Root Mean Square Layer Normalization

Anonymous Author(s)
Affiliation
Address
email

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. We will release our source code soon.

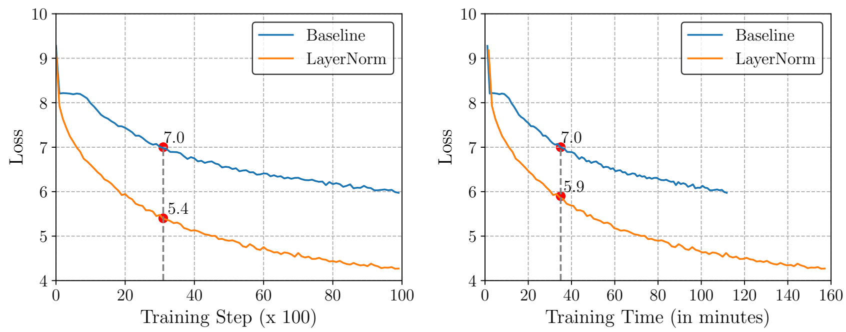
1 Introduction

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 [17, 24], speech recognition [34] to natural language processing [29]. In some cases, LayerNorm was found to be essential for successfully training a model [5]. Besides, the decouple from batch-based samples endows LayerNorm with the superiority over batch normalization (BatchNorm) [11] 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 clearly 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. RMSNorm

Submitted to 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Do not distribute.



(a) Training loss vs. training steps. (b) Training loss vs. training time.

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).

reduces the amount of computation and increases efficiency over LayerNorm. Despite the simpler formulation, the RMS normalizer helps reduce covariate shift 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-caption retrieval and question answering. Experimental results show that across different models, pRMSNorm yields comparable performance against LayerNorm but shows superiority in terms of running speed with a speed-up of $7\%\sim64\%$. When estimating the RMS with partial (6.25%) summed inputs, pRMSNorm achieves competitive performance compared to RMSNorm.

2 Related Work

Deep neural networks suffer from the *internal covariate shift* issue [25], where a layer's input distribution changes as previous layers are updated, which significantly slows the training. One promising direction to solve this problem is normalization. Ioffe and Szegedy [11] 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 [14, 7]. Instead, Salimans and Kingma [20] 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 [17, 34, 29, 5].

These studies pioneer the research direction that integrates normalization as a part of the model architecture. This paradigm ensures encouraging performance by shorting 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 [10] propose batch renormalization so as to reduce the dependence of mini-batches in BatchNorm. Ulyanov et al. [28] replace batch normalization with instance normalization for image generation. Hoffer et al. [9] and Wu et al. [31] 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.

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 [32]. As a side effect, our model reduces the computational overhead and increases efficiency. Recently, Zhang et al. [33] 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 can not 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:

$$\mathbf{a} = \mathbf{W}\mathbf{x}, \quad \mathbf{y} = f\left(\mathbf{a} + \mathbf{b}\right), \tag{1}$$

where $\mathbf{W}$ is weight matrix, $\mathbf{b}$ is bias term which is usually initialized by 0, and $f(\cdot)$ is an element-wise non-linear function. $\mathbf{a} \in \mathbb{R}^n$ denotes the weight-summed inputs to neurons, which is also the target of normalization.

This vanilla network suffers from *internal covariate shift* issue [11], 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:

$$\bar{a}_i = \frac{a_i - \mu}{\sigma} g_i, \quad y_i = f\left(\bar{a}_i + b_i\right), \tag{2}$$

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

$$\mu = \frac{1}{n}\sum_{i=1}^{n} a_i, \quad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_i - \mu)^2}. \tag{3}$$

In this way, LayerNorm forces the norm of neurons to be decoupled from both 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:

$$\bar{a}_i = \frac{a_i}{\text{RMS}(\mathbf{a})} g_i, \quad \text{where } \text{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}. \tag{4}$$

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 the summed inputs as in LayerNorm, we demonstrate through experiments that this property is not fundamental to the success of LayerNorm, and that RMSNorm is similarly effective.

	Weight matrix re-scaling	Weight matrix re-centering	Weight vector re-scaling	Dataset re-scaling	Dataset re-centering	Single training case re-scaling
BatchNorm	✓	✗	✓	✓	✓	✗
WeightNorm	✓	✗	✓	✗	✗	✗
LayerNorm	✓	✓	✗	✓	✗	✓
RMSNorm	✓	✗	✗	✓	✗	✓
pRMSNorm	✓	✗	✗	✓	✗	✓

Table 1: Invariance properties of different normalization methods. "✓" indicates invariant, while "✗" denotes the opposite.

RMS measures the quadratic mean of inputs, which in RMSNorm forces the summed inputs into a $\sqrt{n}$-scaled unit sphere. By doing so, the output distribution remains regardless of the scaling of input and weight distributions, tackling the issue of internal covariate shift. Although Euclidean norm which only differs from RMS by a factor of $\sqrt{n}$ has been successfully explored [20], we empirically find that it does not work for layer normalization. We hypothesize that scaling the sphere with the size of the input vector is important because it makes the normalization more robust across vectors of different size. As far as we know, the idea of employing RMS for neural network normalization has not been investigated before.

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:

$$\mathbf{y} = f\left(\frac{\mathbf{W}\mathbf{x}}{\text{RMS}(\mathbf{a})} \odot \mathbf{g} + \mathbf{b}\right), \tag{5}$$

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:

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

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

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

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 δ, i.e. $\mathbf{x}' = \delta\mathbf{x}$, 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. [21] 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 $\mathbf{g}, \mathbf{b}$ as follows:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{b}} = \frac{\partial\mathcal{L}}{\partial\mathbf{v}}, \quad \frac{\partial\mathcal{L}}{\partial\mathbf{g}} = \frac{\partial\mathcal{L}}{\partial\mathbf{v}} \odot \frac{\mathbf{W}\mathbf{x}}{\text{RMS}(\mathbf{a})}, \tag{8}$$

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

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

$$\frac{\partial\mathcal{L}}{\partial\mathbf{W}} = \sum_{i=1}^{n}\left[\mathbf{x}^T \otimes \left(\text{diag}\left(\mathbf{g}\odot\frac{\partial\mathcal{L}}{\partial\mathbf{v}}\right)\times\mathbf{R}\right)\right]_i, \text{where } \mathbf{R} = \frac{1}{\text{RMS}(\mathbf{a})}\left(\mathbf{I} - \frac{(\mathbf{W}\mathbf{x})(\mathbf{W}\mathbf{x})^T}{n\text{RMS}(\mathbf{a})^2}\right), \tag{9}$$

diag$(\cdot)$ denotes the diagonal matrix of input, $\otimes$ denotes the Kronecker product, and "$\mathbf{I}$" indicates identity matrix. For clarity, we explicitly use "$\times$" to represent matrix multiplication. The matrix term $\mathbf{R}$ associates the gradient of $\mathbf{W}$ with both inputs $\mathbf{x}$ and weight matrix $\mathbf{W}$. 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 δ to either input $\mathbf{x}$ ($\mathbf{x}' = \delta\mathbf{x}$) or weight matrix ($\mathbf{W}' = \delta\mathbf{W}$), we have

$$\mathbf{R}' = \frac{1}{\delta\text{RMS}(\mathbf{a})}\left(\mathbf{I} - \frac{(\delta\mathbf{W}\mathbf{x})(\delta\mathbf{W}\mathbf{x})^T}{n\delta^2\text{RMS}(\mathbf{a})^2}\right) = \frac{1}{\delta}\mathbf{R}. \tag{10}$$

If we put the scaled term $\mathbf{R}'$ back into Eq. (9), we can easily prove that the gradient $\partial\mathcal{L}/\partial\mathbf{W}$ is invariant to input scaling, but keeps the negative correlation with weight matrix scaling. Reducing the sensitivity of gradient $\partial\mathcal{L}/\partial\mathbf{W}$ 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 $\mathbf{a}$, pRMSNorm infers the RMS statistic from first-p% elements of $\mathbf{a}$: $\overline{\text{RMS}}(\mathbf{a}) = \sqrt{\frac{1}{m}\sum_{i=1}^{m} a_i^2}$, where $m = \lceil n \cdot p \rceil$ denotes the number of elements used for RMS estimation. Obviously, the linearity property still holds for the estimated RMS, which indicates pRMSNorm shares the same invariance properties as RMSNorm as shown in Table 1.

$\overline{\text{RMS}}$ 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 [18] and Theano [27]. We add RMSNorm to different models, comparing against an unnormalized baseline and LayerNorm. Unless otherwise noted, all speed-related statistics are measured on one TITAN X (Pascal). Reported time is averaged over 3 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 [29] on WMT14 English-German translation task. More details about the experimental settings are listed in Appendix A.1

We first experiment with RNNSearch. In this experiment, we set the embedding size and hidden size to be 512 and 1024 respectively. Normalization is added to the recurrent connections and feedforward

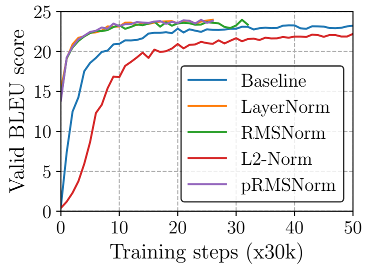


Figure 2: SacreBLEU score on newstest2013 for the RNNSearch. Models are implemented according to *Nematus* [23] in Tensorflow.

Model	Test2014	Test2017	Time
Baseline	21.7	23.4	399s
LayerNorm	22.6	23.6	665s
L2-Norm	20.7	22.0	482s
RMSNorm	22.4	**23.7**	501s (24.7%)
*p*RMSNorm	**22.6**	23.1	493s (25.9%)

Table 2: SacreBLEU score on newstest2014 (Test2014) and newstest2017 (Test2017) 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.

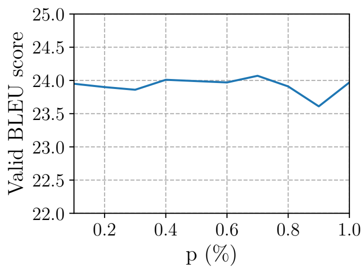


Figure 3: SacreBLEU score on newstest2013 (development set) for the RNNSearch with pRMSNorm. We use Tensorflow-version Nematus, and change p by a constant step size of 10%.

	Model	Test2014	Test2017	Time
Th	Baseline	21.8	22.9	596s
	LayerNorm	22.3	23.8	988s
	RMSNorm	22.5	23.2	652s (34.0%)
	*p*RMSNorm	**22.7**	**24.0**	658s (33.4%)
Py	Baseline	22.7	**24.7**	427s
	LayerNorm	23.2	24.3	857s
	RMSNorm	22.9	24.5	763s (11.0%)
	*p*RMSNorm	**23.2**	24.6	754s (12.0%)

Table 3: SacreBLEU score on newstest2014 (Test2014) and newstest2017 (Test2017) for RNNSearch. "*Th*": Theano-version Nematus, "*Py*": an in-house PyTorch-based RNNSearch.

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 [16].

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.[1] Results in Table 2 highlight the challenge that RNN with LayerNorm in Tensorflow suffers from serious computational inefficiency, where LayerNorm is slower than the Baseline by about 67%. In this respect, RMSNorm performs significantly better, improving upon LayerNorm by ∼25%.

Table 3 further lists translation results of different models implemented in Theano and Pytorch. Overall, RMSNorm yields comparable translation quality compared with LayerNorm but incurs less computational overhead, outperforming LayerNorm with speedups ranging from 11%∼34%. In addition, we observe that though in theory the amount of computation in pRMSNorm is less than that in RMSNorm, pRMSNorm ($p = 6.25\%$) sometimes tends to be slower. We ascribe this to the non-optimal implementation of tensor slicing operation in these computational frameworks, which can be improved with specific low-level coding.

In pRMSNorm, the partial ratio p directly controls the accuracy of estimated RMS, thereby affecting the stability of model training. Figure 3 shows the effect of p on model performance. Surprisingly, we find that the scale of p has little influence on the final translation quality in RNNSearch: using a small ratio does not significantly degenerate BLEU score. We set p to 6.25% for all following experiments.

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 [29] with all models trained for 300K steps. We treat Transformer with no normal-

[1] We note that Nguyen and Chiang [16] 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.

Model	Test2014	Test2017	Time
Baseline	-	-	210s
LayerNorm	26.6	27.7	248s
RMSNorm	**26.8**	27.7	231s (6.9%)
*p*RMSNorm	26.5	**27.8**	225s (9.3%)

Table 4: SacreBLEU score on newstest2014 (Test2014) and newstest2017 (Test2017) for the Transformer. "*Time*": the time in second per 1k training steps, which is measured using Tesla V100. "-" indicates that we fail to train this model and BLEU score is 0.

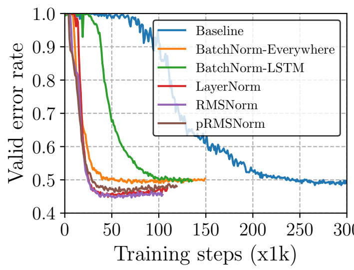


Figure 4: Error rate on validation set for the attentive reader model.

Model		1	2	3	4	ALL
Baseline	M	-2.60	-1.19	-1.43	-1.53	-1.60
	S	7.35	2.33	2.61	2.73	3.04
LayerNorm	M	-0.43	-0.48	-0.50	-0.50	-0.51
	S	1.19	1.51	1.51	1.51	1.51
RMSNorm	M	-0.40	-0.60	-0.69	-0.74	-0.73
	S	1.27	1.51	1.50	1.49	1.50

Table 5: Mean (M) and standard deviation (S) statistics estimated on the hidden-to-hidden mapping of decoder-part GRU cell in RNNSearch model. We use the newstest2013 dataset. *ALL*: the statistics averaged across all token positions. Numbers *1,2,3,4* indicate the statistic estimated for specific token positions.

Model	Time
Baseline	315s
BatchNorm-Everywhere	348s
BatchNorm-LSTM	345s
LayerNorm	392s
RMSNorm	333s (15.1%)
*p*RMSNorm	330s (15.8%)

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

ization 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%$\sim$9%. Compared with RNNSearch, the relative cost of 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.

6.2 CNN/Daily Mail Reading Comprehension

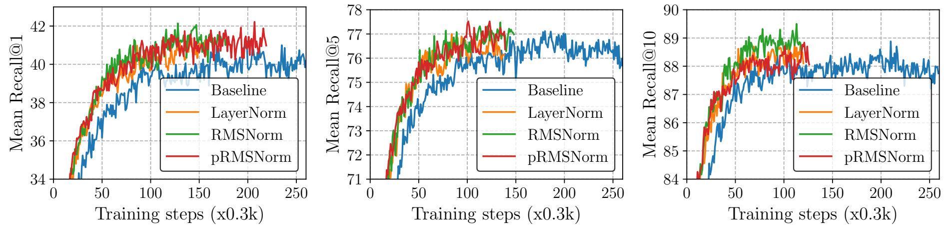
This reading comprehension task is a cloze-style question answering task, where models are required to answer a question regarding to a passage, and the answer is an anonymized entity from the passage [8]. We train a bidirectional attentive reader model proposed by Hermann et al. [8] on the CNN corpus. More details about the experimental settings are given in Appendix A.2. We compare RMSNorm with both LayerNorm and BatchNorm.

Figure 4 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 worse than RMSNorm. Although in Figure 4 the performance of RMSNorm and LayerNorm is comparable, RMSNorm is around 15% faster than LayerNorm as shown in Table 6.[2]

6.3 Image-Caption Retrieval

Image-caption retrieval is a cross-modal task aiming at learning a joint embedding space of images and sentences, which consists of two sub-tasks: *image retrieval* and *caption retrieval*. The former ranks a set of images according to a query caption, and the latter ranks a set of captions based on a query image. We train an order-embedding model (OE) proposed by Vendrov et al. [30] on

[2]Notice that the implementation of BatchNorm is cuDNN-based, so time cost of BatchNorm in Table 6 can not be directly compared with others.



(a) Recall@1 (b) Recall@5 (c) Recall@10

Figure 5: Recall@K values on validation set for the order-embedding models.

	Model	Caption Retrieval				Image Retrieval			
		R@1	**R@5**	**R@10**	**Mean r**	**R@1**	**R@5**	**R@10**	**Mean r**
Existing Work	Sym [30]	45.4		88.7	5.8	36.3		85.8	9.0
	OE [30]	46.7		88.9	5.7	37.9		85.9	8.1
	OE [3]	46.6	79.3	89.1	5.2	37.8	73.6	85.7	7.9
	OE + LayerNorm [3]	48.5	**80.6**	89.8	**5.1**	38.9	74.3	86.3	7.6
This Work	OE + Baseline	45.8	79.7	88.8	5.4	37.6	73.6	85.8	7.7
	OE + LayerNorm	47.9	79.5	89.2	5.3	38.4	74.6	**86.7**	7.5
	OE + RMSNorm	**48.7**	79.7	89.5	5.3	**39.0**	**74.8**	86.3	7.5
	OE + pRMSNorm	46.8	79.8	**90.3**	5.2	39.0	74.5	86.3	**7.4**

Table 7: Average R@K values across 5 test sets from Microsoft COCO. **R@K**: Recall @ K, higher is better. **Mean r**: mean rank, lower is better. The number in bold highlights the best result.

the Microsoft COCO dataset [15] using their public source code in Theano. Model details about experimental settings are provides in Appendix A.3. We compare RMSNorm with two models: one without any normalization (Baseline) and one with LayerNorm.

Model	Time
Baseline	2.11s
LayerNorm	12.02s
RMSNorm	7.12s (40.8%)
pRMSNorm	4.34s (63.9%)

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

Figure 5 shows the R@K curve on validation set after every 300 training steps, and Table 7 lists the final test results. Across all these metrics, RMSNorm and LayerNorm consistently outperform the Baseline in terms of model convergence as shown in Figure 5. We observe that on the validation set, RMSNorm slightly exceeds LayerNorm with respect to recall value. For the final test results as shown in Table 7, both RMSNorm and LayerNorm improve the model performance, reaching higher recall values (except LayerNorm on R@5) and lower mean rank, though RMSNorm reveals better generalization than LayerNorm. Besides, results in Table 8 show that RMSNorm accelerates training speed by 40%∼64% compared with LayerNorm, highlighting better efficiency of pRMSNorm.

6.4 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 pRMSNorm which estimates the RMS on a subset of the summed inputs. While theoretically faster, we did not observe empirical speed improvements for pRMSNorm. 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 $l1$-norm for BatchNorm, we are also interested in exploring different norms for RMSNorm, and in simplifying other normalization techniques such as BatchNorm.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, 2016. ISBN 978-1-931971-33-1.

[2] Devansh Arpit, Yingbo Zhou, Bhargava U Kota, and Venu Govindaraju. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. *arXiv preprint arXiv:1603.01431*, 2016.

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

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv e-prints*, abs/1409.0473, September 2014.

[5] Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–86. Association for Computational Linguistics, 2018.

[6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[7] Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

[8] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701, 2015.

[9] Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. *arXiv preprint arXiv:1803.01814*, 2018.

[10] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances in Neural Information Processing Systems*, pages 1945–1953, 2017.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456, 2015.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Ryan Kiros, Ruslan Salakhutdinov, and Richard S. Zemel. Unifying visual-semantic embeddings with multimodal neural language models. *CoRR*, abs/1411.2539, 2014.

[14] César Laurent, Gabriel Pereyra, Philémon Brakel, Ying Zhang, and Yoshua Bengio. Batch normalized recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2657–2661. IEEE, 2016.

[15] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[16] Toan Q Nguyen and David Chiang. Improving lexical choice in neural machine translation. *arXiv preprint arXiv:1710.01329*, 2017.

[17] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, and Alexander Ku. Image transformer. *arXiv preprint arXiv:1802.05751*, 2018.

[18] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[19] Matt Post. A call for clarity in reporting bleu scores. *arXiv preprint arXiv:1804.08771*, 2018.

[20] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems 29*, pages 901–909. 2016.

[21] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems 31*, pages 2488–2498. 2018.

[22] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[23] Rico Sennrich, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, and Maria Nadejde. Nematus: a Toolkit for Neural Machine Translation. In *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 65–68, Valencia, Spain, April 2017.

[24] Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2556–2565, 2018.

[25] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, 2000.

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[27] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, May 2016.

[28] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, 2016.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. 2017.

[30] Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. Order-embeddings of images and language. *arXiv preprint arXiv:1511.06361*, 2015.

[31] Shuang Wu, Guoqi Li, Lei Deng, Liu Liu, Dong Wu, Yuan Xie, and Luping Shi. L1-norm batch normalization for efficient training of deep neural networks. *IEEE transactions on neural networks and learning systems*, 2018.

[32] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[33] Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Residual learning without normalization via better initialization. In *International Conference on Learning Representations*, 2019.

[34] Shiyu Zhou, Linhao Dong, Shuang Xu, and Bo Xu. Syllable-based sequence-to-sequence speech recognition with the transformer in mandarin chinese. *arXiv preprint arXiv:1804.10752*, 2018.

A Appendix

A.1 Machine Translation

We experiment on the WMT14 English-German translation task, where the training corpus consists of 4.5M aligned sentence pairs. We use newstest2013 as the development set for model selection, newstest2014 and newstest2017 as the test set. We evaluate translation quality with case-sensitive detokenized BLEU score reported by *sacrebleu* [19][3]. Byte pair encoding algorithm is applied to reduce out-of-vocabulary tokens with 32k merge operations [22]. All models are trained based on maximum log-likelihood relaxed by label smoothing with a factor of 0.1.[4]

A.2 CNN/Daily Mail Reading Comprehension

The model is trained on the CNN corpus based on the public source code in Theano [7]. We adopt the *top4* setting, where each passage in the pre-processed dataset contains at most 4 sentences. For fair comparison with LayerNorm, we only employ RMSNorm within LSTM. We set hidden size of LSTM to be 240. Models are optimized via Adam optimizer [12] with a batch size of 64 and learning rate of $8e^{-5}$.

A.3 Image-Caption Retrieval

In OE model, sentences are encoded through a GRU-based RNN [6] and images are represented by the output of a pretrained VGGNet [26]. OE treats the caption-image pairs as a two-level partial order, and trains the joint model using the pairwise ranking loss [13].

We adopt the *10crop* feature from VGGNet as image representation, and set word embedding size and GRU hidden size to be 300 and 1024 respectively. All models are trained with Adam optimizer, with a batch size of 128 and learning rate of $1e^{-3}$. We employ Recall@K (R@K) values for evaluation, and report averaged results on five separate test sets (each consisting of 1000 images and 5000 captions) as our final test results.

[3] Sacrebleu hash: BLEU+case.mixed+lang.en-de+numrefs.1+smooth.exp+test.wmt14+tok.13a+version.1.2.12 and BLEU+case.mixed+lang.en-de+numrefs.1+smooth.exp+test.wmt17+tok.13a+version.1.2.12.

[4] Note that there are minor differences between different frameworks, both in implementation details and setup, explaining performance differences between the baselines.