L1-Norm Batch Normalization for Efficient Training of Deep Neural Networks

Journal:	*IEEE Transactions on Neural Networks and Learning Systems*
Manuscript ID	TNNLS-2018-P-9126.R1
Manuscript Type:	Paper
Date Submitted by the Author:	27-Jun-2018
Complete List of Authors:	Wu, Shuang; Tsinghua University, Department of Precision Instrument Li, Guoqi Deng, Lei; University of California Santa Barbara Liu, Liu; University of California Santa Barbara Wu, Dong; Tsinghua University, Institute of Microelectronics Xie, Yuan; University of California Santa Barbara Shi, Luping
Keywords:	L1-norm, batch normalization, deep neural network, discrete online learning, mobile intelligence

L1-Norm Batch Normalization for Efficient Training of Deep Neural Networks

Shuang Wu*, Guoqi Li*, Lei Deng, Liu Liu, Dong Wu, Yuan Xie, and Luping Shi†

***Abstract*—Batch normalization (BN) has recently become a standard component for accelerating and improving the training of deep neural networks (DNNs). However, BN brings in additional calculations, consumes more memory, and significantly slows down the training iteration. Furthermore, the nonlinear square and sqrt operations in the normalization process impede low bit-width quantization techniques, which draws much attention to the deep learning hardware community. In this work, we propose an L1-norm batch normalization (L1BN) with only linear operations in both forward and backward propagation during training. L1BN is approximately equivalent to the conventional L2-norm BN (L2BN) by multiplying a scaling factor that equals to $\sqrt{\frac{\pi}{2}}$. Experiments on various convolutional neural networks (CNNs) and generative adversarial networks (GANs) reveal that L1BN can maintain the same accuracy and convergence rate as L2BN, but with higher computational efficiency. In real ASIC synthesis with reduced resources, L1BN achieves 25% speedup and 37% energy saving compared to the original L2BN. Our hardware-friendly normalization method not only surpasses L2BN in speed, but also simplifies the design of deep learning accelerators. Last but not the least, L1BN promises a fully quantized training of DNNs, which empowers future AI applications on mobile devices with transfer and continual learning capability.**

***Index Terms*—L1-norm, batch normalization, deep neural network, discrete online learning, mobile intelligence**

I. Introduction

Nowadays, deep neural networks (DNNs) [1] are rapidly permeating into various AI applications, for instance, computer vision [2], speech recognition [3], machine translation [4], Go game [5] and multi-model tasks across them [6]. However, training DNNs is complicated and needs elaborate tuning of hyperparameters, especially on large datasets with diverse samples and thousands of categories. Therefore, the distribution of mini-batch samples shifts stochastically during the training process, which will affect the subsequent layers successively, and eventually make the network's outputs and gradients vanish or explode. This internal covariate shift phenomenon [7] leads to slower convergence, requires careful adaption of learning rate and appropriate initialization of network parameters [8].

To address the problem, batch normalization (BN) [7] has been proposed to facilitate training by explicitly normalizing inputs of each layer to have zero mean and unit variance. Under the guarantee of appropriate distribution, the difficulties of annealing learning rate and initializing parameters are now reduced. Besides, the inherent randomization incurred by the mini-batch statistics serves as a regularizer, which is a better alternative to dropout [9]. Most generative adversarial networks (GANs) also rely on BN in both the generator and the discriminator [10], [11], [12]. With batch normalization, deep generator can start with a normal training process, as well as avoid mode collapse [11] that is a common failure observed in GANs. BN is so helpful in training DNNs that it has almost become a standard component together with the rectifier nonlinearity (ReLU) [13], not only in most deep learning models [14], [15], [16], but also in the neural network accelerator community [17], [18].

Inspired by BN, weight normalization [19] re-parameterizes the incoming weights by their L2-norm. Layer normalization [20] replaces the statistics of a training batch with a single training case, which does not re-parameterize the network. Both methods eliminate the dependencies among examples in a mini-batch and can be applied successfully to recurrent models. In batch re-normalization [21], an affine transformation is proposed to ensure that the training and inference models generate the same outputs that depend on individual example rather than the entire mini-batch.

However, batch normalization usually causes considerable overheads in both the forward and backward propagations. Recently in the field of convolutional neural networks (CNNs), there is a trend towards replacing standard convolution with bottleneck pointwise convolution [22], group convolution [23] or depthwise convolution [24]. Although the numbers of parameters and multiply-accumulate operations (MACs) have been reduced in these models during inference, the feature maps (channels) in each convolution layer have been expanded. Therefore, the computation overheads of BN layers are getting more expensive during training. In other words, a compact model may require more training time to reach an optimal convergence [16].

On the one hand, the additional calculations in batch normalization are costly, especially for resource-limited ASIC devices. When it comes to online learning, i.e., deploying the training process onto terminal devices, the salient resource problem has challenged the extensive application of DNNs in real scenarios. On the other, the square and sqrt operations in-

S. Wu and G. Li contribute equally to this work. S. Wu, G. Li and L. Shi are with the Department of Precision Instrument, Center for Brain Inspired Computing Research, Tsinghua University; D. Wu is with the Institute of Microelectronics, Tsinghua University, Beijing 100084, China (e-mail: wus15@mails.tsinghua.edu.cn; liguoqi@mail.tsinghua.edu.cn; dongwu@tsinghua.edu.cn; lpshi@mail.tsinghua.edu.cn). L. Deng, L. Liu and Y. Xie are with Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA93106, USA (e-mail: leideng@ucsb.edu; liu_liu@ucsb.edu; yuanxie@ucsb.edu)

The work was partially supported by National Science Foundation of China (61603209) and the SuZhou-Tsinghua innovation leading program (2016SZ0102). † Corresponding to: lpshi@mail.tsinghua.edu.cn

troduce strong nonlinearity and make it difficult to employ low bit-width quantization. Although many quantization methods have been proposed to reduce the memory costs and accelerate the computation [25], [26], [27], the BN layers are simply avoided [28] or maintained in float32 precision.

In this work, we introduce an L1-norm BN (L1BN), where the L2-norm variance for mini-batch samples is substituted by an L1-norm "variance". By deriving the chain rule for differentiating BN layer in the backpropagation training, we find that the costly and nonlinear square and sqrt operations can be replaced by hardware-friendly sign and absolute operations. We prove that L1BN is approximately equivalent to the L2BN by multiplying a scaling factor that equals to $\sqrt{\frac{\pi}{2}}$. To evaluate the proposed method, various experiments have been conducted with CNNs on datasets including Fashion-MNIST [29], SVHN [30], CIFAR [31] and ImageNet [32], as well as GANs on CIFAR and LSUN-Bedroom [33]. Results indicate that L1BN is able to achieve comparable accuracy and convergence rate, but with higher computational efficiency. Cost comparisons of basic operations are estimated through ASIC synthesis, L1BN is able to obtain 25% speedup and 37% energy saving. Other hardware resources, e.g., silicon area and cost, can be reduced as well. We believe that this new normalization method can be an efficient alternative to speed up training on dedicated devices, and promises a hardware-friendly learning framework where all the operations and operands are quantized to low bit-width precision.

II. Problem Formulation

Stochastic gradient descent (SGD), known as incremental gradient descent, is a stochastic and iterative approximation of gradient descent optimization. SGD minimizes an objective function with differentiable parameters Θ:

$$\Theta = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, \Theta) \tag{1}$$

where x_i for $i = 1, 2, \ldots, N$ is the training set containing totally N samples, $\sum_{i=1}^{N} \ell(x_i, \Theta)$ is the empirical risk summarized by the risk at each sample $\ell(x_i, \Theta)$. Considering SGD with a mini-batch of m samples, then the gradient of the loss function ℓ can be simply approximated by processing the gradient of m samples and update with the average value:

$$\Theta \leftarrow \Theta - \eta \cdot \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \ell(x_i, \Theta)}{\partial \Theta} \tag{2}$$

where η is the learning rate.

While SGD with mini-batch is simple and effective, it requires careful tuning of the model hyperparameters, especially the learning rate used in the optimizer, as well as the initial values of the model parameters. Otherwise, the outputs of neural networks will be stuck in an inappropriate interval for the following activation (nonlinearity) functions, e.g., [8, 10] for $\mathrm{sigmoid}(x)$, where most values will be saturated. Then the gradients of these values are vanishing, resulting in slow convergence and local minimum. Another issue is that the inputs of each layer are affected by the parameters of all preceding layers, so that small updates of the parameters will accumulate and amplify once the network becomes deeper. This leads to an opposite problem that the outputs and gradients of network are prone to explode. Thus, SGD slows down the training by requiring lower learning rate and careful parameter initialization, and makes it notoriously hard to train deep models with saturating nonlinearities, e.g., $\mathrm{sigmoid}(x)$, $\tanh(x)$, which affects the networks' robustness and challenges its extensive applications.

In [7], the authors refer to this phenomenon as internal covariate shift, and address this problem by explicitly normalizing inputs. This method draws its strength from making normalization a part of the model architecture and performing the normalization across each mini-batch, which is termed as "Batch normalization" (BN). BN ensures that the distribution of pre-activations (values fed into activation) remains stable as the values propagate across layers in deep network, and then the SGD optimizer will be less likely to get stuck in the saturated regime or explode to non-convergence, thus allows us to use higher learning rates and be less careful about parameter initialization.

III. Conventional L2BN

Specifically, BN transforms each dimension in scalar feature independently by making it have zero mean and unit variance. For a layer with c-dimensional inputs $x = \{x^{(1)}, \ldots, x^{(k)}, \ldots, x^{(c)}\}$, each dimension is normalized before fed into the following nonlinearity function

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}} \tag{3}$$

where the expectation $\mathrm{E}[x^{(k)}]$ and variance $\mathrm{Var}[x^{(k)}]$ are computed over all training samples.

Usually for a mini-batch $\mathcal{B}$ containing m samples, we use $\mu_{\mathcal{B}}$ and $\sigma^2_{\mathcal{B}} + \epsilon$ to estimate $\mathrm{E}[x^{(k)}]$ and $\mathrm{Var}[x^{(k)}]$, respectively, which gives that

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \tag{4}$$

where ϵ is a sufficiently small positive parameter, e.g., 1e-5, for keeping numerical stability. The mini-batch mean $\mu_{\mathcal{B}}$ and variance $\sigma^2_{\mathcal{B}}$ is given by

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{5}$$

and

$$\sigma^2_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \tag{6}$$

It is easy to see that $\sigma^2_{\mathcal{B}}$ is calculated based on the L2-norm of the statistics.

Note that this forced normalization may change and hurt the representation capability of a layer. To guarantee that the transformation inserted into the network can represent the identity function, a trainable linear layer that scale and shift the normalized values are introduced:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)} \tag{7}$$

where γ and β are parameters to be trained along with the other model parameters.

During training, we need to back-propagate the gradient of loss ℓ through this transformation, as well as compute the gradients with respect to the parameters γ and β, the chain rule is derived as follows:

$$\begin{aligned}
\frac{\partial \ell}{\partial \hat{x}_i} &= \frac{\partial \ell}{\partial y_i} \cdot \gamma \\
\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} &= \sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2} (\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2} \\
\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} &= \sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \\
\frac{\partial \ell}{\partial x_i} &= \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m} \\
\frac{\partial \ell}{\partial \gamma} &= \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \hat{x}_i \\
\frac{\partial \ell}{\partial \beta} &= \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}
\end{aligned} \tag{8}$$

Although BN accelerates the convergence of training DNNs, it requires additional calculations and generally slows down the iteration process by a large margin. Table I shows the time overheads when training with or without BN on conventional DNNs. All models are built on Tensorflow [34] and trained with one or two Titan-Xp GPUs. Section V-A will detail the implementation and training hyperparameters. For BN, we apply the widely used built-in version: fused batch normalization, which combines multiple required operations into a single kernel on GPU and can accelerate the normalization process [35]. Even so, BN brings in about 30%-50% extra training time. Note that L2-norm is applied to estimate $\sigma_{\mathcal{B}}$, we term this method as the L2-norm batch normalization (L2BN). To address this issue, we propose an L1-norm batch normalization (L1BN) in this work.

TABLE I
TRAINING TIME (MILLISECOND) PER IMAGE WITH OR WITHOUT BATCH NORMALIZATION, AND THE RATIO OF EXTRA TIME

Model	no BN	with BN	+time(%)
ResNet-110	0.46	0.68	47.8
DenseNet-100	1.08	1.58	46.3
MobileNet	1.55	1.95	25.8

IV. PROPOSED L1BN

Our idea is simple but effective, which applies the L1-norm to estimate $\sigma_{\mathcal{B}}$. The L1BN is formulated as

$$y_i = \gamma \cdot \hat{x}_i + \beta \tag{9}$$

and

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}} + \epsilon} \tag{10}$$

where γ, β, $\mu_{\mathcal{B}}$ and ϵ are identical to that in L2BN, and $\sigma_{\mathcal{B}}$ is a term calculated by the L1-norm of mini-batch statistics:

$$\sigma_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^{m} |x_i - \mu_{\mathcal{B}}| \tag{11}$$

The motivation of using L1-norm is that the L1-norm "variance" is linearly correlated with the L2-norm variance if the inputs have a normal distribution, which is commonly satisfied and observed in conventional networks [8].

Theorem 1: For a normally distributed random variable X with variance σ^2, define a random variable Y such that $Y = |X - \mathrm{E}(X)|$, we have

$$\frac{\sigma}{\mathrm{E}(|X - \mathrm{E}(X)|)} = \sqrt{\frac{\pi}{2}} \tag{12}$$

Proof: Note that $X - \mathrm{E}(X)$ belongs to a normal distribution with zero mean μ and variance σ^2. Then $Y = |X - \mathrm{E}(X)|$ has a folded normal distribution or half-normal distribution [36]. Denote μ_Y as the mean of Y, we have

$$\mu_Y = \sqrt{\frac{2}{\pi}} \sigma e^{\frac{-\mu^2}{2\sigma^2}} \tag{13}$$

based on the statistical property of half-normal distribution. As $\mu = 0$, we can obtain that

$$\frac{\sigma}{\mu_Y} = \sqrt{\frac{\pi}{2}} \tag{14}$$

Remark 1: If the inputs of the BN layer obey a normal distribution, by denoting the standard derivation of the inputs as σ_{L_2}, and the L1-norm term in Equation 11 as σ_{L_1}, we have

$$\sigma_{L_2} = \sqrt{\frac{\pi}{2}} \cdot \sigma_{L_1} \tag{15}$$

Let γ_{L_2} and γ_{L_1} be the scale parameters in Equation 7 and 9 for L2BN and L1BN, respectively. To keep the outputs of the two methods identical, ideally we have

$$\gamma_{L_2} = \sqrt{\frac{\pi}{2}} \cdot \gamma_{L_1} \tag{16}$$

□

The above remark is validated in Section V-C. To implement backpropagation when L1-norm is involved, the chain rule is derived as follows:

$$\begin{aligned}
\frac{\partial \ell}{\partial \hat{x}_i} &= \frac{\partial \ell}{\partial y_i} \cdot \gamma \\
\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}} &= \sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{(\sigma_{\mathcal{B}} + \epsilon)^2} \\
\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} &= \sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{-1}{\sigma_{\mathcal{B}} + \epsilon} \\
\frac{\partial \ell}{\partial x_i} &= \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}} \cdot \frac{1}{m} \left\{ \mathrm{sgn}(x_i - \mu_{\mathcal{B}}) - \frac{1}{m} \sum_{j=1}^{m} \mathrm{sgn}(x_j - \mu_{\mathcal{B}}) \right\} \\
&\quad + \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{1}{\sigma_{\mathcal{B}} + \epsilon} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}
\end{aligned} \tag{17}$$

$\frac{\partial \ell}{\partial \gamma}$ and $\frac{\partial \ell}{\partial \beta}$ has exactly the same form as in Equation 8. It

is obvious that

$$\operatorname{sgn}(\hat{x}_i) = \operatorname{sgn}(x_i - \mu_{\mathcal{B}}) \tag{18}$$

Let

$$\begin{aligned} \mu(\tfrac{\partial \ell}{\partial \hat{x}_i}) &= \tfrac{1}{m}\textstyle\sum_{i=1}^{m} \tfrac{\partial \ell}{\partial \hat{x}_i} \\ \mu(\tfrac{\partial \ell}{\partial \hat{x}_i} \cdot \hat{x}_i) &= \tfrac{1}{m}\textstyle\sum_{i=1}^{m} (\tfrac{\partial \ell}{\partial \hat{x}_i} \cdot \hat{x}_i) \end{aligned} \tag{19}$$

Then, by substituting Equations 10 and 11 into Equation 17, we can obtain that

$$\begin{aligned} \tfrac{\partial \ell}{\partial x_i} = \tfrac{1}{\sigma_{\mathcal{B}}+\epsilon}\{ &\tfrac{\partial \ell}{\partial \hat{x}_i} - \mu(\tfrac{\partial \ell}{\partial \hat{x}_i}) \\ &- \mu(\tfrac{\partial \ell}{\partial \hat{x}_i} \cdot \hat{x}_i) \cdot [\operatorname{sgn}(\hat{x}_i) - \mu(\operatorname{sgn}(\hat{x}_i))]\} \end{aligned} \tag{20}$$

Note that square and sqrt operations can be avoided in both the forward and backward propagation when L1BN is involved. As shown in Section V, L1BN is approximately equivalent to L2BN in convergence rate and final accuracy, but with better computational efficiency.

A. L1BN in MLP and CNN

As with L2BN, L1BN can be applied before nonlinearities in deep networks including but not limited to multi-layer perceptrons (MLPs) and CNNs. The only difference is that the L1-norm is applied to calculate σ. As shown in Equation 15, in order to compensate the difference between two deviations, we use the strategy as follows:

1) When the scale factor γ is not applied in batch normalization, we multiply σ_{L_1} by $\sqrt{\frac{\pi}{2}}$ in Equation 11 to approximate σ_{L_2}.
2) When the scale factor γ is applied in batch normalization, we just use σ_{L_1} and let the trainable parameter γ_{L_1} "learn" to compensate the difference automatically through backpropagation training.

Other normalization processes of L1BN are just exactly the same as that in L2BN. For the fully-connected layer in MLPs, each channel is normalized based on m samples of a training mini-batch. While for the convolution layer in CNNs, elements at different spatial locations of individual feature map (channel) are normalized with the same mean and variance. Let $\mathcal{B}$ be the set of all values across both the mini-batch and spatial locations, so for a mini-batch containing m samples with height h and width w, the effective number of data points for each channel is $|\mathcal{B}| = mhw$. Then, parameters γ and β can be learned for each feature map, rather than each spatial location. The algorithm for L1BN is presented in Algorithm 1.

Algorithm 1 Training a L1BN layer with statistics $\{\mu, \sigma\}$, moving-average momentum α, trainable linear layer parameters $\{\gamma, \beta\}$, learning rate η, inference with one sample

Require: a mini-batch of pre-activation samples for training $\mathcal{B} = \{x_1, \ldots, x_m\}$, one pre-activation sample for inference $\mathcal{I} = \{x_{\text{inf}}\}$

Ensure: updated parameters $\{\mu, \sigma, \gamma, \beta\}$, normalized pre-activations $\mathcal{B}^{\text{N}}_{\text{tr}} = \{x^{\text{N}}_1, \ldots, x^{\text{N}}_m\}$, $\mathcal{I}^{\text{N}}_{\text{inf}} = \{x^{\text{N}}_{\text{inf}}\}$

1. Training with mini-batch $\mathcal{B}$:

1.1 Forward:

$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ //mini-batch mean

$\sigma_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} |x_i - \mu_{\mathcal{B}}|$ //mini-batch L1 variance

$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}+\epsilon}$ //normalize

$x^{\text{N}}_i \leftarrow \gamma\hat{x}_i + \beta \equiv \text{L1BN}_{\text{tr}}(x_i)$ //scale and shift

1.2 Backward:

$\gamma \leftarrow \gamma - \eta\frac{\partial \ell}{\partial \gamma}$ //update parameters

$\beta \leftarrow \beta - \eta\frac{\partial \ell}{\partial \beta}$ //update parameters

1.3 Update statistics:

$\mu \leftarrow \alpha\mu + (1-\alpha)\mu_{\mathcal{B}}$ //moving-average

$\sigma \leftarrow \alpha\sigma + (1-\alpha)\sigma_{\mathcal{B}}$ //moving-average

2. Inference with sample $\mathcal{I}$:

$x^{\text{N}}_{\text{inf}} \leftarrow \frac{\gamma}{\sigma+\epsilon} \cdot x_{\text{inf}} + (\beta - \frac{\gamma\mu}{\sigma+\epsilon}) \equiv \text{L1BN}_{\text{inf}}(x_{\text{inf}})$

V. Experiments

A. L1BN on classification tasks

In order to verify the equivalence between L1-norm batch normalization and it's original L2-norm version, we test both methods in various image classification tasks. In these tasks, we parameterize model complexity and task complexity, then apply two-dimensional demonstrations with different CNN architectures on multiple datasets. For each demonstration, all the hyperparameters in L2BN and L1BN remain the same, the only difference is the use of L2-norm σ_{L_2} or L1-norm σ_{L_1}. In the following experiments, SGD optimizer with momentum 0.9 is the default configuration. As suggested in [7], the trainable linear layer is applied in each batch normalization since it introduces very few computation and parameters (γ,β) but can improve the final performance. Therefore, according to the strategy mentioned in Section IV-A, we do not multiply $\sqrt{\frac{\pi}{2}}$ in the calculation of σ_{L_1}.

1) **Simple tasks with shallow models:** Fashion-MNIST [29] is a MNIST-like fashion product database that contains 70k grayscale 28×28 images and preferably represents modern computer vision tasks. We use LeNet-5 [37] network, and train for totally 90 epochs with mini-batch size of 128. The learning rate η is set to 0.1 and divided by 10 at epoch 30 and epoch 60. As for SVHN [30] dataset, we use a VGG-like network with totally 7 layers [28]. Input images are scaled and biased to the range of $[-1, +1]$, the total number of training epochs is reduced to 40. The learning rate η is set to 0.1 and divided by 10 at epoch 20 and epoch 30.

2) **Moderate tasks with very deep models:** Identity connections [14] and densely concatenations [15] are proven to be quite efficient in very deep CNNs with much fewer parameters. We test the effects brought by L1-norm in these structures on CIFAR [31] datasets. A DensetNet-100 [15] network is trained on CIFAR-100, as for the bottleneck architecture [22], a ResNet-110 [14] and a Wide-DenseNet (L = 40, K = 48) [15] are trained on CIFAR-10. We follow the data augmentation in [38] for training: 4 pixels are padded on each side, and a 32×32 patch is randomly cropped from the padded

image or its horizontal flip. For testing, only single view of the original 32×32 image is evaluated. Learning rates and annealing methods are the same as that in [14].

3) **Complicated tasks with deep wide models:** For ImageNet dataset with 1000 categories [32], we adopt AlexNet [2] model but remove dropout and replace the local response normalization layers with L1BN or L2BN. Images are firstly rescaled such that the shorter sides are of length 256, and then cropped out centrally to 256×256. For training, images are then randomly cropped to 224×224 and horizontally flipped. For testing, the single center crop in the validation set is evaluated. The model is trained with mini-batch size of 256 and totally 80 epochs. Weight decay is set to 5e-4, learning rate η is set to 1e-2 and divided by 10 at epoch 40 and epoch 60.

4) **Complicated task with deep slim model:** Recently compact convolutions such as group convolution [23] and depthwise convolution [24] draws extensive attention with equal performance but fewer parameters and MACs. Therefore, we further reproduce MobileNet [16] and evaluate L1BN on ImageNet dataset. At this time, weight decay decreases to 4e-5, learning rate η is set to 0.1 initially and linearly annealed to 1e-3 after 60 epochs. We apply the Inception data argumentation defined in TensorFlow-Slim image classification model library [39]. The training is performed on two Titan-Xp GPUs and the population statistics $\{\mu, \sigma\}$ for batch normalization are updated according to the calculations from single GPU, so the actual batchsize for BN is 128.

TABLE II
TEST OR VALIDATION ERROR RATES (%) FOR L1BN AND L2BN

Dataset	Model	L2BN	L1BN
Fashion	LeNet-5	7.66	**7.62**
SVHN	VGG-7	1.93	**1.92**
CIFAR-10	ResNet-110	6.24	**6.12**
CIFAR-10	Wide-DenseNet	**4.09**	4.13
CIFAR-100	DenseNet-100	22.46	**22.38**
ImageNet	AlexNet	42.5	**42.1**
ImageNet	MobileNet	**29.5**	29.7

The main results are summarized in Table II, all error rates are the average of best accuracies in multiple repeated experiments. In addition, the training curves of two methods using ResNet-110 model on CIFAR-10 dataset are shown in Figure 2. We have two major observations:

From the perspective of the final results, L1BN is equal to or slightly surpasses the original L2BN on many occasions. We argue that this performance improvement is caused by the enlarged normalized statistics $\hat{x}_i$: according to Equation 15, σ_{L_1} is smaller than σ_{L_2}, which results in a 1.25× amplification of $\hat{x}_i$. As mentioned above, the inherent randomization incurred by the mini-batch statistics can serve as a regularizer. Although the following linear layer of BN may "learn" to alleviate the scaling by adjusting parameter γ during training, L1-norm intuitively enhances this randomization and may further regularize large model and finally perform better accuracy. As for the MobileNet result, there are very few parameters in its depthwise filters, so this compact model has less trouble with overfitting. On this occasion, L1-norm regularization may hurt the accuracy by a small margin.

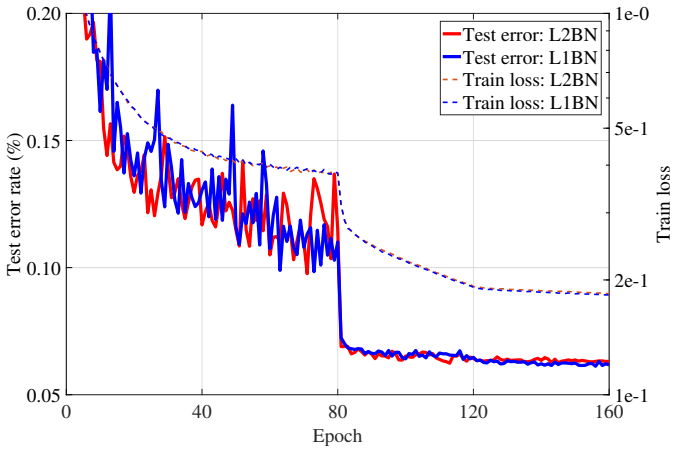

Fig. 1. Training curves of ResNet-110 network on CIFAR-10 dataset using L2BN or L1BN. Training losses contain the sum of weight decay losses.

From the perspective of training curves, the test error of L1BN is a bit more unstable at the beginning. Although this can be improved by more accurate initialization: initialize γ from 1.0 (L2-norm) to $\sqrt{\frac{2}{\pi}} \approx 0.8$ (L1-norm), or follow the first strategy mentioned in Section IV-A, the two loss curves almost overlap completely. Therefore, we directly embrace this instability and let networks find their way out. No other regular distinctions between two optimization processes are noticed in our CNN experiments.

B. L1BN on generative tasks

Since the training of GAN is a zero-sum game between two neural networks without guarantee of convergence, most GAN implementations rely on BN in both the generator and the discriminator to help stabilize training, and prevent the generator from collapsing all generated images to certain failure patterns. In this case, the qualities of generated results will be more sensitive to any numerical differences. Therefore, we apply L1BN to two GAN tests to prove its effectiveness.

Firstly, we generate 32×32 CIFAR-10 images using DCGAN [10] and WGAN-GP [40]. In the reproduction of DCGAN, we use the non-saturating loss function proposed in [41]. As for WGAN-GP, the network remains the same except that no BN is applied to discriminators as suggested in [40]. Besides, other training techniques, e.g., wasserstein distance, gradient penalty, are adopted to improve the training process. Generated images are evaluated by the widely used metric Inception Score (IS) introduced in [11]. Since IS results fluctuate during training, we evaluate sampled images after every 10k generator iterations, and report both the average score of the last 20 evaluations (AVG) and the overall best score (BEST).

Fig. 2. Bedroom images (128×128) generated by a DCGAN generator using L2BN (left) or L1BN (right), the hyperparameters and training techniques are the same as described in WGAN-GP.

Table III shows that L1BN is still equivalent to L2BN in such hyperparameter-sensitive adversarial occasion. Note that in our implementations, some training techniques, hyperparameters and network structures are not the same as described in the referred results. Incorporating these techniques might further bridge the performance gaps.

TABLE III
UNSUPERVISED INCEPTION SCORES ON CIFAR-10 (LARGER VALUES REPRESENT FOR BETTER GENERATION QUALITY).

Method	BN	AVG	BEST
DCGAN (in [12])	L2	6.16 ± 0.07	
Improved GAN [11]	L2	6.86 ± 0.06	
WGAN-GP [40]	L2	7.86 ± 0.07	
DCGAN (ours)	L2	7.07 ± 0.08	7.39 ± 0.08
	L1	**7.18 ± 0.09**	**7.70 ± 0.08**
WGAN-GP (ours)	L2	**6.89 ± 0.12**	**7.02 ± 0.14**
	L1	6.86 ± 0.11	6.97 ± 0.11

Secondly, we perform experiments on LSUN-Bedroom [33] high-resolution image generation task. The original DCGAN only deals with image size 64×64, so an additional upsample de-convolution and downsample convolution layer is applied to DCGAN's generator and discriminator, respectively, to produce higher resolution 128×128 images. In order to stabilize training and avoid mode collapse, we combine DCGAN architecture with WGAN-GP training methods. Since LSUN-Bedroom dataset only has single class (bedroom) of images, the generated images can not be evaluated by Inception Score that requires multiple categories of samples. Figure 2 just intuitively shows generated images after 300k generator iterations. Still, both methods generate comparable samples containing detailed textures, and we observe no significant difference in artistic style.

C. Layerwise and channelwise comparasion

We further offer a layerwise and channelwise perspective to demonstrate the equivalence between L1-norm and L2-norm scaling. After training a ResNet-110 network on CIFAR-10 for 100 epochs, we fix all the network parameters and trainable linear layer parameters $\{\gamma, \beta\}$, then feed the model with a batch of test images. Since the numerical difference between L1BN and L2BN will accumulate among layers, we guarantee that the inputs of each layer for both normalization methods are the same. But within that layer, the standard deviation (L2-norm) σ_{L_2} and the L1-norm deviation σ_{L_1} of channel outputs are calculated simultaneously.

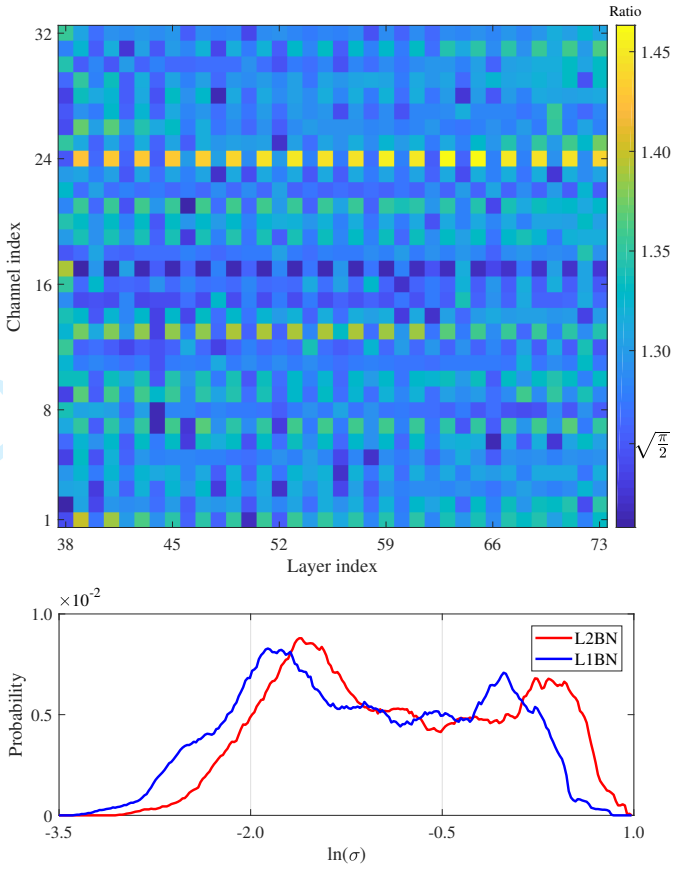

Fig. 3. Top: Colormap of layerwise and channelwise ratios $\sigma_{L_2}/\sigma_{L_1}$ averaged across 100 mini-batches. Bottom: Probability histograms of σ_{L_2} and σ_{L_1}, the axis x is logarithmic scaled.

TABLE IV
TOTAL NUMBERS AND COMPUTATIONAL OVERHEADS OF BASIC ARITHMETIC OPERATIONS IN ASIC SYNTHESIS. HERE M DENOTES THE TOTAL NUMBER OF INTRA-CHANNEL DATA POINTS WITHIN ONE BATCH FOR SIMPLICITY, AND C DENOTES THE NUMBER OF CHANNELS.

type	overhead	SIGN	ABS	ADD	SUB	MUL	DIV	SQUARE	SQRT
Quantity	L2BN	0	0	5mc+c	2mc	4mc+2c	2c	mc	c
	L1BN	mc	mc	5mc+c	2mc	3mc+2c	2c	0	0
float32	time(ns)	0.00	0.71	43.00	43.00	35.83	99.49	27.23	99.50
	power(μW)	0.00	0.01	29.50	29.60	97.20	254.00	39.10	59.00
	area(μm^2)	0.00	3.39	8095	8097	19625	36387	11539	11280
int8	time(ns)	0.00	3.47	5.25	5.45	11.46	24.90	8.54	7.60
	power(μW)	0.00	0.59	0.77	0.87	9.55	5.90	3.64	0.82
	area(μm^2)	0.00	256	272	306	2319	1762	1241	277

In Remark 1, it is pointed out that, when the inputs obey a normal distribution, ideally L1BN and L2BN are identical if we multiply $\sqrt{\frac{\pi}{2}}$ in Equation 11 for L1BN. In other words, if all the other conditions are the same, the standard derivation σ_{L_2} is $\sqrt{\frac{\pi}{2}}$ multiple of σ_{L_1} in each layer. In Figure 3, the average ratios $\sigma_{L_2}/\sigma_{L_1}$ confirms this hypothesis. As shown in the colormap of Figure 3, the ratios of intermediate layers (from layer 38 to layer 73, totally 1152 channels) are very close to the value of $\sqrt{\frac{\pi}{2}} \approx 1.25$. Also, the histograms of σ_{L_2} and σ_{L_1} are similar except for a phase shift in the logarithmic axis x , which is consistent with Theorem 1 and Remark 1.

D. Computational efficiency of L1BN

Via replacing the L2-norm variance with the L1-norm "variance", L1BN improves the computational efficiency, especially on ASIC devices with reduced resources. In Section IV-A, we have pointed out that the effective number of data points $|\mathcal{B}|$ for normalization equals to m and mhw for a fully-connected layer and a convolution layer, respectively. Since spatial statistics are always coupled with mini-batch statistics, we use m to denote the total number of intra-channel data points within one batch for simplicity, and c denotes the number of channels. In the first two rows of Table IV, we count the total numbers of basic arithmetic operations according to Equations 4, 8, 11, 17. The major improvements come from the reductions of multiplication, square and sqrt operations.

Next, the computational overheads of basic arithmetic operations are estimated on the SMIC LOGIC013 RVT process with datatype float32, int32, float16, int16, float8 and int8. We only show the results of float32 and int8 in Table IV. Compared to square and sqrt operations, sign and absolute operations are quite efficient in speed and power, and save silicon area and cost. In Figure 4, the time and power consumption of L1BN and L2BN are estimated. L1BN can averagely achieve 25% speedup and 37% power saving in practice. We can conclude that by reducing the costly multiplication, square and sqrt operations in the L2BN layer, L1BN is able to improve the training efficiency regarding hardware resources, time and power consumption, especially for resource-limited mobile devices where DSP and FPU are not available.

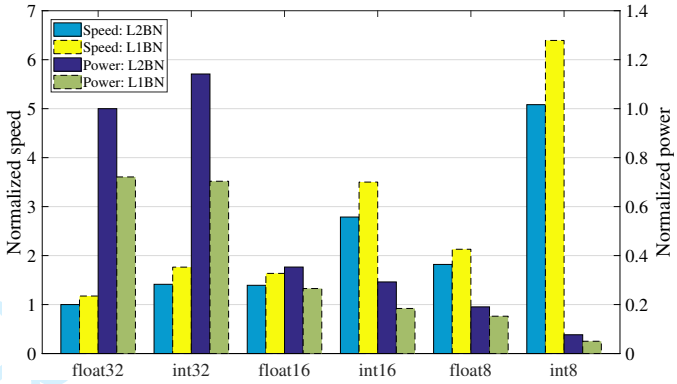

Fig. 4. Estimated time and power consumption for L1-norm and L2-norm on different CNN models.

VI. DISCUSSIONS AND CONCLUSIONS

To reduce the overheads of L2-norm based batch normalization layer, we propose the L1-norm based batch normalization. L1BN is equivalent to L2BN by multiplying a scaling factor $\sqrt{\frac{\pi}{2}}$ if the inputs obey a normal distribution, which is commonly satisfied and observed in conventional networks. Experiments on various CNNs and GANs reveal that L1BN presents comparable classification accuracies, generation qualities and convergence rates. By replacing the costly and nonlinear square and sqrt operations with absolute and sign operations during training, L1BN enables higher computational efficiency. Cost comparisons of basic operations are estimated through ASIC synthesis, L1BN is able to obtain 25% speedup and 37% energy saving. Other hardware resources, e.g., silicon area and cost, can be reduced as well.

Previous deep learning hardware mainly target at accelerating the off-line inference, i.e., the deployment of a well-trained compressed network. Wherein MACs usually occupy much attention, and BN can be regarded as a linear layer once training is done. However, the capability of continual learning in real-life and on-site occasions is essential for future artificial general intelligence (AGI). Thus, online training is very important to both the datacenter equipped with thousands of

CPUs and GPUs, as well as edge devices with resource-limited FPGAs and ASICs, wherein batch normalization should not be bypassed. L1BN with less resource overheads and faster speed can benefit most of the current deep learning models.

Moreover, transferring both training and inference processes to low-precision representation is an effective leverage to alleviate the complexity of hardware design. Regretfully, most of the existing quantization methods remain the BN layer in full-precision (float32) because of the strongly nonlinear square and sqrt operations. By replacing them with absolute and sign operations, L1BN greatly promises a fully quantized neural network with low-precision dataflow for efficient online training, which is crucial to future adaptive terminal devices.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[3] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International Conference on Machine Learning*, 2016, pp. 173–182.

[4] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[6] A. Karpathy, A. Joulin, and L. F. Fei-Fei, "Deep fragment embeddings for bidirectional image sentence mapping," in *Advances in neural information processing systems*, 2014, pp. 1889–1897.

[7] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[10] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[11] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242.

[12] X. Huang, Y. Li, O. Poursaeed, J. Hopcroft, and S. Belongie, "Stacked generative adversarial networks," *arXiv preprint arXiv:1612.04357*, 2016.

[13] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[15] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[17] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. B. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas." in *FPGA*, 2017, pp. 15–24.

[18] L. Jiang, M. Kim, W. Wen, and D. Wang, "Xnor-pop: A processing-in-memory architecture for binary convolutional neural networks in wide-io2 drams," in *Low Power Electronics and Design (ISLPED, 2017 IEEE/ACM International Symposium on*. IEEE, 2017, pp. 1–6.

[19] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 901–909.

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

[21] S. Ioffe, "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models," *arXiv preprint arXiv:1702.03275*, 2017.

[22] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[23] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE, 2017, pp. 5987–5995.

[24] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *arXiv preprint arXiv:1610.02357*, 2016.

[25] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 4107–4115.

[26] Q. He, H. Wen, S. Zhou, Y. Wu, C. Yao, X. Zhou, and Y. Zou, "Effective quantization methods for recurrent neural networks," *arXiv preprint arXiv:1611.10176*, 2016.

[27] L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li, "Gated xnor networks: Deep neural networks with ternary weights and activations under a unified discretization framework," *arXiv preprint arXiv:1705.09283*, 2017.

[28] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=HJGXzmspb

[29] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[30] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, no. 2, 2011, p. 5.

[31] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[33] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," *arXiv preprint arXiv:1506.03365*, 2015.

[34] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[35] "Tensorflow performance guide," https://www.tensorflow.org/performance/performance_guide.

[36] F. Leone, L. Nelson, and R. Nottingham, "The folded normal distribution," *Technometrics*, vol. 3, no. 4, pp. 543–550, 1961.

[37] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[38] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Artificial Intelligence and Statistics*, 2015, pp. 562–570.

[39] "Tensorflow-slim image classification model library," https://github.com/tensorflow/models/tree/master/research/slim.

[40] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *arXiv preprint arXiv:1704.00028*, 2017.

[41] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

Re: Paper ID: TNNLS-2018-P-9126

Title: L1-Norm Batch Normalization for Efficient Training of Deep Neural Networks

Authors: Shuang Wu, Guoqi Li, Lei Deng, Liu Liu, Dong Wu, Yuan Xie, Luping Shi

Dear Prof. Haibo He,

We deeply appreciate your e-mail dated May 30, 2018, providing us the Reviewers' comments and the Editor's decision on our paper listed above.

We thank you for giving us an opportunity to revise our paper. The paper has been carefully revised according to the Reviewers' and the Editor's comments and suggestions. Attached please find the revised paper and a summary of changes and our responses to the Reviewers' comments. For clarity, in this summary, the comments and suggestions from the Editor and Reviewers are presented in italic font.

We deeply thank the Associate Editor and the anonymous Reviewers for their valuable suggestions and comments, which have helped us to improve our paper. We hope that the revised version is now of a satisfactory quality.

Once again, thank you for all the kind efforts in processing our paper. We are looking forward to hearing from you.

Yours sincerely,

Shuang Wu, Guoqi Li, Lei Deng, Liu Liu, Dong Wu, Yuan Xie, Luping Shi

Response to Associate Editor

Re: Paper ID: TNNLS-2018-P-9126

Title: L1-Norm Batch Normalization for Efficient Training of Deep Neural Networks

Authors: Shuang Wu, Guoqi Li, Lei Deng, Liu Liu, Dong Wu, Yuan Xie, Luping Shi

Dear Associate Editor,

Thank you for your kind efforts in handling our manuscript. This paper has been carefully revised according to the Reviewers' suggestions and comments. As we have supplemented an experiment on ASIC synthesis done by Dr. Dong Wu, he is added into the authors list now. A summary of changes in our revised manuscript and point-to-point responses to the Reviewers' comments can be found below.

Comment 1: *The paper does have a good contribution to make, and has potential, provided that the authors address the issues raised by the reviewers. While the idea presented is simple, and mostly motivated to reduce computational complexity, the significance of replacing L2BN with L1BN needs to be emphasized, specifically with reference to the total computational cost in relation to the fact that similar performance is obtained, and in some case a small improvement in performance.*

Response: Thanks for your encouragement. The significances of replacing L2BN with L1BN are described as follows:

- Batch normalization (BN) is so helpful in training DNNs that it has almost become a standard component. Therefore, the understanding and optimization of BN can directly benefit the entire deep learning community. Our work analyzes the statistics of BN and their effects in the training of deep neural networks, the idea of L1-norm is both motivated theoretically and demonstrated empirically.
- Our L1BN can maintain the same accuracy and convergence rate as L2BN, but with higher computational efficiency. In our revised manuscript, we have switched from FPGA to real ASIC synthesis to estimate the overall computational efficiency, which is more accurate and supports multiple datatypes. Experiments show that L1BN can achieve 25% speedup and 37% energy saving. Other hardware resources, such as silicon area and cost, can be reduced as well. More detailed answers can be found in the response to Comment 1 for Reviewer 3.
- BN challenges the network quantization, which draws much attention to the accelerator community.

Our L1BN simplifies the non-linear operations in L2BN, thus promises a fully quantized neural network with low-precision dataflow for efficient discrete training. Recently, Range Batch Normalization [1] was proposed based on our work (arxiv preprint), which directly targeted this issue.

- Our paper (arxiv preprint) has been cited for 3 times [1][2][3] in the past three months, which indicates that our work is valuable to the community.

Comment 2: *A number of concepts are used in the paper, but needs to be explained in more detail, for example, fused batch norm, the argument that L1BN brings in additional randomness, why learn the scale factor, saturated activations, amongst others. Careful consideration should also be given to the reference to the concepts, for example saturated activations. Then, a number of concepts are used, without reference to literature where these concepts have been introduced and discussed, e.g. dropout.*

Response: In our revised manuscript, we have supplemented more references and explanations, e.g. dropout [4], fused batch normalization [5], mode collapse [6], saturated activation, ReLU [7], etc.

We give a detailed explanation and a reference for **"fused batch normalization"** [5]. It is just a faster implementation that combines multiple required operations into a single kernel on GPU and accelerates the normalization process.

We have modified and explained the argument "**additional randomness**". In the original BN paper [8], it is claimed and demonstrated that:

"This led to about 1% improvement in the validation accuracy, which is consistent with the view of Batch Normalization as a regularizer: the randomization inherent in our method should be most beneficial when it affects an example differently each time it is seen."

In our L1BN, since the L1-norm σ_{L_1} is smaller than L2-norm σ_{L_2}, the output of normalization will be holistically $1.25\times$ larger, which enhances the randomization effect and may further regularize large model.

The use of trainable linear layer (γ, β) after normalization is a common practice suggested in the original L2BN paper [8]. It introduces very few parameters and computation but can improve the final performance. To be consistent with the results using L2BN in other papers, we choose to **learn the rescale factors**. This is the same as L2BN.

As for the concept "**saturated activation**", we have made it more concise in our revised manuscript. Please refer to the response to Comment 4 for Reviewer 2.

Comment 3: *Then, also note the concerns about the empirical work, in that it is not always clear which data sets have been used for training. Appropriate statistical tests should be used to indicate if any differences in performance are statistically significant or not.*

Response: We have added the dataset information in Figure 1 and Figure 3: ResNet-110 is trained on CIFAR-10.

The results on classification tasks are the average of best accuracies in multiple (5-10) repeated experiments. Among all these experiments, L1BN is equal to or slightly surpasses the original L2BN.

In Figure 3, we offer a layerwise and channelwise perspective to demonstrate the equivalence between L1-norm and L2-norm scaling. The colormap and probability histograms of ratios $\sigma_{L_2}/\sigma_{L_1}$ from intermediate layers (layer 38 to layer 73, totally 1152 channels) are averaged across 100 mini-batches. The results are consistent with Theorem 1 and Remark 1.

In Table IV and Figure 4, we have supplemented appropriate statistical tests in real ASIC synthesis with multiple datatypes: float32, int32, float16, int16, float8 and int8. As with the response in Comment 1, L1BN can averagely achieve 25% speedup and 37% power saving considering the overall time and power consumption in practice.

Comment 4: *Lastly, the linguistic quality of the paper is not acceptable, and the authors are advised to make use of a professional English language editor.*

Response: Thanks for your suggestion. In our revised manuscript, we have carefully checked through the whole paper to fix all the typos and grammatical errors we have identified, and improved the writing of the paper as well.

Related references:

[1] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," *arXiv preprint arXiv:1805.11046*, 2018.

[2] J. Bjorck, C. Gomes, and B. Selman, "Understanding batch normalization," *arXiv preprint arXiv:1806.02375*, 2018.

[3] E. Hoffer, R. Banner, I. Golan, and D. Soudry, "Norm matters: efficient and accurate normalization

schemes in deep networks," *arXiv preprint arXiv:1803.01814*, 2018.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[5] "Tensorflow performance guide," `https://www.tensorflow.org/performance/performance_guide`.

[6] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242.

[7] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

Responses to Reviewer 1

Dear reviewer:

Thanks for your insightful and detailed comments. Please find our responses to individual questions below:

Comment 1: *Near Equation 4, the epsilon is added for numerical stability not "avoid numerical error".*

Response: Thanks for catching that. We have fixed our expression, and shown an example value of 1e-5 used in batch normalization.

Comment 2: *The first paragraph of page 3, what is "fused"batchnorm? Would you mind providing a reference?*

Response: In our revised manuscript, we give a detailed explanation and a reference: fused-batchnorm [1] is just a faster implementation. It combines multiple required operations into a single kernel on GPU and accelerates the normalization process.

Comment 3: *Figure 3, the sigma and mus are **fixed** during inference so I don't understand why they are being calculated for Figure 3. Are you trying to show that the input is Gaussian distributed? In addition, if the learning involves gamma and beta (the additional scalar), those figures are quite irrelevant.*

Response: In Theorem 1, we have proved that L1-norm statistics σ_{L1} are linearly correlated to L2-norm statistics σ_{L2} if the input is Gaussian distributed. In Figure 3, we are trying to show that this linear correlation is still approximately satisfied in real network training. So we directly export the L1-norm and L2-norm statistics, and make a comparision.

Note that the following explanations are now based on the revised manuscript.

- **Gamma** (γ) and **beta** (β) are linear layer parameters, which are learned during training.
- **Sigma** (σ) and **mu** (μ) are statistics calculated according to the inputs.

Batch normalization is processed as follows:

(a) Calculate σ and μ according to the current inputs (Equation 5 and 6);

(b) Normalize inputs using σ and μ (Equation 4);

(c) Optional: a trainable linear layer with scale γ and bias β (Equation 7).

We want to demonstrate that process (a) and (b) can be replaced by L1-norm formulation (Equation 11 and 10). Therefore, we need to fix all the network parameters and trainable linear layer parameters.

In Figure 3, we calculate and plot the statistics (σ), to show that the L1-norm statistics (σ, μ) are linearly correlated to L2-norm statistics. That is why we can use L1-norm to replace L2-norm and achieve higher efficiency.

As for the trainable linear layer, it introduces very few computation and parameters (γ, β) but can improve the final performance. Therefore, we choose to apply process (c) in each batch normalization, and the parameters are learned during backpropagation training. This is the same as L2BN.

Comment 4: *Which dataset is that ResNet-110 trained on?*

Response: ResNet-110 is trained on CIFAR-10. We have added the dataset information in Figure 1 and Figure 3.

Comment 5: *L1-BatchNorm brings additional "randomness"–I can't really make sense of this argument.*

Response: We have modified and explained this argument in our revised manuscript, now the "randomness" is **enhanced**, not **additional**. In the original BN paper [2], it is claimed and demonstrated that:

"This led to about 1% improvement in the validation accuracy, which is consistent with the view of Batch Normalization as a regularizer: the randomization inherent in our method should be most beneficial when it affects an example differently each time it is seen."

In our L1BN, since the L1-norm σ_{L_1} is smaller than L2-norm σ_{L_2}, the output of normalization will be holistically $1.25\times$ larger, which enhances the randomization effect and may further regularize large model.

Related references:

[1] "Tensorflow performance guide," https://www.tensorflow.org/performance/

`performance_guide`.

[2] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

Responses to Reviewer 2

Dear reviewer:

We appreciate your detailed feedback and probing questions. Please find our responses to individual questions below:

Comment 1: *While the use of L1 normalization in place of L2 in batch normalization is interesting, the topic itself is shallow. Still, your treatment is extensive ...*

Response: Thanks for your encouragement. Although the idea of our paper is simple, the benefits are general and useful:

- Batch normalization (BN) is so helpful in training DNNs that it has almost become a standard component. Therefore, the understanding and optimization of BN can directly benefit the entire deep learning community. Our work analyzes the statistics of BN and their effects in the training of deep neural networks, the idea of L1-norm is both motivated theoretically and demonstrated empirically.
- Our L1BN can maintain the same accuracy and convergence rate as L2BN, but with higher computational efficiency. In our revised manuscript, we have switched from FPGA to real ASIC synthesis to estimate the overall computational efficiency, which is more accurate and supports multiple datatypes. Experiments show that L1BN can achieve 25% speedup and 37% energy saving. Other hardware resources, such as silicon area and cost, can be reduced as well. More detailed answers can be found in the response to Comment 1 for Reviewer 3.
- BN challenges the network quantization, which draws much attention to the accelerator community. Our L1BN simplifies the non-linear operations in L2BN, thus promises a fully quantized neural network with low-precision dataflow for efficient discrete training. Recently, Range Batch Normalization [1] was proposed based on our work (arxiv preprint), which directly targeted this issue.
- Our paper (arxiv preprint) has been cited for 3 times [1][2][3] in the past three months, which indicates that our work is valuable to the community.

Comment 2: *It seems that the central contribution of your work is the experimental work, particularly with smaller networks and the efficiency gain.*

Response: In Theorem 1, we have proved that L1-norm statistics σ_{L1} are linearly correlated to L2-norm statistics σ_{L2} if the input is Gaussian distributed. Then we use many experiments to verify the effectiveness of L1BN in real network training. Our experiments are conducted from three aspects:

- CNN and GAN experiments on GPUs show that L1BN is equivalent to L2BN in convergence rates and final accuracies.
- In Figure 3, we offer a layerwise and channelwise perspective to demonstrate the equivalence between L1-norm and L2-norm scaling. The colormap and probability histograms of ratios $\sigma_{L_2}/\sigma_{L_1}$ from intermediate layers (layer 38 to layer 73, totally 1152 channels) are averaged across 100 mini-batches. The results are consistent with Theorem 1 and Remark 1.
- In our revised manuscript, we have switched from FPGA to real ASIC synthesis to estimate the overall computational efficiency, which is more accurate and supports multiple datatypes. Experiments show that L1BN can achieve 25% speedup and 37% energy saving. Other hardware resources, such as silicon area and cost, can be reduced as well.

Comment 3: *Firstly, there are missing citations, for example, you mention Dropout, but it is not cited.*

Response: Thank you for your careful findings. In our revised manuscript, we have supplemented more references and explanations, e.g. dropout [4], fused batch normalization [5], mode collapse [6], activation, ReLU [7], etc.

Comment 4: *Other issues are regarding your explanation of certain phenomena. Examples of this include the use of "saturated activations" on page 2, it is not clear how that is relevant.*

Response: We have removed some irrelevant expressions, and made it more convincing and fluent.

As for the "saturated activation" on page 2, it is relevant because values stuck in the saturation regime [8] will have vanishing gradients, which may slow down the convergence. Via using batch normalization, the distributions of pre-activations are explicitly shifted to an approximatively linear interval, e.g., [-1, +1], where gradients can propagate with proper magnitudes.

Since this is not the main content, we have made it more concise in our revised manuscript.

Comment 5: *In addition, the explanation of the rationale for your experiments is not well detailed. For example, I understand the set-up on page 4 but I cannot follow the reason why you choose to learn the rescale factor.*

Response: We have added explanation in this version. It should be noted that:

- **Gamma** γ and **beta** β are linear layer parameters, which are learned during training.
- **Sigma** σ and **mu** μ are statistically calculated according to current inputs.

Our work mainly focuses on σ and μ, and L1BN is proven to be an efficient alternative to L2BN. The use of trainable linear layer (γ, β) after normalization is a common practice suggested in the original L2BN paper [8]. It introduces very few parameters and computation but can improve the final performance. To be consistent with the results using L2BN in other papers, we also choose to learn the rescale factors.

Comment 6: *Lastly, some grammatical issues make the reading of this paper difficult.*

Response: Thanks for your reminder. In the revised manuscript, we have gone through the paper very carefully, and tried our best to polish it.

Related references:

[1] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," *arXiv preprint arXiv:1805.11046*, 2018.

[2] J. Bjorck, C. Gomes, and B. Selman, "Understanding batch normalization," *arXiv preprint arXiv:1806.02375*, 2018.

[3] E. Hoffer, R. Banner, I. Golan, and D. Soudry, "Norm matters: efficient and accurate normalization schemes in deep networks," *arXiv preprint arXiv:1803.01814*, 2018.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[5] "Tensorflow performance guide," `https://www.tensorflow.org/performance/performance_guide`.

[6] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242.

[7] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

Responses to Reviewer 3

Dear reviewer:

Thanks for your thoughtful comments and probing questions. Please find our responses to individual questions below:

Comment 1: *My main concern about this paper is its significance. In particular, how much more efficient L1BN is compared to L2BN in real experiments? Note that L2BN only needs 1 root operation each pass (you can store the value of $\sqrt{\sigma^2+\epsilon}$ for later use). Given that other operations and other factors in the experiments, how significant this replacement of L2BN by L1BN is in the overall computational cost? Section IV-D only count the number of root. Could the authors show the computational curve (in time/W) of couple real experiments?*

Response: Thanks for your valuable advice, we find that the power and speed estimation on FPGAs are affected by many factors, e.g., DSP, register, RAM blocks, logic resources, and even time constrains, which make it difficult to explain very clearly. Therefore, in the revised manuscript, we have switched to real ASIC synthesis to estimate the overall computational efficiency, which is more accurate and supports more datatypes. Table IV and Figure 4 are updated according to the new results.

We analyze the total number of all the basic arithmetic operations in the entire normalization process, where the number of channels (c) and the number of data points (m) are explicitly presented in Table IV. As for the number of root operations, each L2BN layer has c root operations. That is to say, each channel has 1 root operation (now denoted as sqrt operation in this version), not each layer.

Example: batchsize $m = 128$

- A fully-connected layer with 256 outputs $\boldsymbol{x} = [x^{(1)}, x^{(2)}, \ldots, x^{(256)}]$, the number of root operations is 1×256. The total number of square operations is 128×256.
- A convolution layer with 256 output feature maps $\boldsymbol{I} = [I^{(1)}, I^{(2)}, \ldots, I^{(256)}]$, each feature map has 7×7 pixels, the number of root operations is 1×256. The total number of square operations is 128×7×7×256.

Compared to multiplication and square operations with the total number of mc, the root operations are less (only c, but not 1). However, it introduces strong nonlinearity and makes it difficult to employ low bit-width quantization.

Our L1BN simplifies the non-linear operations in L2BN, thus promises a fully quantized neural network with low-precision dataflow for efficient discrete training. Recently, Range Batch Normalization [1] was proposed based on our work (arxiv preprint), which directly targeted this issue.

Table IV: Total numbers and computational overheads of basic arithmetic operations in ASIC synthesis. Here m denotes the total number of intra-channel data points within one batch for simplicity, and c denotes the number of channels.

type	overhead	SIGN	ABS	ADD	SUB	MUL	DIV	SQUARE	SQRT
Quantity	L2BN	0	0	5mc+c	2mc	4mc+2c	2c	mc	c
	L1BN	mc	mc	5mc+c	2mc	3mc+2c	2c	0	0
float32	time(ns)	0.00	0.71	43.00	43.00	35.83	99.49	27.23	99.50
	power(μW)	0.00	0.01	29.50	29.60	97.20	254.00	39.10	59.00
	area(μm^2)	0.00	3.39	8095	8097	19625	36387	11539	11280
int8	time(ns)	0.00	3.47	5.25	5.45	11.46	24.90	8.54	7.60
	power(μW)	0.00	0.59	0.77	0.87	9.55	5.90	3.64	0.82
	area(μm^2)	0.00	256	272	306	2319	1762	1241	277

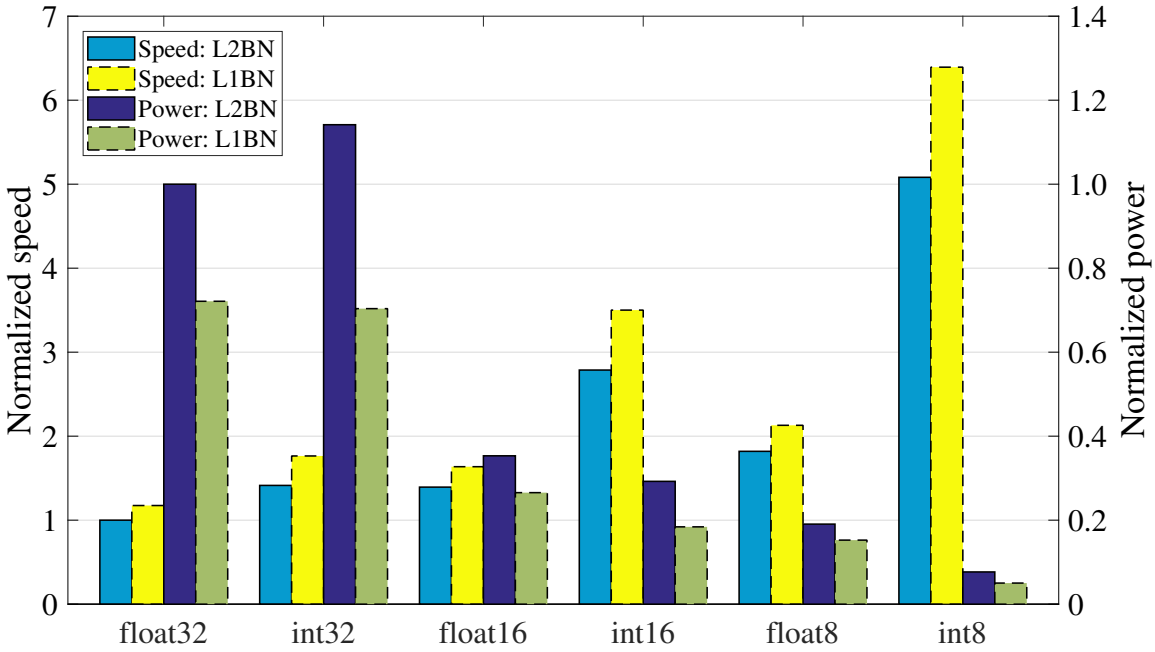

Fig 4. Estimated time and power consumption for L1-norm and L2-norm on different CNN models.

Comment 2: *I can understand that root is costly. Why would square be a costly operator? Should it cost similar to a multiplication?*

Response: Yes, it costs similar to a multiplication operation. But its corresponding alternative is an abs operation in L1BN, not multiplication. The square is a costly operator compared to the abs. We have made this clearer in our revised manuscript.

Comment 3: *In Figure 4, how many root operations are counted in each L2BN layer for one batch?*

Response: As mentioned in Comment 1, each L2BN layer has c root operations, where c represents for the number of channels.

Comment 4: *On page 5 line 47-51 and Section IV-C, why would L1BN bring more randomness and further regularize the model? It may need more elaboration on this point. I cannot get it from Figure 3.*

Response: We have modified and explained this argument in our revised manuscript, now the "randomness" is **enhanced**, not **additional**. In the original BN paper [2], it is claimed and demonstrated that:

"This led to about 1% improvement in the validation accuracy, which is consistent with the view of Batch Normalization as a regularizer: the randomization inherent in our method should be most beneficial when it affects an example differently each time it is seen."

In our L1BN, since the L1-norm σ_{L_1} is smaller than L2-norm σ_{L_2}, the output of normalization will be holistically $1.25\times$ larger, which enhances the randomization effect and may further regularize large model.

Related references:

[1] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," *arXiv preprint arXiv:1805.11046*, 2018.

[2] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.