Root Mean Square Layer Normalization

Biao Zhang 1  Rico Sennrich 2, 1
1 School of Informatics, University of Edinburgh
2 Institute of Computational Linguistics, University of Zurich
B.Zhang@ed.ac.uk, sennrich@cl.uzh.ch

Abstract

Layer normalization (LayerNorm) has been successfully applied to various deep neural networks to help stabilize training and boost model convergence because of its capability in handling re-centering and re-scaling of both inputs and weight matrix. However, the computational overhead introduced by LayerNorm makes these improvements expensive and significantly slows the underlying network, e.g. RNN in particular. In this paper, we hypothesize that re-centering invariance in LayerNorm is dispensable and propose root mean square layer normalization, or RMSNorm. RMSNorm regularizes the summed inputs to a neuron in one layer according to root mean square (RMS), giving the model re-scaling invariance property and implicit learning rate adaptation ability. RMSNorm is computationally simpler and thus more efficient than LayerNorm. We also present partial RMSNorm, or pRMSNorm where the RMS is estimated from p% of the summed inputs without breaking the above properties. Extensive experiments on several tasks using diverse network architectures show that RMSNorm achieves comparable performance against LayerNorm but reduces the running time by 7%~64% on different models. Source code is available at https://github.com/bzhangGo/rmsnorm.

1 Introduction

How to train deep neural networks efficiently is a long-standing challenge. To accelerate model convergence, Ba et al. [3] propose the layer normalization (LayerNorm) which stabilizes the training of deep neural networks by regularizing neuron dynamics within one layer via mean and variance statistics. Due to its simplicity and requiring no dependencies among training cases, LayerNorm has been widely applied to different neural architectures, which enables remarkable success on various tasks ranging from computer vision [19, 26], speech recognition [37] to natural language processing [31, 35]. In some cases, LayerNorm was found to be essential for successfully training a model [6]. Besides, the decoupling from batch-based samples endows LayerNorm with the superiority over batch normalization (BatchNorm) [12] in handling variable-length sequences using RNNs.

Unfortunately, the incorporation of LayerNorm raises computational overhead. Although this is negligible to small and shallow neural models with few normalization layers, this problem becomes severe when underlying networks grow larger and deeper. As a result, the efficiency gain from faster and more stable training (in terms of number of training steps) is counter-balanced by an increased computational cost per training step, which diminishes the net efficiency, as show in Figure 1. One major feature of LayerNorm that is widely regarded as contributions to the stabilization is its re-centering invariance property: the summed inputs after LayerNorm remain intact when the inputs or weight matrix is shifted by some amount of noise. We argue that this mean normalization does not reduce the variance of hidden states or model gradients, and hypothesize that it has little impact on the success of LayerNorm.

In this paper, we propose root mean square layer normalization (RMSNorm), which regularizes the summed inputs to a neuron in one layer with the root mean square (RMS) statistic alone.
Figure 1: Training procedure of a GRU-based RNNSearch [4] for the first 10k training steps. Baseline means the original model without any normalization. When the Baseline training loss arrives at 7.0, the loss of LayerNorm reaches 5.4 after the same number of training steps [1(a)] but only 5.9 after the same training time [1(b)].

RMSNorm reduces the amount of computation and increases efficiency over LayerNorm. Despite the simpler formulation, the RMS normalizer helps stabilize the magnitude of layer activations, ensuring invariance to the re-scaling of both weights and datasets. We also show the possibility of estimating RMS on a subset of the summed inputs, maintaining this invariance property. Assuming that the summed inputs have an independent identically distributed structure, we propose partial RMSNorm, where only the first $p\%$ summed inputs are utilized for RMS estimation.

We thoroughly examine our model on various tasks, including machine translation, image classification, image-caption retrieval and question answering. Experimental results show that across different models, RMSNorm yields comparable performance against LayerNorm but shows superiority in terms of running speed with a speed-up of $7\%$–$64\%$. When estimating the RMS with partial (6.25\%) summed inputs, $p$RMSNorm achieves competitive performance compared to RMSNorm.

## 2 Related Work

One bottleneck deep neural networks have been hypothesized to suffer from is the *internal covariate shift* issue [27], where a layer’s input distribution changes as previous layers are updated, which significantly slows the training [3]. One promising direction to solve this problem is normalization. Ioffe and Szegedy [12] introduce batch normalization (BatchNorm) to stabilize activations based on mean and variance statistics estimated from each training mini-batch. Unfortunately, the reliance across training cases deprives BatchNorm of the capability in handling variable-length sequences, though several researchers develop different strategies to enable it in RNNs [16, 8]. Instead, Salimans and Kingma [22] propose weight normalization (WeightNorm) to reparameterize weight matrix so as to decouple the length of weight vectors from their directions. Ba et al. [3] propose layer normalization which differs from BatchNorm in that statistics are directly estimated from the same layer without accessing other training cases. Due to its simplicity and effectiveness, LayerNorm has been successfully applied to various deep neural models, and achieves state-of-the-art performance on different tasks [19, 37, 31, 6].

These studies pioneer the research direction that integrates normalization as a part of the model architecture. This paradigm ensures encouraging performance by shortening model convergence but at the cost of consuming more time for each running step. To improve efficiency, Arpit et al. [2] employ a data-independent method to approximately estimate mean and variance statistics, thus avoiding calculating batch statistics. Ioffe [11] propose batch renormalization so as to reduce the dependence of mini-batches in BatchNorm. Ulyanov et al. [30] replace batch normalization with instance normalization for image generation. Hoffer et al. [10] and Wu et al. [33] observe that $l1$-norm can act as an alternative of variance in BatchNorm with the benefit of fewer nonlinear operations and higher computational efficiency. Nevertheless, all these work still follow the original normalization structure and utilize mean statistic estimated from the whole summed inputs to handle re-centering invariance.

---

$^1$Note that the internal covariate shift is given as motivation by [12, 3]. Recent studies have proposed alternative explanations for the success of normalization, such as the uncontrollable growth of layer activations in unnormalized deep networks [5].
Different from these related work, the proposed RMSNorm modifies the normalization structure by removing the re-centering operation and regularizing the summed inputs with RMS alone. Our model only maintains the re-scaling invariance property which we find can be inherited when the RMS is estimated from only subset of the summed inputs, partially inspired by the group normalization [34]. As a side effect, our model reduces the computational overhead and increases efficiency. Recently, Zhang et al. [36] show that with careful initialization, residual networks can be trained as stable as those with normalization. However, the approach mainly aims at improving residual networks and cannot be freely switched without modifying all initialization layers. Besides, it is not trivial to be adapted to other general neural networks, such as RNNs where model depth expands along the variable sequence length. By contrast, our model is simple, effective and can be used as a drop-in replacement of LayerNorm.

3 Background

We briefly review LayerNorm in this section based on a standard feed-forward neural network. Given an input vector \( \mathbf{x} \in \mathbb{R}^m \), a feed-forward network projects it into an output vector \( \mathbf{y} \in \mathbb{R}^n \) through a linear transformation followed by a non-linear activation as follows:

\[
    a_i = \sum_{j=1}^{m} w_{ij} x_j, \quad y_i = f (a_i + b_i),
\]

where \( w_i \) is weight vector to the \( i \)-th output neuron, \( b_i \) is bias scalar which is usually initialized by 0, and \( f(\cdot) \) is an element-wise non-linear function. \( a \in \mathbb{R}^n \) denotes the weight-summed inputs to neurons, which is also the target of normalization.

This vanilla network might suffer from internal covariate shift issue [12], where a layer’s input distribution changes as previous layers are updated. This could negatively affect the stability of parameters’ gradients, delaying model convergence. To reduce this shift, LayerNorm normalizes the summed inputs so as to fix their mean and variance as follows:

\[
    \bar{a}_i = \frac{a_i - \mu}{\sigma} g_i, \quad y_i = f (\bar{a}_i + b_i),
\]

where \( \bar{a}_i \) is the \( i \)-th value of vector \( \bar{a} \in \mathbb{R}^n \), which acts as the normalized alternative of \( a_i \) for layer activation. \( g \in \mathbb{R}^n \) is the gain parameter used to re-scale the standardized summed inputs, and is set to 1 at the beginning. \( \mu \) and \( \sigma^2 \) are the mean and variance statistic respectively estimated from raw summed inputs \( a \):

\[
    \mu = \frac{1}{n} \sum_{i=1}^{n} a_i, \quad \sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (a_i - \mu)^2}.
\]

Thus, LayerNorm forces the norm of neurons to be decoupled from the inputs and weight matrix.

4 RMSNorm

A well-known explanation of the success of LayerNorm is its re-centering and re-scaling invariance property. The former enables the model to be insensitive to shift noises on both inputs and weights, and the latter keeps the output representations intact when both inputs and weights are randomly scaled. In this paper, we hypothesize that the re-scaling invariance is the reason for success of LayerNorm, rather than re-centering invariance.

We propose RMSNorm which only focuses on re-scaling invariance and regularizes the summed inputs simply according to the root mean square (RMS) statistic:

\[
    \bar{a}_i = \frac{a_i}{\text{RMS}(\mathbf{a})} g_i, \quad \text{where RMS}(\mathbf{a}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} a_i^2}.
\]

Intuitively, RMSNorm simplifies LayerNorm by totally removing the mean statistic in Eq. (3) at the cost of sacrificing the invariance that mean normalization affords. When the mean of summed inputs is zero, RMSNorm is exactly equal to LayerNorm. Although RMSNorm does not re-center
4.1 Invariance Analysis

Invariance measures whether model output after normalization changes highly in accordance with its input and weight matrix. Ba et al. [3] show that different normalization methods reveal different invariance properties, which contributes considerably to the model’s robustness. In this section, we theoretically examine the invariance properties of RMSNorm.

We consider the following general form of RMSNorm:

$$y = f \left( \frac{Wx}{\text{RMS}(a)} \odot g + b \right),$$  \hfill (5)

where $\odot$ denotes element-wise multiplication. Our main results are summarized in Table 1. RMSNorm is invariant to both weight matrix and input re-scaling, because of the following linearity property of RMS:

$$\text{RMS}(\alpha x) = \alpha \text{RMS}(x),$$  \hfill (6)

where $\alpha$ is a scale value. Suppose the weight matrix is scaled by a factor of $\delta$, i.e. $W' = \delta W$, then this change does not affect the final layer output:

$$y' = f \left( \frac{W'x}{\text{RMS}(a')} \odot g + b \right) = f \left( \frac{\delta Wx}{\delta \text{RMS}(a)} \odot g + b \right) = y.$$  \hfill (7)

By contrast, if the scaling is only performed on individual weight vectors, this property does not hold anymore as different scaling factors break the linearity property of RMS. Similarly, if we enforce a scale on the input with a factor of $\delta$, i.e. $x' = \delta x$, the output of RMSNorm remains through an analysis analogous to that in Eq. (7). We can easily extend the equality to batch-based inputs as well as the whole dataset. Therefore, RMSNorm is invariant to the scaling of its inputs.

The main difference to LayerNorm is that RMSNorm is not re-centered and thus does not show similar linearity property for variable shifting. It is not invariant to all re-centering operations.

4.2 Gradient Analysis

The above analysis only considers the effect of scaling inputs and the weight matrix on the layer output. In a general setting, however, a RMSNorm-enhanced neural network is trained via standard stochastic gradient descent approach, where the robustness of model gradient is very crucial to parameters’ update and model convergence (see also Santurkar et al. [23] who argue that the success of normalization methods does not come from the added stability to layer inputs, but due to increased smoothness of the optimization landscape). In this section, we investigate the properties of model gradients in RMSNorm.
Given a loss function $\mathcal{L}$, we perform back-propagation through Eq. (4) to obtain the gradient with respect to parameters $g, b$ as follows:

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial v} ′,$$
$$\frac{\partial \mathcal{L}}{\partial g} = \frac{\partial \mathcal{L}}{\partial v} ⊗ \frac{\partial v}{\partial \text{RMS}(a)}, \tag{8}$$

where $v$ is short for the whole expression inside $f(\cdot)$ in Eq. (4), and $\partial \mathcal{L}/\partial v$ is the gradient back-propagated from $\mathcal{L}$ to $v$. Both gradients $\partial \mathcal{L}/\partial b$ and $\partial \mathcal{L}/\partial g$ are invariant to the scaling of inputs $x$ and the weight matrix $W$ (in the case of $\partial \mathcal{L}/\partial g$ because of the linearity property in Eq. (6)). Besides, the gradient of $g$ is proportional to the normalized summed inputs, rather than raw inputs. This powers the stability of the magnitude of $g$.

Unlike these vector parameters, the gradient of the weight matrix $W$ is more complicated due to the quadratic computation in RMS. Formally,

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{i=1}^{n} \left[ x′ ⊗ \left( \text{diag} \left( g ⊕ \frac{\partial \mathcal{L}}{\partial v} \right) × R \right) \right], \text{ where } R = \frac{1}{\text{RMS}(a)} \left( I - \frac{(Wx)(Wx)^T}{n\text{RMS}(a)^2} \right), \tag{9}$$

$\text{diag}(\cdot)$ denotes the diagonal matrix of input, $⊗$ denotes the Kronecker product, and “$I$” indicates identity matrix. For clarity, we explicitly use “$\times$” to represent matrix multiplication. The matrix term $R$ associates the gradient of $W$ with both inputs $x$ and weight matrix $W$. With a thorough analysis, we can demonstrate that this term is negatively correlated with both input and weight matrix scaling. After assigning a scale of $\delta$ to either input $x$ ($x′ = \delta x$) or weight matrix ($W′ = \delta W$), we have

$$R′ = \frac{1}{\delta \text{RMS}(a)} \left( I - \frac{(\delta Wx)(\delta Wx)^T}{n\delta^2\text{RMS}(a)^2} \right) = \frac{1}{\delta} R. \tag{10}$$

If we put the scaled term $R′$ back into Eq. (9), we can easily prove that the gradient $\partial \mathcal{L}/\partial W$ is invariant to input scaling, but keeps the negative correlation with weight matrix scaling. Reducing the sensitivity of gradient $\partial \mathcal{L}/\partial W$ to the scaling of inputs ensures its smoothness and improves the stability of learning. On the other hand, the negative correlation acts as an implicit learning rate adaptor and dynamically controls the norm of gradients which avoids large-norm weight matrix and improves model convergence.

5 pRMSNorm

The re-scaling invariance property of RMSNorm ascribes to the linearity property of RMS. Considering that neurons in one layer often have independent identically distributed structure, we argue that the RMS can be estimated on a subset of these neurons rather than all of them. We propose partial RMSNorm (pRMSNorm). Given the unnormalized input $a$, pRMSNorm infers the RMS statistic from first-$p\%$ elements of $a$: $\text{RMS}(a) = \sqrt{\frac{1}{k} \sum_{i=1}^{k} a_i^2}$, where $k = \lceil n \cdot p \rceil$ denotes the number of elements used for RMS estimation. The linearity property still holds for RMS as in Eq. (8), which indicates pRMSNorm shares the same invariance properties as RMSNorm as shown in Table 1.

RMS is a biased estimation of the RMS which is often inaccurate. Though theoretically pRMSNorm approximates to RMSNorm, we observe gradient instability where the gradient tends to explode with small $m$. In practice, however, models with pRMSNorm can succeed in satisfactory convergence with a partial ratio of 6.25%.

6 Experiments

To test the efficiency of layer normalization across different implementations, we perform experiments with Tensorflow [1], PyTorch [20] and Theano [29]. We add RMSNorm to different models, comparing against an unnormalized baseline and LayerNorm. These models are based on diverse architectures, covering different RNN variants, convolutional and self-attentional models, and various activations (such as sigmoid, tanh, and softmax), with initialization ranging from uniform, normal, orthogonal with different initialization ranges or variances. Unless otherwise noted, all speed-related statistics are measured on one TITAN X (Pascal). Reported time is averaged over 3 runs. We also list the standard deviation of these three runs.
6.1 Machine Translation

Machine translation aims at transforming a sentence from one (source) language to another (target) language. We focus on neural machine translation based on an attention-enhanced encoder-decoder framework. We train two different models, a GRU-based RNNSearch [4] and a self-attention based neural Transformer [31] on WMT14 English-German translation task. More details about the experimental settings as well as comparison with WeightNorm are listed in Appendix A.1

We first experiment with RNNSearch. Normalization is added to the recurrent connections and feedforward layers. Apart from RNNSearch without any normalization (Baseline) and with LayerNorm, we also compare against the same model equipped with L2-Norm (i.e. replacing RMS with L2-Norm), which has been observed to improve lexical selection [18].

Figure 2 illustrates the evolution of BLEU score on our development set after every 30k training steps, and Table 2 summarizes the test results. In short, both LayerNorm and RMSNorm outperform the Baseline by accelerating model convergence: they reduce the number of training steps until convergence by about 50%, and improve test accuracy, with RMSNorm being comparable to LayerNorm. This supports our hypothesis that re-scaling invariance is the core property of LayerNorm, and that RMSNorm is an effective substitute. Our results with L2-Norm show that it fails to improve the model.

Table 2: SacreBLEU score on newstest2014 (Test14) and newstest2017 (Test17) for RNNSearch using Tensorflow-version Nematus. “Time”: the time in second per 1k training steps. We set $p$ to 6.25%. We highlight the best results in bold, and show the speedup of RMSNorm against LayerNorm in bracket.

<table>
<thead>
<tr>
<th>Model</th>
<th>Test14</th>
<th>Test17</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>21.7</td>
<td>23.4</td>
<td>399±3.40s</td>
</tr>
<tr>
<td>LayerNorm</td>
<td>22.6</td>
<td>23.6</td>
<td>665±32.5s</td>
</tr>
<tr>
<td>L2-Norm</td>
<td>20.7</td>
<td>22.0</td>
<td>482±19.7s</td>
</tr>
<tr>
<td>RMSNorm</td>
<td>22.4</td>
<td>23.7</td>
<td>501±11.8s (24.7%)</td>
</tr>
<tr>
<td>$p$RMSNorm</td>
<td>22.6</td>
<td>23.1</td>
<td>493±10.7s (25.9%)</td>
</tr>
</tbody>
</table>

Table 2: SacreBLEU score on newstest2014 (Test14) and newstest2017 (Test17) for RNNSearch using Tensorflow-version Nematus. “Time”: the time in second per 1k training steps. We set $p$ to 6.25%. We highlight the best results in bold, and show the speedup of RMSNorm against LayerNorm in bracket.

We also experiment with Transformer, which is based on self-attention, avoiding recurrent connections and allowing a higher degree of parallelization. Still, layer normalization is an important part of the architecture. We use an in-house Tensorflow implementation of the Transformer, and employ the base setting as in [31] with all models trained for 300K steps. We treat Transformer with no normalization as our Baseline, and compare RMSNorm-enhanced Transformer with LayerNorm-equipped Transformer. Table 4 shows the results, from which we observe the importance of normalization for Transformer, without which training fails. RMSNorm achieves BLEU scores comparable to LayerNorm, and yields a speedup of 7%~9%. Compared with RNNSearch, the relative cost of

---

2We note that Nguyen and Chiang [18] only applied L2-Norm to the last layer, and treat the scaling factor as a hyperparameter. While not a replication of their experiment, we still found it worth testing L2-Norm as an alternative to LayerNorm.
normalization is lower because there are significantly fewer sequential normalization operations in Transformer.

**Effect of Normalization on Mean and Standard Deviation** Table 5 shows the distribution of mean and standard deviation of hidden representations across token positions for an RNNSearch model. Mean and standard deviation are unstable in the baseline, as observed by Ba et al. [3]. Due to their normalization properties, both RMSNorm and LayerNorm stabilize standard deviation. Although the mean in RMSNorm is not normalized, in practice it is more stable than the mean of the baseline. This supports our hypothesis that RMSNorm stabilizes recurrent activations without the need to explicitly normalize the mean.

**On the Robustness of RMSNorm** One remaining question is whether the re-centering operation in LayerNorm (which RMSNorm abandons) makes models more robust towards arbitrary weight/bias initializations. We perform an experiment on RNNSearch with Nematus in Tensorflow, and change the center of weight initialization to 0.2. Results in Figure 4 show that LayerNorm becomes very unstable with abnormal initialization, but RMSNorm is more robust (both underperform the original initialization). Our empirical evidence so far suggests that RMSNorm is similarly robust as LayerNorm, or more.

### 6.2 CNN/Daily Mail Reading Comprehension

This reading comprehension task is a cloze-style question answering task, where models are required to answer a question regarding to a passage, and the answer is an anonymized entity from the passage [9]. We train a bidirectional attentive reader model proposed by Hermann et al. [9] on the CNN corpus. More details about the experimental settings are given in Appendix A.2. We compare RMSNorm with both LayerNorm and BatchNorm.

Figure 5 and Table 6 show the results. After normalizing RNN by BatchNorm with separate statistics for each time step in a sequence, both BatchNorm-LSTM and BatchNorm-Everywhere help speed up the convergence of training process. By contrast, LayerNorm and RMSNorm not only converge faster than BatchNorm, but also reach lower validation error rate, though pRMSNorm performs slightly
Figure 5: Error rate on validation set for the attentive reader model.

Figure 6: Recall@K values on validation set for the order-embedding models.

Table 6: Time in seconds per 0.1k training steps for the attentive reader model.

Table 8: Time in seconds per 0.1k training steps for the order-embedding model.

Figure 9 and Table 10 show the results. Models enhanced with a normalization technique converge faster than Baseline, among which BatchNorm performs the best. Similar to previous observation \(^3\), Notice that the implementation of BatchNorm is cuDNN-based, so time cost of BatchNorm in Table 6 cannot be directly compared with others.

---

\(^3\) Notice that the implementation of BatchNorm is cuDNN-based, so time cost of BatchNorm in Table 6 cannot be directly compared with others.
### Table 7: Average R@K values across 5 test sets from Microsoft COCO.

<table>
<thead>
<tr>
<th>Model</th>
<th>Caption Retrieval</th>
<th>Image Retrieval</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R@1</td>
<td>R@5</td>
</tr>
<tr>
<td>OE + Baseline [3]</td>
<td>45.4</td>
<td>88.7</td>
</tr>
<tr>
<td>OE + Baseline [3]†</td>
<td>46.7</td>
<td>89.9</td>
</tr>
<tr>
<td>OE + LayerNorm [3]</td>
<td>46.6</td>
<td>79.3</td>
</tr>
<tr>
<td>OE + Baseline [3]‡</td>
<td>48.5</td>
<td>80.6</td>
</tr>
<tr>
<td>OE + LayerNorm [3]</td>
<td>48.5</td>
<td>80.6</td>
</tr>
<tr>
<td>OE + Baseline</td>
<td>45.8</td>
<td>79.7</td>
</tr>
<tr>
<td>OE + LayerNorm</td>
<td>47.6</td>
<td>79.5</td>
</tr>
<tr>
<td>OE + RMSNorm</td>
<td>48.7</td>
<td>79.7</td>
</tr>
<tr>
<td>OE + pRMSNorm</td>
<td>46.8</td>
<td>79.8</td>
</tr>
</tbody>
</table>

**Mean r**: mean rank, lower is better. The number in bold highlights the best result. † denotes the reproduced results of ‡.

### Table 9: Training error rate for the ConvPool-CNN-C model.

<table>
<thead>
<tr>
<th>Model</th>
<th>Test Error</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>8.96%</td>
<td>21±0.0s</td>
</tr>
<tr>
<td>BatchNorm</td>
<td>8.25%</td>
<td>38±0.0s</td>
</tr>
<tr>
<td>WeightNorm</td>
<td>8.28%</td>
<td>23±0.0s</td>
</tr>
<tr>
<td>LayerNorm</td>
<td>10.49%</td>
<td>39±0.4s</td>
</tr>
<tr>
<td>RMSNorm</td>
<td>8.83%</td>
<td>31±0.3s (20.5%)</td>
</tr>
<tr>
<td>pRMSNorm</td>
<td>10.37%</td>
<td>30±0.4s (23.1%)</td>
</tr>
</tbody>
</table>

Table 10: Test error rate and time in seconds per training epoch for the ConvPool-CNN-C model. Time is measured with GeForce RTX 2080 Ti.

we also find that layer normalization works worse than BatchNorm and WeightNorm for image processing. Though LayerNorm outperforms Baseline by shorting model convergence, it fails to generalize to the test set, degenerating the test error by 1.53%. In contrast, RMSNorm shows better generalization, surpassing the Baseline by 0.013% and saving about 20.5% training time compared to LayerNorm. pRMSNorm gains further speedup of 2.6%, albeit at the cost of sacrificing test accuracy of 1.54%.

### 7 Conclusion and Future Work

This paper presents RMSNorm, a novel normalization approach that normalizes the summed inputs according to the RMS. RMSNorm preserves the re-scaling invariance property of LayerNorm but eschews the re-centering invariance property which contributes less to the model training. Compared with LayerNorm, models with RMSNorm suffers from less computational overhead. RMSNorm can be easily applied to different model architectures as a drop-in replacement of LayerNorm. Experiments on several NLP tasks show that RMSNorm is comparable to LayerNorm in quality, but accelerates the running speed. Actual speed improvements depend on the framework, hardware, neural network architecture and relative computational cost of other components, and we empirically observed speedups of 7%~64% across different models and implementations. Our efficiency improvement come from simplifying the computation, and we thus expect them to be orthogonal to other means of increasing training speed, such as low-precision arithmetic and GPU kernel fusion. We also experimented with $p$RMSNorm which estimates the RMS on a subset of the summed inputs. While theoretically faster, we did not consistently observe empirical speed improvements for $p$RMSNorm. We leave it to future work to investigate if the performance can be improved via code optimization.

In the future, we would like to take more analysis about the success behind RMSNorm. Inspired by recent success of 11-norm for BatchNorm, we will explore different norms for RMSNorm, and simplify other normalization techniques such as BatchNorm.

### Acknowledgments

We thank the reviewers for their insightful comments, and Antonio Valerio Miceli Barone for his support with weight normalization for MT. This project has received funding from the grant H2020-ICT-2018-2-825460 (ELITR) by the European Union. Biao Zhang also acknowledges the support of the Baidu Scholarship. This work has been performed using resources provided by the Cambridge Tier-2 system operated by the University of Cambridge Research Computing Service (http://www.hpc.cam.ac.uk) funded by EPSRC Tier-2 capital grant EP/P020259/1.
References


