheng Shen [*1] **Zhewei Yao** [*1] **Amir Gholami** [1] **Michael W. Mahoney** [1] **Kurt Keutzer** [1]

Abstract

The standard normalization method for neural network (NN) models used in Natural Language Processing (NLP) is layer normalization (LN). This is different than batch normalization (BN), which is widely-adopted in Computer Vision. The preferred use of LN in NLP is principally due to the empirical observation that a (naive/vanilla) use of BN leads to significant performance degradation for NLP tasks; however, a thorough understanding of the underlying reasons for this is not always evident. In this paper, we perform a systematic study of NLP transformer models to understand why BN has a poor performance, as compared to LN. We find that the statistics of NLP data across the batch dimension exhibit large fluctuations throughout training. This results in instability, if BN is naively implemented. To address this, we propose Power Normalization (PN), a novel normalization scheme that resolves this issue by (i) relaxing zero-mean normalization in BN, (ii) incorporating a running quadratic mean instead of per batch statistics to stabilize fluctuations, and (iii) using an approximate backpropagation for incorporating the running statistics in the forward pass. We show theoretically, under mild assumptions, that PN leads to a smaller Lipschitz constant for the loss, compared with BN. Furthermore, we prove that the approximate backpropagation scheme leads to bounded gradients. We extensively test PN for transformers on a range of NLP tasks, and we show that it significantly outperforms both LN and BN. In particular, PN outperforms LN by 0.4/0.6 BLEU on IWSLT14/WMT14 and 5.6/3.0 PPL on PTB/WikiText-103. We make our code publicly available at https://github.com/sIncerass/powernorm.

[*]Equal contribution [1]UC Berkeley. Correspondence to: Amir Gholami <amirgh@berkeley.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

1. Introduction

Normalization has become one of the critical components in Neural Network (NN) architectures for various machine learning tasks, in particular in Computer Vision (CV) and Natural Language Processing (NLP). However, currently there are very different forms of normalization used in CV and NLP. For example, Batch Normalization (BN) (Ioffe & Szegedy, 2015) is widely adopted in CV, but it leads to significant performance degradation when naively used in NLP. Instead, Layer Normalization (LN) (Ba et al., 2016) is the standard normalization scheme used in NLP. All recent NLP architectures, including Transformers (Vaswani et al., 2017), have incorporated LN instead of BN as their default normalization scheme. In spite of this, the reasons why BN fails for NLP have not been clarified, and a better alternative to LN has not been presented.

In this work, we perform a systematic study of the challenges associated with BN for NLP, and based on this we propose Power Normalization (PN), a novel normalization method that significantly outperforms LN. In particular, our contributions are as follows:

- We find that there are clear differences in the batch statistics of NLP data versus CV data. In particular, we observe that batch statistics for NLP data have a very large variance throughout training. This variance exists in the corresponding gradients as well. In contrast, CV data exhibits orders of magnitude smaller variance. See Figure 2 and 3 for a comparison of BN in CV and NLP.
- To reduce the variation of batch statistics, we modify typical BN by relaxing zero-mean normalization, and we replace the variance with the quadratic mean. We denote this scheme as PN-V. We show theoretically that PN-V preserves the first-order smoothness property as in BN; see Lemma 2.
- We show that using running statistics for the quadratic mean results in significantly better performance, up to 1.5/2.0 BLEU on IWSLT14/WMT14 and 7.7/3.4 PPL on PTB/WikiText-103, as compared to BN; see Table 1 and 2. We denote this scheme as PN. Using running statistics requires correcting the typical backpropagation scheme in BN. As an alternative, we propose an

approximate backpropagation to capture the running statistics. We show theoretically that this approximate backpropagation leads to bounded gradients, which is a necessary condition for convergence; see Theorem 4.

- We perform extensive tests showing that PN also improves performance on machine translation and language modeling tasks, as compared to LN. In particular, PN outperforms LN by 0.4/0.6 BLEU on IWSLT14/WMT14, and by 5.6/3.0 PPL on PTB/WikiText-103. We emphasize that the improvement of PN over LN is without any change of hyperparameters.

- We analyze the behaviour of PN and LN by computing the Singular Value Decomposition of the resulting embedding layers, and we show that PN leads to a more well-conditioned embedding layer; see Figure 6. Furthermore, we show that PN is robust to small-batch statistics, and it still achieves higher performance, as opposed to LN; see Figure 5.

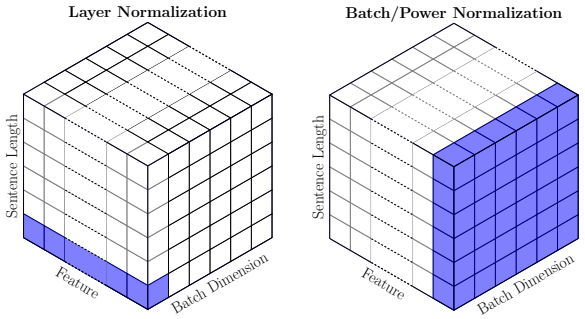

Figure 1. The illustration of layer normalization (left) and batch/power normalization (right). The entries colored in blue show the components used for calculating the statistics.

2. Related Work

Normalization is widely used in modern deep NNs such as ResNet (He et al., 2016), MobileNet-V2 (Sandler et al., 2018), and DenseNet (Huang et al., 2017) in CV, as well as LSTMs (Hochreiter & Schmidhuber, 1997; Ba et al., 2016), transformers (Vaswani et al., 2017), and transformer-based models (Devlin et al., 2019; Liu et al., 2019) in NLP. There are two main categories of normalization: weight normalization (Salimans & Kingma, 2016; Miyato et al., 2018; Qiao et al., 2019) and activation normalization (Ioffe & Szegedy, 2015; Jarrett et al., 2009; Krizhevsky et al., 2012; Ba et al., 2016; Ulyanov et al., 2016; Wu & He, 2018; Li et al., 2019). Here, we solely focus on the latter, and we briefly review related work in CV and NLP.

Normalization in Computer Vision Batch Normalization (BN) (Ioffe & Szegedy, 2015) has become the de-facto normalization for NNs used in CV. BN normalizes the activations (feature maps) by computing channel-wise mean and variance across the batch dimension, as schematically shown in Figure 1. It has been found that BN leads to robustness with respect to sub-optimal hyperparameters (e.g., learning rate) and initialization, and it generally results in more stable training for CV tasks (Ioffe & Szegedy, 2015). Following the seminal work of (Ioffe & Szegedy, 2015), there have been two principal lines of research: (i) extensions/modifications of BN to improve its performance, and (ii) theoretical/empirical studies to understand why BN helps training.

With regard to (i), it was found that BN does not perform well for problems that need to be trained with small batches, e.g., image segmentation (often due to memory limits) (Zagoruyko & Komodakis, 2016; Lin et al., 2017; Goldberger et al., 2005). The work of (Ioffe, 2017) proposed batch renormalization to remove/reduce the dependence of batch statistics to batch size. It was shown that this approach leads to improved performance for small batch training as well as cases with non-i.i.d. data. Along this direction, the work of (Singh & Shrivastava, 2019) proposed "EvalNorm," which uses corrected normalization statistics. Furthermore, the recent work of (Yan et al., 2020) proposed "Moving Average Batch Normalization (MABN)" for small batch BN by replacing batch statistics with moving averages.

There has also been work on alternative normalization techniques, and in particular Layer Normalization (LN), proposed by (Ba et al., 2016). LN normalizes across the channel/feature dimension as shown in Figure 1. This could be extended to Group Norm (GN) (Wu & He, 2018), where the normalization is performed across a partition of the features/channels with different pre-defined groups. Instance Normalization (IN) (Ulyanov et al., 2016) is another technique, where per-channel statistics are computed for each sample.

With regard to (ii), there have been several studies to understand why BN helps training in CV. The original motivation was that BN reduces the so-called "Internal Covariance Shift" (ICS) (Ioffe & Szegedy, 2015). However, this explanation was viewed as incorrect/incomplete (Rahimi, 2017). In particular, the recent study of (Santurkar et al., 2018) argued that the underlying reason that BN helps training is that it results in a smoother loss landscape. This was later confirmed for deep NN models by measuring the Hessian spectrum of the network with/without BN (Yao et al., 2019).

Normalization in Natural Language Processing Despite the great success of BN in CV, the large computation and storage overhead of BN at each time-step in recurrent

neural networks (RNNs) made it impossible/expensive to deploy for NLP tasks (Cooijmans et al., 2017). To address this, the work of (Cooijmans et al., 2017; Hou et al., 2019) used shared BN statistics across different time steps of RNNs. However, it was found that the performance of BN is significantly lower than LN for NLP. For this reason, LN became the default normalization technique, even for the recent transformer models introduced by (Vaswani et al., 2017).

Only limited recent attempts were made to compare LN with other alternatives or investigate the reasons behind the success of LN in transformer models. For instance, (Zhang & Sennrich, 2019) proposes RMSNorm, which removes the re-centering invariance in LN and performs re-scaling invariance with the root mean square summed of the inputs. They showed that this approach achieves similar performance to LN, but with smaller (89% to 64%) overhead. Furthermore, (Nguyen & Salazar, 2019) studies different variants of weight normalization for transformers in low-resource machine translation. The recent work of (Xu et al., 2019) studies why LN helps training, and in particular it finds that the derivatives of LN help recenter and rescale backward gradients. From a different angle, (Zhang et al., 2019b;a) try to ascribe the benefits of LN to solving the exploding and vanishing gradient problem at the beginning of training. They also propose two properly designed initialization schemes which also enjoy that property and are able to stabilize training for transformers.

However, most of these approaches achieve similar or marginal improvement over LN. More importantly, there is still not a proper understanding of why BN performs poorly for transformers applied to NLP data. Here, we address this by systematically studying the BN behavior throughout training; and, based on our results, we propose Power Normalization (PN), a new normalization method that significantly outperforms LN for a wide range of tasks in NLP.

3. Batch Normalization

Notation. We denote the input of a normalization layer as $\boldsymbol{X} \in \mathbb{R}^{B\times d}$, where d is the embedding/feature size and B is the batch size[1]. We denote $\mathcal{L}$ as the final loss of the NN. The i-th row (column) of a matrix, e.g., $\boldsymbol{X}$, is denoted by $\boldsymbol{X}_{i,:}$ ($\boldsymbol{X}_{:,i}$). We also write the i-th row of the matrix as its lower-case version, i.e., $\boldsymbol{x}_i = \boldsymbol{X}_{i,:}$. For a vector y, y_i denotes the i-th element in y.

Without other specification: (i) for two vector $\mathrm{x} \in \mathbb{R}^d$ and $\mathrm{y} \in \mathbb{R}^d$, we denote xy as the element-wise product, x + y as the element-wise sum, and $\langle \mathrm{x}, \mathrm{y} \rangle$ as the inner product; (ii) for a vector $\mathrm{y} \in \mathbb{R}^d$ and a matrix $\boldsymbol{X} \in \mathbb{R}^{B\times d}$, we denote $\mathrm{y}\odot\boldsymbol{X}$ as $[y_1\boldsymbol{X}_{:,1}, ..., y_d\boldsymbol{A}_{:,d}]$ and $\mathrm{y}+\boldsymbol{X}$ as $[\mathrm{y}+\boldsymbol{X}_{1,:}; ...; \mathrm{y}+\boldsymbol{X}_{B,:}]$; and (iii) for a vector $\mathrm{y} \in \mathbb{R}^d$, $\mathrm{y} > C$ means that each entry of y is larger than the constant C, i.e., $y_i > C$ for all i.

[1] For NLP tasks, we flatten sentences/word in one dimension, i.e., the batch size actually corresponds to all **non-padded** words in a training batch.

Algorithm 1 Batch Normalization (Every Iteration)

begin Forward Propagation:

Input: $\boldsymbol{X} \in \boldsymbol{R}^{B\times d}$

Output: $\boldsymbol{Y} \in \boldsymbol{R}^{B\times d}$

$\mu_B = \frac{1}{B}\sum_{i=1}^{B} \boldsymbol{x}_i$ // Get mini-batch mean

$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(\boldsymbol{x}_i - \mu_B)^2$ // Get mini-batch variance

$\widetilde{\boldsymbol{X}} = \frac{\boldsymbol{X}-\mu_B}{\sigma_B}$ // Normalize

$\boldsymbol{Y} = \gamma \odot \widetilde{\boldsymbol{X}} + \beta$ // Scale and shift

$\mu = \alpha\mu + (1-\alpha)\mu_B$ // Update running mean

$\sigma^2 = \alpha\sigma^2 + (1-\alpha)\sigma_B^2$ // Update running variance

begin Backward Propagation:

Input: $\frac{\partial\mathcal{L}}{\partial\boldsymbol{Y}} \in \boldsymbol{R}^{B\times d}$

Output: $\frac{\partial\mathcal{L}}{\partial\boldsymbol{X}} \in \boldsymbol{R}^{B\times d}$

$\frac{\partial\mathcal{L}}{\partial\boldsymbol{X}}$ based on Eq. 3 // Gradient of $\boldsymbol{X}$

Inference: $\boldsymbol{Y} = \gamma \odot \frac{\boldsymbol{X}-\mu}{\sigma} + \beta$

3.1. Formulation of Batch Normalization

We briefly review the formulation of BN (Ioffe & Szegedy, 2015). Let us denote the mean (variance) of $\boldsymbol{X}$ along the batch dimension as $\mu_B \in \mathbb{R}^d$ ($\sigma_B^2 \in \mathbb{R}^d$). The batch dimension is illustrated in Figure 1. The BN layer first enforces zero mean and unit variance, and it then performs an affine transformation by scaling the result by $\gamma, \beta \in \mathbb{R}^d$, as shown in Algorithm 1.

The Forward Pass (FP) of BN is performed as follows. Let us denote the intermediate result of BN with zero mean and unit variance as $\widetilde{\boldsymbol{X}}$, i.e.,

$$\widetilde{\boldsymbol{X}} = \frac{\boldsymbol{X} - \mu_B}{\sigma_B}. \tag{1}$$

The final output of BN, $\boldsymbol{Y}$, is then an affine transformation applied to $\widetilde{\boldsymbol{X}}$:

$$\boldsymbol{Y} = \gamma \odot \widetilde{\boldsymbol{X}} + \beta. \tag{2}$$

The corresponding Backward Pass (BP) can then be derived as follows. Assume that the derivative of $\mathcal{L}$ with respect to $\boldsymbol{Y}$ is given, i.e., $\frac{\partial\mathcal{L}}{\partial\boldsymbol{Y}}$ is known. Then, the derivative with respect to input can be computed as:

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{x}_i} = \frac{\gamma}{\sigma_B}\frac{\partial\mathcal{L}}{\partial\boldsymbol{y}_i} - \frac{\gamma}{\sigma_B B}\sum_{j\in B}\Big(\underbrace{\frac{\partial\mathcal{L}}{\partial\boldsymbol{y}_j}}_{\text{from } \mu_B:\, g_\mu} + \underbrace{\frac{\partial\mathcal{L}}{\partial\boldsymbol{y}_j}\check{\mathbf{x}}_j\check{\mathbf{x}}_i}_{\text{from } \sigma_B^2:\, g_{\sigma^2}}\Big). \tag{3}$$

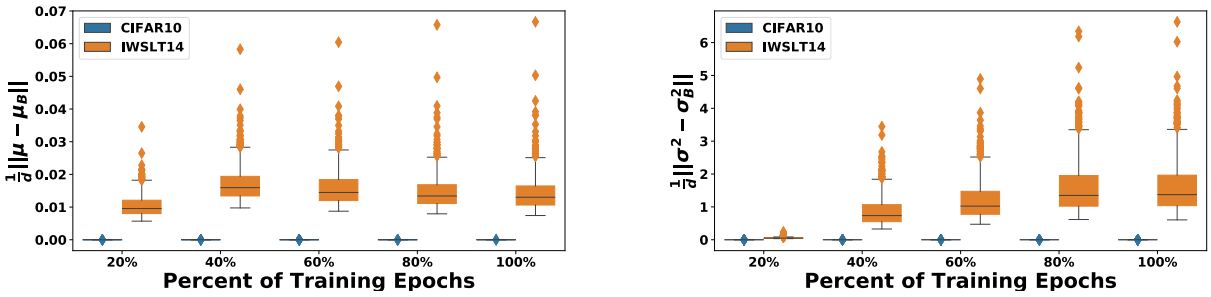

Figure 2. The average Euclidean distance between the batch statistics (μ_B, σ_B^2) and the running statistics (μ, σ^2) stored in first BN during forward pass for ResNet20 on Cifar-10 and Transformer on IWLST14. We can clearly see that the ResNet20 statistics have orders of magnitude smaller variation than the running statistics throughout training. However, the corresponding statistics in Transformer$_{\text{BN}}$ exhibit very high variance with extreme outliers. This is true both for the mean (shown in the left) as well as variance (shown in right). This is one of the contributing factors to the low performance of BN in transformers.

See Lemma 6 in Appendix C for details. We denote the terms contributed by μ_B and σ_B^2 as g_μ and g_{σ^2}, respectively.

In summary, there are four batch statistics in BN, two in FP and two in BP. The stability of training is highly dependent on these four parameters. In fact, naively implementing the BN as above for transformers leads to poor performance. For example, using transformer with BN (denoted as Transformer$_{\text{BN}}$) results in 1.1 and 1.4 lower BLEU score, as compared to the transformer with LN (Transformer$_{\text{LN}}$), on IWSLT14 and WMT14, respectively; see Table 1.

This is significant performance degradation, and it stems from instabilities associated with the above four batch statistics. To analyze this, we studied the batch statistics using the standard setting of ResNet20 on Cifar-10 and Transformer$_{\text{BN}}$ on IWSLT14 (using a standard batch size of 128 and tokens of 4K, respectively). In the first experiment, we probed the fluctuations between batch statistics, μ_B/σ_B, and the corresponding BN running statistics, μ/σ, throughout training. This is shown for the first BN layer of ResNet20 on Cifar-10 and Transformer$_{\text{BN}}$ on IWSLT14 in Figure 2. Here, the y-axis shows the average Euclidean distance between batch statistics (μ_B, σ_B) and the running statistics (μ, σ), and the x-axis is different epochs of training, where we define the average Euclidean distance as $\text{dist}(\mu_B, \mu) = \frac{1}{d}\|\mu_B - \mu\|$.

The first observation is that Transformer$_{\text{BN}}$ shows significantly larger distances between the batch statistics and the running statistics than ResNet20 on Cifar-10, which exhibits close to zero fluctuations. Importantly, this distance between σ_B and σ significantly increases throughout training, but with extreme outliers. During inference, we have to use the running statistics. However, such large fluctuations would lead to a large inconsistency between statistics of the testing data and the BN's running statistics.

The second observation comes from probing the norm of g_μ and g_{σ^2} defined in Eq. 3, which contribute to the gradient backpropagation of input. These results are shown in Figure 3, where we report the norm of these two parameters for ResNet20 and Transformer$_{\text{BN}}$. For Transformer$_{\text{BN}}$, we can see very large outliers that actually persist throughout training. This is in contrast to ResNet20, for which the outliers vanish as training proceeds.

4. Power Normalization

Based on our empirical observations, we propose Power Normalization (PN), which effectively resolves the performance degradation of BN. This is achieved by incorporating the following two changes to BN. First, instead of enforcing unit variance, we enforce unit quadratic mean for the activations. The reason for this is that we find that enforcing zero-mean and unit variance in BN is detrimental due to the large variations in the mean, as discussed in the previous section. However, we observe that unlike mean/variance, the unit quadratic mean is significantly more stable for transformers. Second, we incorporate running statistics for the quadratic mean of the signal, and we incorporate an approximate backpropagation method to compute the corresponding gradient. We find that the combination of these two changes leads to a significantly more effective normalization, with results that exceed LN, even when the same training hyperparameters are used. Below we discuss each of these two components.

4.1. Relaxing Zero-Mean and Enforcing Quadratic Mean

Here, we describe the first modification in PN. As shown in Figure 2 and 3, μ_B and g_μ exhibit significant number of large outliers, which leads to inconsistencies between training and inference statistics. We first address this by relaxing

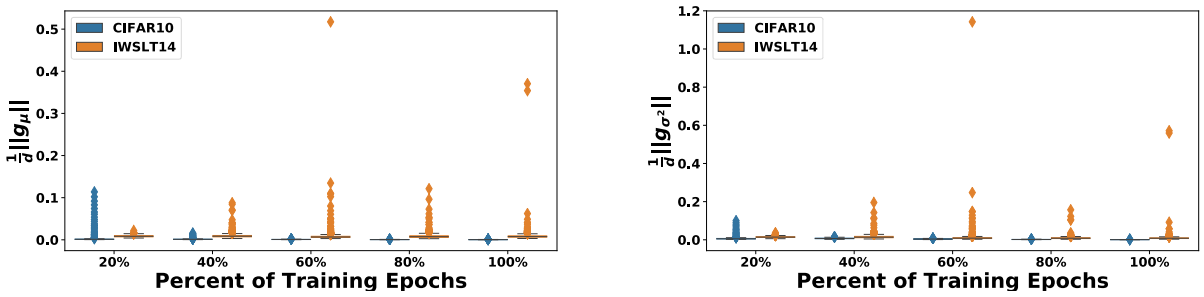

Figure 3. The average gradient norm of the input of the first BN layer contributed by μ_B and σ_B for ResNet20 on Cifar10 and Transformer$_{\text{BN}}$ on IWSLT14 during the BP (note that $d = 16$ for Cifar-10 and $d = 512$ for IWSLT experiment). It can be clearly seen that the norm of g_μ and g_{σ^2} for ResNet20 has orders of magnitude smaller variation throughout training, as compared to that for Transformer$_{\text{BN}}$. Also, the outliers for ResNet20 vanish at the end of training, which is in contrast to Transformer$_{\text{BN}}$, for which the outliers persist. This is true both for g_μ (shown in left) as well as g_{σ^2} (shown in right).

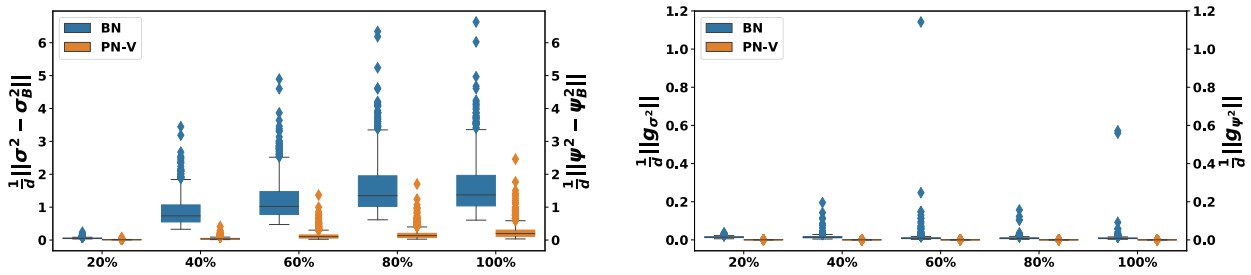

Figure 4. Results for Transformer on IWSLT14. (Left) The average Euclidean distance between batch statistics (σ_B, ψ_B) and the running statistics (σ, ψ) stored in first BN/PN-V during forward propagation (FP). (Right) The average norm of gradient of the input of the first BN/PN-V contributed by σ_B/ψ_B. During FP, ψ_B has much smaller variations of running statistics, as compared to σ_B, as shown in left, It can also be clearly seen that during BP, the norm of g_{ψ^2} exhibits many fewer outliers, as compared to g_{σ^2}, throughout the training.

the zero-mean normalization, and we use the quadratic mean of the signal, instead of its variance. The quadratic mean exhibits orders of magnitude smaller fluctuations, as shown in Figure 4. We refer to this normalization (i.e., no zero mean and unit quadratic mean enforcement) as PN-V, defined as follows.

Definition 1 (PN-V)**.** *Let us denote the quadratic mean of the batch as* ${\psi_B}^2 = \frac{1}{B}\sum_{i=1}^{B} \boldsymbol{x}_i^2$*. Furthermore, denote* $\widehat{\boldsymbol{X}}$ *as the signal scaled by* ψ_B*, i.e.,*

$$\widehat{\boldsymbol{X}} = \frac{\boldsymbol{X}}{\psi_B}. \tag{4}$$

Then, the output of PN-V *is defined as*

$$\boldsymbol{Y} = \gamma \odot \widehat{\boldsymbol{X}} + \beta, \tag{5}$$

where $\gamma \in \mathbb{R}^d$ *and* $\beta \in \mathbb{R}^d$ *are two parameters (vectors) in* PN-V *(which is the same as in the affine transformation used in BN).*

Note that here we use the same notation $\boldsymbol{Y}$ *as the output in Eq. 2 without confusion.*

The corresponding BP of PN-V is as follows:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_i} = \frac{\gamma}{\psi_B}\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_i} - \frac{\gamma}{B\psi_B}\sum_{j\in B}\underbrace{\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_j}}_{\text{from } {\psi_B}^2:\, g_{\psi^2}}\hat{\mathbf{x}}_j\hat{\mathbf{x}}_i \quad . \tag{6}$$

See Lemma 8 in Appendix C for the full details. Here, g_{ψ^2} is the gradient attributed by ${\psi_B}^2$. Note that, compared to BN, there exist only two batch statistics in FP and BP: ${\psi_B}^2$ and g_{ψ^2}. This modification removes the two unstable factors corresponding to μ_B and σ_B in BN (g_μ, and g_{σ^2} in Eq. 3). This modification also results in significant performance improvement, as reported in Table 1 for IWSLT14 and WMT14. By directly replacing BN with PN-V (denoted as Transformer$_{\text{PN-V}}$), the BLEU score increases from 34.4 to 35.4 on IWSLT14, and 28.1 to 28.5 on WMT14. These improvements are significant for these two tasks. For example, (Zhang et al., 2019b;a) only improves the BLEU score by 0.1 on IWSLT14.

As mentioned before, ψ_B exhibits orders of magnitude smaller variations, as compared to σ_B. This is shown in Fig-

ure 4, where we report the distance between the running statistics for σ, dist(σ_B^2, σ^2), and ψ, dist$({\psi_B}^2, \psi^2)$. Similarly during BP, we compute the norm of g_{σ^2} and g_{ψ^2}, and we report it in Figure 4 throughout training. It can be clearly seen that during BP, the norm of g_{ψ^2} exhibits many fewer outliers as compared to g_{σ^2}.

In (Santurkar et al., 2018), the authors provided theoretical results suggesting that employing BN in DNNs can lead to a smaller Lipschitz constant of the loss.

It can be shown that PN-V also exhibits similar behaviour, under mild assumptions. In more detail, let us denote the loss of the NN without normalization as $\widehat{\mathcal{L}}$. With mild assumptions, (Santurkar et al., 2018) shows that the norm of $\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}}$ (with BN) is smaller than the norm of $\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{X}}$. Here, we show that, under the same assumptions, PN-V can achieve the same results that BN does. See Appendix C for details, including the statement of Assumption 9.

Lemma 2 (The effect of PN-V on the Lipschitz constant of the loss)**.** *Under Assumption 9, we have*

$$\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}_{:,i}}\|^2 = \frac{\gamma_i^2}{(\psi_B)_i^2}\left(\|\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{X}_{:,i}}\|^2 - \langle \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{X}_{:,i}}, \frac{\widehat{\boldsymbol{X}}_{:,i}}{\sqrt{B}}\rangle^2\right). \quad (7)$$

See the proof in Appendix C. Note that $\langle \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{X}_{:,i}}, \frac{\widehat{\boldsymbol{X}}_{:,i}}{\sqrt{B}}\rangle^2$ is non-negative, and hence the Lipschitz constant of $\mathcal{L}$ is smaller than that of $\widehat{\mathcal{L}}$ if $\gamma_i \leqslant (\psi_B)_i$. This is what we observe in practice, as shown in Appendix B.

4.2. Running Statistics in Training

Here, we discuss the second modification in PN. First note that even though Transformer$_{\text{PN-V}}$ outperforms Transformer$_{\text{BN}}$, it still can not match the performance of LN. This could be related to the larger number of outliers present in ψ_B, as shown in Figure 4. A straightforward solution to address this is to use running statistics for the quadratic mean (denoted as ψ^2), instead of using per batch statistics, since the latter changes in each iteration. However, using running statistics requires modification of the backpropagation, which we described below.

Definition 3 (PN)**.** *Denote the inputs/statistics at the t-th iteration by $\cdot^{(t)}$, e.g., $\boldsymbol{X}^{(t)}$ is the input data at t-th iteration. In the forward propagation, the following equations are used for the calculation:*

$$\widehat{\boldsymbol{X}}^{(t)} = \frac{\boldsymbol{X}^{(t)}}{\psi^{(t-1)}}, \quad (8)$$

$$\boldsymbol{Y}^{(t)} = \gamma \odot \widehat{\boldsymbol{X}}^{(t)} + \beta, \quad (9)$$

$$(\psi^{(t)})^2 = (\psi^{(t-1)})^2 + (1-\alpha)({\psi_B}^2 - (\psi^{(t-1)})^2). \quad (10)$$

Here, $0 < \alpha < 1$ is the moving average coefficient in the forward propagation, and ψ_B is the statistic for the current batch. Since the forward pass evolves running statistics, the backward propagation cannot be accurately computed—namely, the accurate gradient calculation needs to track back to the first iteration. Here, we propose to use the following approximated gradient in backward propagation:

$$(\widetilde{\boldsymbol{X}}^{(t)})' = \frac{\partial \mathcal{L}}{\partial \widehat{\boldsymbol{X}}^{(t)}} - \nu^{(t-1)} \odot \widehat{\boldsymbol{X}}^{(t)}, \quad (11)$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}^{(t)})} = \frac{(\widetilde{\boldsymbol{X}}^{(t)})'}{\psi^{(t-1)}}, \quad (12)$$

$$\nu^{(t)} = \nu^{(t-1)}(1-(1-\alpha)\Gamma^{(t)}) + (1-\alpha)\Lambda^{(t)}, \quad (13)$$

where $\Gamma^{(t)} = \frac{1}{B}\sum_{i=1}^{B} \hat{\mathbf{x}}_i^{(t)} \hat{\mathbf{x}}_i^{(t)}$ *and* $\Lambda^{(t)} = \frac{1}{B}\sum_{i=1}^{B} \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{x}}_i^{(t)}} \hat{\mathbf{x}}_i^{(t)}$.

This backpropagation essentially uses running statistics by computing the gradient of the loss w.r.t. the quadratic mean of the current batch, rather than using the computationally infeasible method of computing directly the gradient w.r.t. running statistics of the quadratic mean. Importantly, this formulation leads to bounded gradients which is necessary for convergence, as shown below.

Theorem 4 (Gradient of $\mathcal{L}$ w.r.t. $\boldsymbol{X}$ is bounded in PN)**.** *For any datum point of $\widetilde{\boldsymbol{X}}$ (i.e. $\widetilde{\boldsymbol{X}}_{i,:}$), the gradients computed from Eq. 11 are bounded by a constant.*

Furthermore, the gradient of $\boldsymbol{X}_{i,:}$ is also bounded, as given Eq. 12.

See the proof in Appendix C. The pseudo-code for PN algorithm is presented in Algorithm 2.

Algorithm 2 Power Normalization (Every Iteration)

begin Forward Propagation:

Input: $\boldsymbol{X} \in \boldsymbol{R}^{B \times d}$

Output: $\boldsymbol{Y} \in \boldsymbol{R}^{B \times d}$

${\psi_B}^2 = \frac{1}{B}\sum_{i=1}^{B} \boldsymbol{x}_i^2$ // Get mini-batch statistics

$\widehat{\boldsymbol{X}} = \frac{\boldsymbol{X}}{\psi}$ // Normalize

$\boldsymbol{Y} = \gamma \odot \widehat{\boldsymbol{X}} + \beta$ // Scale and shift

$\psi^2 = \alpha\psi^2 + (1-\alpha){\psi_B}^2$ // Update running statistics

begin Backward Propagation:

Input: $\frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}} \in \boldsymbol{R}^{B \times d}$

Output: $\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}} \in \boldsymbol{R}^{B \times d}$

$\frac{\partial \mathcal{L}}{\partial \widehat{\boldsymbol{X}}} = \gamma \odot \frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}}$ // Intermediate Gradient

$\widehat{\boldsymbol{X}}' = \frac{\partial \mathcal{L}}{\partial \widehat{\boldsymbol{X}}} - \nu\widehat{\boldsymbol{X}}$ // Intermediate Estimated Gradient

$\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}} = \frac{\widehat{\boldsymbol{X}}'}{\psi}$ // Gradient of $\boldsymbol{X}$

$\nu = \nu(1-(1-\alpha)\Gamma) + (1-\alpha)\Lambda$ // See Definition 3 for Γ and Λ

Inference: $\boldsymbol{Y} = \gamma \odot \frac{\boldsymbol{X}}{\psi} + \beta$

5. Results

5.1. Experiment Setup

We compare our PN method with LN and BN for a variety of sequence modeling tasks: Neural Machine Translation (MT); and Language Modeling (LM). We implement our code for MT using *fairseq-py* (Ott et al., 2019), and (Ma et al., 2019) for LM tasks. For a fair comparison, we directly replace the LN in transformers (Transformer$_{\text{LN}}$) with BN (Transformer$_{\text{BN}}$) or PN (Transformer$_{\text{PN}}$) without varying the position of each normalization layer or changing the training hyperparameters.

For all the experiments, we use the pre-normalization setting in (Wang et al., 2019), where the normalization layer is located right before the multi-head attention module and point-wise feed-forward network module. Following (Wang et al., 2019), we generally increase the learning rate by a factor of 2.0, relative to the common post-normalization transformer (Vaswani et al., 2017). Below we discuss tasks specific settings. [2]

Neural Machine Translation We evaluate our methods on two widely used public datasets: IWSLT14 German-to-English (De-En) and WMT14 English-to-German (En-De) dataset. We follow the settings reported in (Ott et al., 2018). We use transformer `big` architecture for WMT14 (4.5M sentence pairs) and `small` architecture for IWSLT14 (0.16M sentence pairs). For inference, we average the last 10 checkpoints, and we set the length penalty to 0.6/1.0, and the beam size to 4/5 for WMT/IWSLT, following (Ott et al., 2019). All the other hyperparamters (learning rate, dropout, weight decay, warmup steps, etc.) are set identically to the ones reported in the literature for LN (i.e., we use the same hyperparameters for BN/PN).

Language Modeling We experiment on both PTB (Mikolov et al., 2011) and Wikitext-103 (Merity et al., 2017), which contain 0.93M and 100M tokens, respectively. We use three layers tensorized transformer core-1 for PTB and six layers tensorized transformer core-1 for Wikitext-103, following (Ma et al., 2019). Furthermore, we apply the multi-linear attention mechanism with masking, and we report the final testing set perplexity (PPL).[3]

5.2. Experiment Results

Neural Machine Translation We use BLEU (Papineni et al., 2002) as the evaluation metric for MT. Following standard practice, we measure tokenized case-sensitive BLEU and case-insensitive BLEU for WMT14 En-De and IWSLT14 De-En, respectively. For a fair comparison, we do not include other external datasets. All the transformers in Table 1 are using six encoder layers and six decoder layers.

Model	IWSLT14 `small`	WMT14 `big`
Transformer (Vaswani et al., 2017)	34.4	28.4
DS-Init (Zhang et al., 2019a)	34.4	29.1
Fixup-Init (Zhang et al., 2019b)	34.5	29.3
Scaling NMT (Ott et al., 2018)	/	29.3
Dynamic Conv (Wu et al., 2019)	35.2	29.7
Transformer + LayerDrop (Fan et al., 2020)	/	29.6
Pre-Norm Transformer$_{\text{LN}}$	35.5	29.5
Pre-Norm Transformer$_{\text{BN}}$	34.4	28.1
Pre-Norm Transformer$_{\text{PN-V}}$	35.5	28.5
Pre-Norm Transformer$_{\text{PN}}$	**35.9**	**30.1**

Table 1. MT performance (BLEU) on IWSLT14 De-En and WMT14 En-De testsets. Using PN-V instead of BN significantly improves the performance, but LN still outperforms. However, PN achieves much higher BLEU scores, as compared to LN.

The results are reported in Table 1. In the first section of rows, we report state-of-the-art results for these two tasks with comparable model sizes. In the second section of rows, we report the results with different types of normalization. Notice the significant drop in BLEU score when BN is used (34.4/28.1), as opposed to LN (35.5/29.5). Using PN-V instead of BN helps reduce this gap, but LN still outperforms. However, the results corresponding to PN exceeds LN results by more than 0.4/0.6 points, This is significant for these tasks. Comparing with other concurrent works like DS-Init and Fixup-Init (Zhang et al., 2019a;b), the improvements in Transformer$_{\text{PN}}$ are still significant.

Model	PTB Test PPL	WikiText-103 Test PPL
Tied-LSTM (Inan et al., 2017)	48.7	48.7
AWD-LSTM-MoS (Yang et al., 2018)	56.0	29.2
Adaptive Input (Baevski & Auli, 2019)	57.0	20.5
Transformer-XL$_{\texttt{base}}$ (Dai et al., 2019)	54.5	24.0
Transformer-XL$_{\texttt{large}}$ (Dai et al., 2019)	–	18.3
Tensor-Transformer$_{1\text{core}}$ (Ma et al., 2019)	57.9	20.9
Tensor-Transformer$_{2\text{core}}$ (Ma et al., 2019)	49.8	18.9
Tensor-Transformer$_{1\text{core}}$ + LN	53.2*	20.9*
Tensor-Transformer$_{1\text{core}}$ + BN	60.7	27.2
Tensor-Transformer$_{1\text{core}}$ + PN-V	55.3	21.3
Tensor-Transformer$_{1\text{core}}$ + PN	**47.6**	**17.9**

Table 2. Results with state-of-the-art methods on PTB and WikiText-103. '-' indicates no reported results in that setting, '$*$' indicates the results are from our own implementation. PN achieves 5.6/3 points lower testing PPL on PTB and WikiTest-103, respectively, compared to LN.

[2] More detailed experimental settings and comparisons between normalization methods are provided in Appendix A, B.3.

[3] We also report the validation perplexity in Appendix B.

Language Modeling We report the LM results in Table 2, using the tensorized transformer proposed in (Ma et al., 2019). Here we observe a similar trend. Using BN results in a significant degradation, increasing testing PPL by more than 7.5/6.3 for PTB/WikiText-103 datasets (achieving 60.7/27.2 as opposed to 53.2/20.9). However, when we incorporate the PN normalization, we achieve state-of-the-art results for these two tasks (for these model sizes and without any pre-training on other datasets). In particular, PN results in 5.6/3 points lower testing PPL, as compared to LN. Importantly, note that using PN we achieve better results than (Ma et al., 2019), with the same number of parameters.

5.3. Analysis

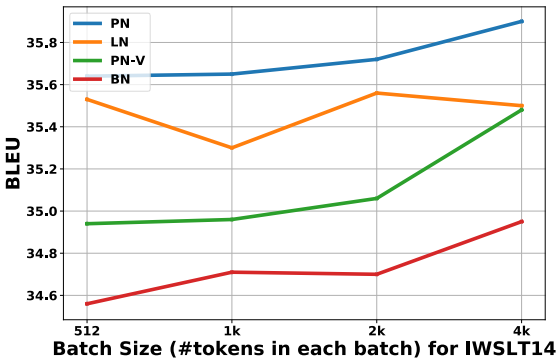

Figure 5. Ablation study of the performance of PN, PN-V, LN and BN on IWSLT14 trained using different batch sizes. Note that the performance of PN consistently outperforms LN. In the meanwhile, PN-V can only match the result of LN when mini-batch gets to 4K. Among all the settings, BN behaves poorly and abnormally across different mini-batches.

The Effect of Batch Size for Different Normalization To understand better the effects of our proposed methods PN and PN-V, we change the batch size used to collect statistics in BN, LN, and PN. To this end, we keep the total batch size constant at 4K tokens, and we vary the mini-batch size used to collect statistics from 512 to 4K. Importantly, note that we keep the total batch size constant at 4K, and we use gradient accumulation for smaller mini-batches. For example, for the case with mini-batch of 512, we use eight gradient accumulations. The results are reported in Figure 5. We can observe that BN behaves poorly and abnormally across different mini-batches. Noticeably, after relaxing the zero-mean normalization in BN and replacing the variance estimation with quadratic mean, PN-V matches the performance of LN for 4K mini-batch and consistently outperforms BN. However, it underperforms LN. In contrast, we can see that PN consistently achieves higher results under different mini-batch settings.

Representation Power of learned Embedding To investigate further the performance gain of PN, we compute the Singular Value Decomposition of the embedding layers, as proposed by (Gao et al., 2019), which argued that the singular value distribution could be used as a proxy for measuring representational power of the embedding layer. It has been argued that having fast decaying singular values leads to limiting the representational power of the embeddings to a small sub-space. If this is the case, then it may be preferable to have a more uniform singular value distribution (Wang et al., 2020). We compute the singular values for word embedding matrix of LN and PN, and we report the results in Figure 6. It can be clearly observed that the singular values corresponding to PN decay more slowly than those of LN. Intuitively, one explanation for this might be that PN helps by normalizing all the tokens across the batch dimension, which can result in a more equally distributed embeddings. This may illustrate one of the reasons why PN outperforms LN.

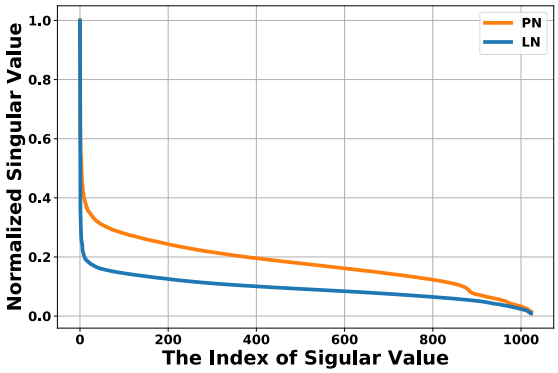

Figure 6. Singular values of embedding matrix trained with LN/PN on WMT14. We normalize the singular values of each matrix so that they are comparable with the largest one as 1. Note that the singular values corresponding to PN decay more slowly than those of LN.

6. Conclusion

In this work, we systematically analyze the ineffectiveness of vanilla batch normalization (BN) in transformers. Comparing NLP and CV, we show evidence that the batch statistics in transformers on NLP tasks have larger variations. This further leads to the poor performance of BN in transformers. By decoupling the variations into FP and BP computation, we propose PN-V and PN to alleviate the variance issue of BN in NLP. We also show the advantages of PN-V and PN, both theoretically and empirically. Theoretically, PN-V preserves the first-order smoothness property as in BN. The approximate backpropagation of PN leads to bounded gradients. Empirically, we show that PN outperforms LN in neural machine translation (0.4/0.6 BLEU on

IWSLT14/WMT14) and language modeling (5.6/3.0 PPL on PTB/WikiText-103) by a large margin. We also conduct further analysis of the effect of PN-V/PN/BN/LN under different batch size settings to show the significance of statistical estimations, and we investigate the representation power of learned embeddings matrix by LN/PN to illustrate the effectiveness of PN.

Acknowledgments

This work was supported by funds from Intel and Samsung. We are grateful to support from Google Cloud, Google TFTC team, as well as support from the Amazon AWS. We would like to acknowledge ARO, DARPA, NSF, and ONR for providing partial support of this work. We are also grateful to Zhuohan Li, Zhen Dong, Yang Liu, the members of Berkeley NLP, and the members of the Berkeley RISE Lab for their valuable feedback.

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Baevski, A. and Auli, M. Adaptive input representations for neural language modeling. In *ICLR*, 2019.

Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., and Courville, A. Recurrent batch normalization. In *ICLR*, 2017.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. In *ACL*, 2019.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

Fan, A., Grave, E., and Joulin, A. Reducing transformer depth on demand with structured dropout. In *ICLR*, 2020.

Gao, J., He, D., Tan, X., Qin, T., Wang, L., and Liu, T. Representation degeneration problem in training natural language generation models. In *ICLR*, 2019.

Goldberger, J., Hinton, G. E., Roweis, S. T., and Salakhutdinov, R. R. Neighbourhood components analysis. In *NeurIPS*, 2005.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 1997.

Hou, L., Zhu, J., Kwok, J., Gao, F., Qin, T., and Liu, T.-y. Normalization helps training of quantized lstm. In *NeurIPS*, 2019.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *CVPR*, 2017.

Inan, H., Khosravi, K., and Socher, R. Tying word vectors and word classifiers: A loss framework for language modeling. In *ICLR*, 2017.

Ioffe, S. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *NeurIPS*, 2017.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. What is the best multi-stage architecture for object recognition? In *ICCV*, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

Li, B., Wu, F., Weinberger, K. Q., and Belongie, S. Positional normalization. In *NeurIPS*, 2019.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *ICCV*, 2017.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Ma, X., Zhang, P., Zhang, S., Duan, N., Hou, Y., Zhou, M., and Song, D. A tensorized transformer for language modeling. In *NeurIPS*, 2019.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *ICLR*, 2017.

Mikolov, T., Deoras, A., Kombrink, S., Burget, L., and Černockỳ, J. Empirical evaluation and combination of advanced language modeling techniques. In *INTERSPEECH*, 2011.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. In *ICLR*, 2018.

Nguyen, T. Q. and Salazar, J. Transformers without tears: Improving the normalization of self-attention. *arXiv preprint arXiv:1910.05895*, 2019.

Ott, M., Edunov, S., Grangier, D., and Auli, M. Scaling neural machine translation. In *Machine Translation*, 2018.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. fairseq: A fast, extensible toolkit for sequence modeling. In *NAACL: Demonstrations*, 2019.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. Bleu: a method for automatic evaluation of machine translation. In *ACL*, 2002.

Qiao, S., Wang, H., Liu, C., Shen, W., and Yuille, A. Weight standardization. *arXiv preprint arXiv:1903.10520*, 2019.

Rahimi, A. Nuerips 2017 test-of-time award presentation, December 2017.

Salimans, T. and Kingma, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NeurIPS*, 2016.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. How does batch normalization help optimization? In *NeurIPS*, 2018.

Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. In *ACL*, 2016.

Singh, S. and Shrivastava, A. Evalnorm: Estimating batch normalization statistics for evaluation. In *ICCV*, 2019.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *NeurIPS*, 2017.

Wang, L., Huang, J., Huang, K., Hu, Z., Wang, G., and Gu, Q. Improving neural language generation with spectrum control. In *ICLR*, 2020.

Wang, Q., Li, B., Xiao, T., Zhu, J., Li, C., Wong, D. F., and Chao, L. S. Learning deep transformer models for machine translation. In *ACL*, 2019.

Wu, F., Fan, A., Baevski, A., Dauphin, Y., and Auli, M. Pay less attention with lightweight and dynamic convolutions. In *ICLR*, 2019.

Wu, Y. and He, K. Group normalization. In *ECCV*, 2018.

Xu, J., Sun, X., Zhang, Z., Zhao, G., and Lin, J. Understanding and improving layer normalization. In *NeurIPS*, 2019.

Yan, J., Wan, R., Zhang, X., Zhang, W., Wei, Y., and Sun, J. Towards stabilizing batch statistics in backward propagation of batch normalization. In *ICLR*, 2020.

Yang, Z., Dai, Z., Salakhutdinov, R., and Cohen, W. W. Breaking the softmax bottleneck: A high-rank rnn language model. In *ICLR*, 2018.

Yao, Z., Gholami, A., Keutzer, K., and Mahoney, M. W. PyHessian: Neural networks through the lens of the Hessian. *arXiv preprint arXiv:1912.07145*, 2019.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zhang, B. and Sennrich, R. Root mean square layer normalization. In *NeurIPS*, 2019.

Zhang, B., Titov, I., and Sennrich, R. Improving deep transformer with depth-scaled initialization and merged attention. In *EMNLP*, 2019a.

Zhang, H., Dauphin, Y. N., and Ma, T. Residual learning without normalization via better initialization. In *ICLR*, 2019b.

A. Training Details

A.1. Machine Translation.

Dataset The training/validation/test sets for the IWSLT14 dataset contain about 153K/7K/7K sentence pairs, respectively. We use a vocabulary of 10K tokens based on a joint source and target byte pair encoding (BPE) (Sennrich et al., 2016). For the WMT14 dataset, we follow the setup of (Vaswani et al., 2017), which contains 4.5M training parallel sentence pairs. Newstest2014 is used as the test set, and Newstest2013 is used as the validation set. The 37K vocabulary for WMT14 is based on a joint source and target BPE factorization.

Hyperparameter Given the unstable gradient issues of decoders in NMT (Zhang et al., 2019a), we only change all the normalization layers in the 6 encoder layers from LN to BN/PN, and we keep all the 6 decoder layers to use LN. For Transformer$_{\text{PN-V}}$ `big` and Transformer$_{\text{BN}}$ `big` (not Transformer$_{\text{PN}}$ `big`), we use the synchronized version, where each FP and BP will synchronize the mean/variance/quadratic mean of different batches at different nodes. For PN, we set the α in the forward and backward steps differently, and we tune the best setting over 0.9/0.95/0.99 on the validation set. To control the scale of the activation, we also involve a layer-scale layer (Zhang & Sennrich, 2019) in each model setting before the normalization layer. The warmup scheme for accumulating ψ is also employed, as suggested in (Yan et al., 2020) . Specifically, we do not tune the warmup steps, but we set it identical to the warmup steps for the learning rate schedule in the optimizer (Vaswani et al., 2017). We set dropout as 0.3/0.0 for Transformer `big`/`small` model, respectively. We use the Adam optimizer and follow the optimizer setting and learning rate schedule in (Wang et al., 2019). We set the maximum number of updates following (Ott et al., 2018) to be 300k for WMT and 100k for IWSLT. We used early stopping to stop the experiments by showing no improvement over the last 10/5 epochs. For the `big` model, we enlarge the batch size and learning rate, as suggested in (Ott et al., 2019), to accelerate training. We employ label smoothing of value $\epsilon_{\text{ls}} = 0.1$ in all experiments. We implement our code for MT using *fairseq-py* (Ott et al., 2019).

Evaluation We use BLEU[4] (Papineni et al., 2002) as the evaluation metric for MT. Following standard practice, we measure tokenized case-sensitive BLEU and case-insensitive BLEU for WMT14 En-De and IWSLT14 De-En, respectively. For a fair comparison, we do not include other external datasets. For inference, we average the last 10 checkpoints, and we set the length penalty to 0.6/1.0 and beam size to 4/5 for WMT/IWSLT, following (Ott et al., 2019).

A.2. Language Modeling.

Dataset PTB (Mikolov et al., 2011) has 0.93M training tokens, 0.073M validation words, and 0.082M test word. Wikitext-103 (Merity et al., 2017) contains 0.27M unique tokens, and 100M training tokens from 28K articles, with an average length of 3.6K tokens per article. We use the same evaluation scheme that was provided in (Dai et al., 2019).

Hyperparameter We use three layers tensorized transformer core-1 for PTB and six layers tensorized transformer core-1 for Wikitext-103, following (Ma et al., 2019). This means there exists only one linear projection in multi-linear attention. We replace every LN layer with a PN layer. For PN, we set the α in forward and backward differently, and we tune the best setting over 0.9/0.95/0.99 on the validation set. The warmup scheme and layer-scale are also the same as the hyperparameter setting introduced for machine translation. We set the dropout as 0.3 in all the datasets. The model is trained using 30 epochs for both PTB and WikiText-103. We use the Adam optimizer, and we follow the learning rate setting in (Ma et al., 2019). We set the warmup steps to be 4000 and label smoothing to be $\epsilon_{\text{ls}} = 0.1$ in all experiments.

B. Extra Results

B.1. Empirical Results for Lemma 2.

Under Assumption 9, mentioned in Section 4.1 and discussed in Appendix C.1, we show

$$\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}_{:,i}}\|^2 = \frac{\gamma_i^2}{(\psi_B)_i^2}\big(\|\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{X}_{:,i}}\|^2 - \langle \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{X}_{:,i}}, \frac{\widehat{\boldsymbol{X}}_{:,i}}{\sqrt{B}}\rangle^2\big).$$

Given that $\langle \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{X}_{:,i}}, \frac{\widehat{\boldsymbol{X}}_{:,i}}{\sqrt{B}}\rangle^2$ is non-negative, the Lipschitz constant of $\mathcal{L}$ is smaller than that of $\widehat{\mathcal{L}}$ if $\gamma_i \leqslant (\psi_B)_i$. Here, we

[4] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl