Root Mean Square Layer Normalization

First Author
Affiliation / Address line 1
Affiliation / Address line 2
Affiliation / Address line 3
email@domain

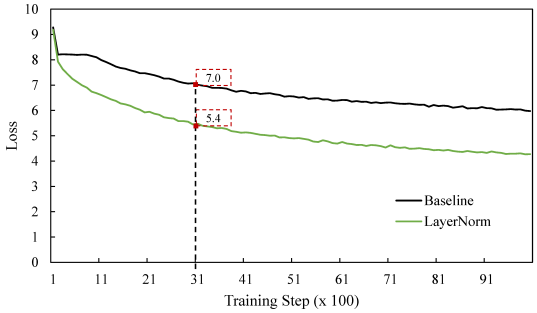
Second Author
Affiliation / Address line 1
Affiliation / Address line 2
Affiliation / Address line 3
email@domain

Abstract

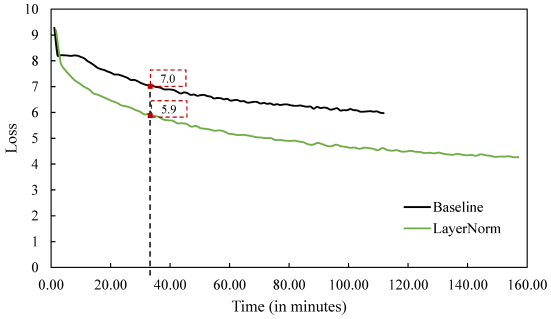
Layer normalization (LayerNorm) has been successfully applied to various deep neural networks to help stabilize training and boost model convergence. However, the calculation of mean and variance statistics used for normalization is sequential and time-consuming, which significantly slows the underlying network, e.g. recurrent neural network in particular. In this paper, we propose root mean square layer normalization, or *RMSNorm*, to reduce the normalization computation while keeping the stabilization capability. Unlike LayerNorm, RMSNorm regularizes the summed inputs to a neuron within one layer according to root mean square, giving the model invariance properties but avoiding sequentially calculating the mean and variance statistics so as to be faster. We conducted extensive experiments on both language-based and image-based tasks with diverse network architectures. The results show that RMSNorm achieves comparable performance, but reduces the running time by 2.5%∼15% on different models against LayerNorm.

1 Introduction

How to train deep neural networks efficiently is a long-standing challenge. To accelerate model convergence so as to reduce the overall training time, Ba et al. (2016) 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 (Parmar et al., 2018; Sharma et al., 2018), speech recognition (Zhou et al., 2018) to natural language processing (Vaswani et al., 2017). In some cases,



(a) Training loss vs. training steps.



(b) Training loss vs. training time.

Figure 1: Training procedure of a GRU-based RNNSearch (Bahdanau et al., 2014) 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).

LayerNorm was found to be essential for successfully training a model (Chen et al., 2018). Besides, the decouple from batch-based samples endows LayerNorm with the superiority over batch normalization (BatchNorm) (Ioffe and Szegedy, 2015) in handling variable-length sequences using RNNs.

One major drawback of LayerNorm which is shared with other normalization techniques such as BatchNorm is its inefficiency in the sequential calculation of normalization statistics: the vari-

ance cannot be obtained before getting the mean. Therefore, during both forward pass and back-propagation, LayerNorm takes at least two-step computation without parallelization. Although this is negligible to small and shallow neural models with few normalization layers, this inefficiency 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. This hinders complex neural models with LayerNorm from scaling to large-scale datasets, thereby impeding its practical usage.

In this paper, we propose root mean square layer normalization (RMSNorm) to accelerate the computation. RMSNorm regularizes the summed inputs to a neuron in one layer with the root mean square (RMS) statistic alone. In this way, it avoids the two-step calculation of the original layer normalization, thus leading to speed improvement. Despite the simpler formulation, the RMS normalizer helps reduce covariate shift of layer activations, ensuring the invariance properties to the re-scaling of both weights and datasets. Compared with LayerNorm, RMSNorm reaches a better trade-off between efficiency and effectiveness.

We thoroughly examined our model on both language-based and image-based tasks, including machine translation, image classification, image-caption retrieval and question answering, where recurrent, convolutional and attentive neural models are all involved. Experimental results show that across different models, RMSNorm yields comparable performance against LayerNorm, but shows clear superiority in terms of running speed, with a speed-up of 2.5%$\sim$15%.

2 Related Work

Deep neural networks suffer from the *internal covariate shift* issue (Shimodaira, 2000), 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 (2015) 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 (Laurent et al., 2016; Cooijmans et al., 2016). Instead, Salimans and Kingma (2016) propose weight normalization (WeightNorm) to reparameterize weight matrix so as to decouple the length of weight vectors from their directions. Ba et al. (2016) 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 (Parmar et al., 2018; Zhou et al., 2018; Vaswani et al., 2017; Chen et al., 2018).

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 on the cost of consuming more time for each running step. To improve efficiency, Arpit et al. (2016) employ a data-independent manner to approximately estimate mean and variance statistics, thus avoiding calculating batch statistics. Ioffe (2017) propose batch renomalization so as to reduce the dependence of mini-batches in BatchNorm. Ulyanov et al. (2016) replace batch normalization with instance normalization for image generation. Hoffer et al. (2018) and Wu et al. (2018) 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, these work mainly focus on improving BatchNorm, and are not trivial to be directly transferred to LayerNorm.

Our RMSNorm shares the aim of increasing efficiency with related work, but simplifies the computation in a novel way: instead of normalizing mean and variance, we normalize the root mean square of the summed inputs. Our model is thus mathematically different, but similarly effective and faster than LayerNorm, and can be used as a drop-in replacement.

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 (Ioffe and Szegedy, 2015), 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. Unfortunately, the calculation of σ relies on that of μ as shown in Eq. (3). This means that LayerNorm requires two sequential computations, which significantly reduces its computational efficiency.

4 RMSNorm

To avoid the sequential dependency between μ and σ as shown in Eq. (3), we propose RMSNorm which regularizes the summed inputs simply according to the root mean square (RMS) statistic:

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

Intuitively, RMSNorm simplifies LayerNorm by totally removing the mean statistic in Eq. (3), thereby increasing computational efficiency 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 empirically that this property is not fundamental to the success of LayerNorm, and that the more efficient RMSNorm is similarly effective.

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 (Salimans and Kingma, 2016), 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. (2016) 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:

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

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:

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

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

	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	✓	✗	✗	✓	✗	✓

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

affect the final layer output:

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

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. (2018) 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:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{b}} = \frac{\partial\mathcal{L}}{\partial\mathbf{v}}, \tag{8}$$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{g}} = \frac{\partial\mathcal{L}}{\partial\mathbf{v}} \odot \frac{\mathbf{W}\mathbf{x}}{\text{RMS}(\mathbf{a})}, \tag{9}$$

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,

$$\begin{aligned}&\frac{\partial\mathcal{L}}{\partial\mathbf{W}} = \sum_{i=1}^{n}\left[\mathbf{x}^T \otimes \left(\text{diag}\left(\mathbf{g} \odot \frac{\partial\mathcal{L}}{\partial\mathbf{v}}\right) \times \mathbf{R}\right)\right]_i, \\ &\text{where } \mathbf{R} = \frac{1}{\text{RMS}(\mathbf{a})}\left(\mathbf{I} - \frac{(\mathbf{W}\mathbf{x})(\mathbf{W}\mathbf{x})^T}{n\text{RMS}(\mathbf{a})^2}\right),\end{aligned} \tag{10}$$

$\text{diag}(\cdot)$ 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

$$\begin{aligned}\mathbf{R}' &= \frac{1}{\delta\text{RMS}(\mathbf{a})}\left(\mathbf{I} - \frac{(\delta\mathbf{W}\mathbf{x})(\delta\mathbf{W}\mathbf{x})^T}{n\delta^2\text{RMS}(\mathbf{a})^2}\right) \\ &= \frac{1}{\delta\text{RMS}(\mathbf{a})}\left(\mathbf{I} - \frac{(\mathbf{W}\mathbf{x})(\mathbf{W}\mathbf{x})^T}{n\text{RMS}(\mathbf{a})^2}\right) \\ &= \frac{1}{\delta}\mathbf{R}.\end{aligned} \tag{11}$$

If we put the scaled term $\mathbf{R}'$ back into Eq. (10), 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.

Informally, 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 enables RMSNorm to perform implicit learning rate adaptation as in the LayerNorm (Ba et al., 2016). In particular, when the weight parameter grows twice as large, its gradient will be penalized by a factor of ½ which implicitly decays the learning rate.

5 Experiments

Like LayerNorm and unlike BatchNorm, RMSNorm performs normalization only in the same layer without any reliance on other training cases. This makes RMSNorm appropriate to different neural architectures, such as CNN, RNN and Attention, without any specific adaptation. To demonstrate the effectiveness of RMSNorm, we experiments with 4 different tasks: machine translation, image classification, image-caption retrieval and question answering. To test the efficiency of layer normalization across different implementations, we perform experiments with Tensorflow (Abadi et al., 2016), PyTorch (Paszke et al., 2017) and Theano (Theano Development Team, 2016).[1] We add RMSNorm to different models, comparing against an unnormalized baseline and LayerNorm. Unless otherwise noted, all speed-related statistics are measured on one GeForce GTX 1080 Ti GPU.

5.1 Machine Translation

Machine translation is a task of 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 (Bahdanau et al., 2014) and a self-attention based neural Transformer (Vaswani et al., 2017). 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* (Post, 2018). Byte pair encoding algorithm is applied to reduce out-of-vocabulary tokens with 32k merge operations (Sennrich et al., 2015). All models are trained based on maximum log-likelihood relaxed by label smoothing with a factor of 0.1.[2]

We first experiment with RNNSearch, a RNN-based sequence-to-sequence model. 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 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 (Nguyen and Chiang, 2017). Figure 2 illustrates the evolution of BLEU score on

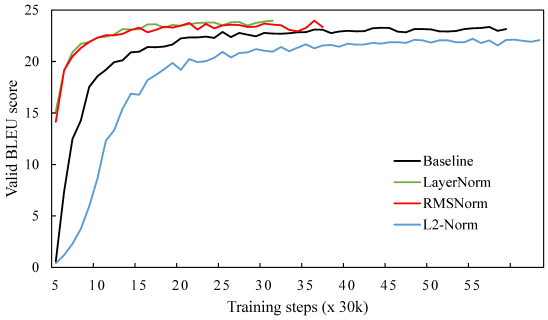


Figure 2: SacreBLEU score on newstest2013 for the RNNSearch. Models are implemented according to *Nematus* (Sennrich et al., 2017) in Tensorflow.

	Model	Test2014	Test2017	Time
Tf	Baseline	21.7	23.4	695s
	LayerNorm	**22.6**	23.6	920s
	L2-Norm	20.7	22.0	792s
	RMSNorm	22.4	**23.7**	816s (**11%**)
Th	Baseline	21.8	22.9	970s
	LayerNorm	22.3	**23.8**	1106s
	RMSNorm	**22.5**	23.2	1050s (**5%**)
Py	Baseline	22.7	**24.7**	745s
	LayerNorm	**23.2**	24.3	1131s
	RMSNorm	22.9	24.5	1030s (**9%**)

Table 2: SacreBLEU score on newstest2014 (Test2014) and newstest2017 (Test2017) for RNNSearch. "*Tf*": Tensorflow-version Nematus, "*Th*": Theano-version Nematus, "*Py*": an in-house PyTorch-based RNNSearch. "*Time*": the time in second per 1k training steps. We highlight the best results in bold, and show the speedup of RMSNorm against LayerNorm in bracket.

[1] Implementations will be released at the time of publication.

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

Model	Test2014	Test2017	Time
Baseline	-	-	583s
LayerNorm	26.6	27.7	619s
RMSNorm	**26.8**	**27.7**	603s (**2.5%**)

Table 3: SacreBLEU score on newstest2014 (Test2014) and newstest2017 (Test2017) for the Transformer. "*Time*": the time in second per 1k training steps. Best results are highlighted in bold. The number in bracket is the speedup of RMSNorm over LayerNorm. "-" indicates that we fail to train this model.

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. Our results with L2-Norm show that it fails to improve the model.[3] Results in Table 2 highlight the challenge that RNN with LayerNorm suffers from serious computational inefficiency, where LayerNorm is slower than Baseline by 15% 50% for 1k training steps depending on the toolkit and implementation. In this respect, RMSNorm performs significantly better, improving upon LayerNorm across different computational frameworks with speedups ranging from 5% to 11%.

Unlike RNNSearch, Transformer 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 perform experiments with an in-house Tensorflow implementation of the Transformer, and employ the base setting as in (Vaswani et al., 2017) 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 3 lists the results, from which we can observe the importance of normalization for Transformer, without which training fails. RMSNorm achieves BLEU scores comparable to LayerNorm, and yields a speedup of 2.5%. Compared with RNNSearch, the relative cost of normalization is lower because there are significantly fewer sequential normalization operations in Transformer.

[3]We note that Nguyen and Chiang (2017) 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.

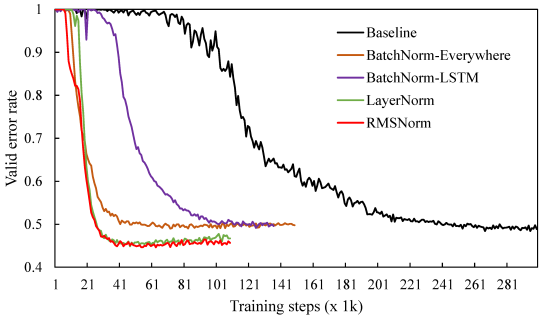


Figure 3: Error rate on validation set for the attentive reader model.

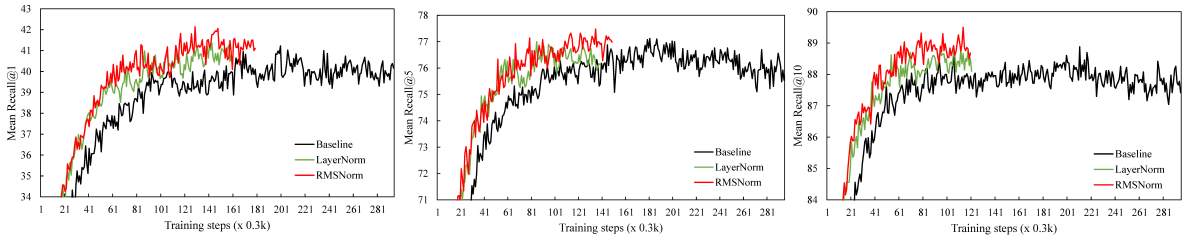
Model	Time
Baseline	275s
BatchNorm-Everywhere	304s
BatchNorm-LSTM	307s
LayerNorm	338s
RMSNorm	302s (**10%**)

Table 4: Time in seconds per 0.1k training steps for the attentive reader model. The number in bracket is the speedup of RMSNorm over LayerNorm.

5.2 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 (Hermann et al., 2015). We train a bidirectional attentive reader model proposed by Hermann et al. (2015) on the CNN corpus based on the public source code in Theano (Cooijmans et al., 2016). We adopt the *top4* setting, where each passage in the pre-processed dataset contains at most 4 sentences. In this experiment, we compare RMSNorm with the following four model variants: the vanilla model with no normalization (Baseline), the Baseline with BatchNorm applied to the LSTM alone (BatchNorm-LSTM), the Baseline with BatchNorm applied to everywhere (BatchNorm-Everywhere) and the Baseline with LayerNorm on LSTM (LayerNorm). 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 (Kingma and Ba, 2014) with a batch size of 64 and learning rate of $8e^{-5}$.

Figure 3 and Table 4 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, though BatchNorm-Everywhere performs slightly



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

Figure 4: 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 (Vendrov et al., 2015)	45.4		88.7	5.8	36.3		85.8	9.0
	OE (Vendrov et al., 2015)	46.7		88.9	5.7	37.9		85.9	8.1
	OE (Ba et al., 2016)	46.6	79.3	89.1	5.2	37.8	73.6	85.7	7.9
	OE + LayerNorm (Ba et al., 2016)	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**

Table 5: 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.

Model	Time
Baseline	5.63s
LayerNorm	7.56s
RMSNorm	6.77s (**10%**)

Table 6: Time in seconds per 0.3k training steps for the order-embedding model. The speedup of RMSNorm over LayerNorm is shown in bracket.

better. By contrast, LayerNorm and RMSNorm not only converge faster than BatchNorm, but also reach lower validation error rate. Although in Figure 3 the performance of RMSNorm and LayerNorm is comparable, RMSNorm is around 10% faster than LayerNorm as shown in Table 4. Notice that the implementation of BatchNorm is cuda-based, so time cost of BatchNorm in Table 4 can not be directly compared with others.

5.3 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. (2015) on the Microsoft COCO dataset (Lin et al., 2014) using their public source code in Theano[4]. In OE, sentences are encoded through a GRU-based RNN (Cho et al., 2014) and images are represented by the output of a pretrained VGGNet (Simonyan and Zisserman, 2014). OE treats the caption-image pairs as a two-level partial order, and trains the joint model using the pairwise ranking loss (Kiros et al., 2014).

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. We compare RMSNorm with two models: one without any normalization (Baseline) and one with LayerNorm. 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.

Figure 4 shows the R@K curve on validation set after every 300 training steps, and Table 5 lists the final test results. We summarize R@1, R@5, and R@10 metrics for evaluation, and across all these metrics, RMSNorm and LayerNorm consistently outperform the Baseline in terms of model convergence as shown in Figure 4. We also observe that on the validation set, RMSNorm slightly exceeds LayerNorm with respect to recall value. For the final test results as shown in Table 5, both RMSNorm and LayerNorm improve the model

[4]https://github.com/ivendrov/order-embedding

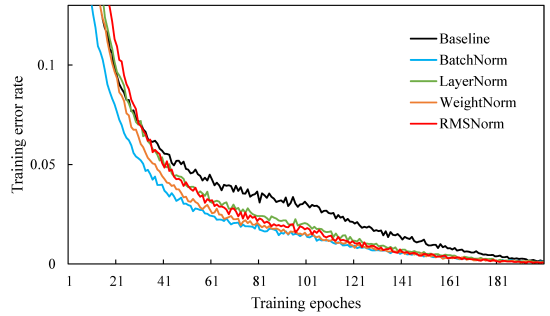


Figure 5: Training error rate for the ConvPool-CNN-C model.

Model	Test Error	Time
Baseline	8.96%	51s
BatchNorm	8.25%	66s
WeightNorm	8.28%	53s
LayerNorm	10.49%	72s
RMSNorm	8.83%	61s (**15%**)

Table 7: Test error rate and time in seconds per training epoch for the ConvPool-CNN-C model. The speedup of RMSNorm over LayerNorm is shown in bracket.

performance, reaching higher recall values (except LayerNorm on R@5) and lower mean rank, though RMSNorm reveals better generalization than LayerNorm. Table 6 further highlights the speed measure for all models. In particular, RMSNorm accelerates training speed by 10% compared with LayerNorm.

5.4 CIFAR-10 Classification

CIFAR-10 is a supervised image classification task, with 10 different classes. We train a modified version of the ConvPool-CNN-C architecture (Krizhevsky and Hinton, 2009), and follow the same experimental protocol as Salimans and Kingma (2016), using the public source code in Theano[5]. We compare RMSNorm with four different variants: the original model without any normalization (Baseline), the Baseline with LayerNorm, the Baseline with BatchNorm and the Baseline with WeightNorm. We apply layer normalization to the width and height dimensions of image representation, and perform gain scaling and bias shifting on the channel dimension. We train all models using Adam optimizer with a batch size of 100. Learning rate is set to 0.0003 for Baseline and 0.003 for others following (Salimans and Kingma, 2016).

We show the training error curve in Figure 5, and list the test error and training time in Table 7. Models enhanced with a normalization technique converge faster than Baseline, among which BatchNorm performs the best. Similar to previous observation (Ba et al., 2016), we also find that layer normalization works worse than BatchNorm and WeightNorm for image processing. In particular, although LayerNorm outperforms Baseline by shorting model convergence, it fails to generalize to the test set, degenerating the test error by 1.53%. In this respect, RMSNorm shows better generalization, surpassing the Baseline by 0.013%. In terms of speed, RMSNorm saves about 15% training time compared to LayerNorm. The success of RMSNorm over LayerNorm further suggests that RMSNorm can act as an alternative to LayerNorm with comparable performance and higher computational efficiency.

5.5 Conclusion and Future Work

This paper presents RMSNorm, a novel normalization approach that normalizes the summed inputs according to the root mean square statistic. RMSNorm is more computational efficient than LayerNorm because it avoids the sequential computation of mean and variance statistics. RMSNorm imposes no additional dependence among different training cases, thus can be easily applied to different model architectures. Experiments on both image-based and language-based tasks show that RMSNorm is comparable to LayerNorm in quality, but accelerates the running speed. Actual speed improvements depend on the framework and relative computational cost of other components, and we empirically observed speedups of 2.5%$\sim$15% across different models and implementations. Our efficiency improvement come from simplifying the computation and reducing sequential dependences in the computation graph, and we thus expect them to be orthogonal to other means of increasing training speed, such as low-precision arithmetics and GPU kernel fusion.

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.

[5]https://github.com/TimSalimans/weight_norm

References

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. 2016. 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.

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

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

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

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. 2018. 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.

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

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

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

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

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

Sergey Ioffe and Christian Szegedy. 2015. 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.

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

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

A. Krizhevsky and G. Hinton. 2009. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*.

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

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

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

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

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

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

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

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

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. 2017. 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.

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

Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. 2018. 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.

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

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

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

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

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

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

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

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