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

Shuang Wu, Guoqi Li, *Member, IEEE*, Lei Deng, *Member, IEEE*, Liu Liu, Dong Wu, Yuan Xie, *Fellow, IEEE*, and Luping Shi, *Member, IEEE*

***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 draw much attention to the deep learning hardware community. In this paper, we propose an $L1$-norm BN (L1BN) with only linear operations in both forward and backward propagations during training. L1BN is approximately equivalent to the conventional $L2$-norm BN (L2BN) by multiplying a scaling factor that equals $(\pi/2)^{1/2}$. Experiments on various convolutional neural networks and generative adversarial networks reveal that L1BN can maintain the same performance and convergence rate as L2BN but with higher computational efficiency. In real application-specified integrated circuit 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 least, L1BN promises a fully quantized training of DNNs, which empowers future artificial intelligence applications on mobile devices with transfer and continual learning capability.**

***Index Terms*— Batch normalization (BN), deep neural network (DNN), discrete online learning, $L1$-norm, mobile intelligence.**

I. Introduction

TODAY, deep neural networks (DNNs) [1] are rapidly permeating into various artificial intelligence applications, for instance, computer vision [2], speech recognition [3], machine translation [4], Go game [5], and multimodel tasks across them [6]. However, training DNNs is complicated and needs elaborate tuning of hyperparameters, especially on large data sets with diverse samples and thousands of categories. Therefore, the distribution of minibatch 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. In addition, the inherent randomization incurred by the minibatch statistics serves as a regularizer and approximates inference in Bayesian models [9]–[11], making BN a better alternative to dropout [12]. Most generative adversarial networks (GANs) also rely on BN in both the generator and the discriminator [13]–[15]. With BN, a deep generator can start with a normal training process, as well as avoid mode collapse [14] 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) [16], not only in most deep learning models [17]–[19] but also in the neural network accelerator community [20], [21].

Inspired by BN, weight normalization [22] reparameterizes the incoming weights by their $L2$-norm. Layer normalization [23] replaces the statistics of a training batch with a single training case, which does not reparameterize the network. Both methods eliminate the dependencies among samples in a minibatch and overcome the difficulties of applying normalization in recurrent models [24]. In batch renormalization [25], an affine transformation is proposed to ensure that the training and inference models generate the same outputs that depend on individual example rather than entire minibatch.

However, BN usually causes considerable overheads in both the forward and backward propagations. Recently, in the field of convolutional neural networks (CNNs), there is a trend toward replacing the standard convolution with bottleneck pointwise convolution [26], group convolution [27], or depthwise convolution [28]. Although the number of

Manuscript received March 2, 2018; revised June 27, 2018 and October 4, 2018; accepted October 4, 2018. This work was supported in part by the National Natural Science Foundation of China under Grant 61327902, Grant 61603209, and Grant 61876215, in part by the Suzhou-Tsinghua Innovation Leading Program under Grant 2016SZ0102, in part by the Brain-Science Special Program of Beijing under Grant Z181100001518006, and in part by the National Science Foundation under Grant 1730309 and Grant 1725447. *(Shuang Wu and Guoqi Li contribute equally to this work.) (Corresponding author: Luping Shi.)*

S. Wu, G. Li, and L. Shi are with the Center for Brain-Inspired Computing Research, Tsinghua University, Beijing 100084, China, also with the Beijing Innovation Center for Future Chip, Tsinghua University, Beijing 100084, China, and also with the Optical Memory National Engineering Research Center, Department of Precision Instrument, Tsinghua University, Beijing 100084, China (e-mail: lpshi@mail.tsinghua.edu.cn).

L. Deng, L. Liu, and Y. Xie are with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA 93106 USA.

D. Wu is with the Institute of Microelectronics, Tsinghua University, Beijing 100084, China.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2018.2876179

parameters and multiply–accumulate (MAC) operations has 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 [19].

On the one hand, the additional calculations in BN are costly, especially for resource-limited application-specified integrated circuit (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 hand, the square and sqrt operations introduce 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 [29]–[31], the BN layers are simply avoided [32] or maintained in float32 precision.

In this paper, we introduce an $L1$-norm BN (L1BN), where the $L2$-norm variance for minibatch samples is substituted by an $L1$-norm "variance." Then, the costly and nonlinear square and sqrt operations can be replaced by hardware-friendly sign and absolute operations. L1BN is approximately equivalent to the $L2$-norm BN (L2BN) by multiplying a scaling factor that equals $(\pi/2)$. To evaluate the proposed method, various experiments have been conducted with CNNs on data sets including Fashion-MNIST [33], Street View House Numbers Dataset (SVHN) [34], CIFAR [35], and ImageNet [36], as well as GANs on CIFAR and LSUN-Bedroom [37]. Results indicate that L1BN is able to achieve comparable performance 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 L1BN 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. In addition, $L1$-norm is an orthogonal method and can be fruitfully combined with many other advances of the network normalization [22]–[25].

II. Problem Formulation

Stochastic gradient descent (SGD), known as the incremental gradient descent, is a stochastic and iterative approximation of gradient descent optimization. SGD minimizes an objective function with differentiable parameters $\mathbb{C}$

$$\mathbb{C} = \arg\min_{\mathbb{C}} \frac{1}{N} \sum_{i=1}^{N} 4(x_i, \mathbb{C}) \tag{1}$$

where x_i for $i = 1, 2, \ldots, N$ is the training set containing totally N samples and $\sum_{i=1}^{N} 4(x_i, \mathbb{C})$ is the empirical risk summarized by the risk at each sample $4(x_i, \mathbb{C})$. Considering SGD with a minibatch of m samples, then the gradient of the loss function 4 can be simply approximated by processing the gradient of m samples and update with the average value

$$\mathbb{C} \leftarrow \mathbb{C} - \eta \cdot \frac{1}{m} \sum_{i=1}^{m} \frac{\partial 4(x_i, \mathbb{C})}{\partial \mathbb{C}} \tag{2}$$

where η is the learning rate.

While SGD with minibatch 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 for 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 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; 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., sigmoid(x) and $\tanh(x)$, which affect the networks' robustness and challenges its extensive applications.

Ioffe and Szegedy [7] 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 minibatch, which is termed as "BN." BN ensures that the distribution of preactivations (values fed into activation) remains stable as the values propagate across layers in deep network. Then, the SGD optimizer will be less likely to get stuck in the saturated regime or explode to nonconvergence, allowing 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 minibatch $\mathcal{B}$ containing m samples, we use $\mu_{\mathcal{B}}$ and $\sigma^2_{\mathcal{B}} + z$ to estimate $\mathrm{E}[x^{(k)}]$ and $\mathrm{Var}[x^{(k)}]$, respectively, which gives that

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

where z is a sufficiently small positive parameter, e.g., $1\mathrm{e}-5$, for keeping numerical stability. The minibatch mean $\mu_{\mathcal{B}}$ and

TABLE I
TRAINING TIME (MILLISECOND) PER IMAGE WITH OR WITHOUT BN, 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

variance σ_B^2 are given as

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{5}$$

and

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2. \tag{6}$$

It is easy to see that σ_B^2 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 scales and shifts the normalized values is introduced

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

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

During training, we need to backpropagate 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:

$$\begin{aligned}
\frac{\partial \ell}{\partial \hat{x}_i} &= \frac{\partial \ell}{\partial y_i}\cdot\gamma \\
\frac{\partial \ell}{\partial \sigma_B^2} &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial \hat{x}_i}\cdot(x_i-\mu_B)\cdot\frac{-1}{2}\cdot(\sigma_B^2+z)^{-3/2} \\
\frac{\partial \ell}{\partial \mu_B} &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial \hat{x}_i}\cdot\frac{-1}{\sqrt{\sigma_B^2+z}} \\
\frac{\partial \ell}{\partial x_i} &= \frac{\partial \ell}{\partial \hat{x}_i}\cdot\frac{1}{\sqrt{\sigma_B^2+z}} + \frac{\partial \ell}{\partial \sigma_B^2}\cdot\frac{2(x_i-\mu_B)}{m} + \frac{\partial \ell}{\partial \mu_B}\cdot\frac{1}{m} \\
\frac{\partial \ell}{\partial \gamma} &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial y_i}\cdot\hat{x}_i \\
\frac{\partial \ell}{\partial \beta} &= \sum_{i=1}^{m}\frac{\partial \ell}{\partial y_i}.
\end{aligned} \tag{8}$$

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 CNNs. All models are built on Tensorflow [38] and trained with one or two Titan-Xp GPUs. Section V-A will detail the we apply the widely used built-in version: fused BN [39] that combines multiple required operations into a single kernel on GPU and can accelerate the normalization process. Even so, BN brings in about 30%–50% extra training time. Note that $L2$-norm is applied to estimate σ_B we term this method as the L2BN. To address this issue, we propose an L1BN in this paper.

IV. PROPOSED L1BN

Our idea is simple but effective, which applies the $L1$-norm to estimate σ_B. The L1BN is formulated as

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

and

$$\hat{x}_i = \frac{x_i - \mu_B}{\sigma_B + z} \tag{10}$$

where γ, β, μ_B, and z are identical to that in L2BN, and σ_B is a term calculated by the $L1$-norm of minibatch statistics

$$\sigma_B = \frac{1}{m}\sum_{i=1}^{m} |x_i - \mu_B|. \tag{11}$$

The motivation of using the $L1$-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 [40]. Denote μ_Y as the mean of Y, we have

$$\mu_Y = \sqrt{\frac{2}{\pi}}\sigma e^{\frac{-\mu^2}{2\sigma^2}} + \mu\left(1-2\alpha\left(-\frac{\mu}{\sigma}\right)\right) \tag{13}$$

based on the statistical property of folded normal distribution. α is the normal cumulative distribution function. 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 (11) as σ_{L_1}, we have

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

implementation and training hyperparameters. For BN,

Let γ_{L2} and γ_{L1} be the scale parameters in (7) and (9) for L2BN and L1BN, respectively. To keep the outputs of the two methods identical, ideally we have

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

Q

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

$$\frac{\partial \ell}{\partial \hat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_B} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}_i} \cdot (x_i - \mu_B) \cdot \frac{-1}{(\sigma_B + z)^2}$$

$$\frac{\partial \ell}{\partial \mu_B} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{-1}{\sigma_B + z}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \sigma_B} \cdot \frac{1}{m} \operatorname{sgn}(x_i - \mu_B) + \left\{ \frac{\partial \ell}{\partial \mu_B} - \frac{1}{m} \sum_{j=1}^{m} \frac{\partial \ell}{\partial \hat{x}_j} \cdot \operatorname{sgn}(x_j - \mu_B) \right\} \cdot \frac{1}{m} + \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{1}{\sigma_B + z} + \frac{\partial \ell}{\partial \mu_B} \cdot \frac{1}{m}. \tag{17}$$

$(\partial \ell / \partial \gamma)$ and $(\partial \ell / \partial \beta)$ have exactly the same form as in (8). It is obvious that

$$\operatorname{sgn}(\hat{x}_i) = \operatorname{sgn}(x_i - \mu_B). \tag{18}$$

Let

$$\mu = \frac{\partial \ell}{\partial \hat{x}_i} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}_i} \pm \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}_i} \cdot \hat{x}_i \quad \mu \frac{\partial \ell}{\partial \hat{x}_i} \cdot \hat{x}_i. \tag{19}$$

Then, by substituting (10) and (11) into (17), we can obtain that

$$\frac{\partial \ell}{\partial x_i} = \frac{1}{\sigma_B + z} \left[\frac{\partial \ell}{\partial \hat{x}_i} - \mu \frac{\partial \ell}{\partial \hat{x}_i} - \mu \frac{\partial \ell}{\partial \hat{x}_i} \cdot \hat{x}_i \cdot [\operatorname{sgn}(\hat{x}_i) - \mu(\operatorname{sgn}(\hat{x}_i))] \right]. \tag{20}$$

Note that square and sqrt operations can be avoided in both the forward and backward propagations when L1BN is involved. As shown in Section V, L1BN is approximately equivalent to L2BN in convergence rate and final performance 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 multilayer perceptrons (MLPs) and CNNs. The only difference is that the $L1$-norm is applied to calculate σ. As shown in (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 BN, we multiply σ_{L_1} by $\sqrt{(\pi/2)}$ in (11) to approximate σ_{L_2}.
2) When the scale factor γ is applied in BN, we just use σ_{L_1} and let the trainable parameter γ_{L_1} "learn" to compensate the difference automatically through back-

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 $B = \{x_1, ..., x_m\}$, one pre-activation sample for inference $I = \{x_{\text{inf}}\}$

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

1. Training with mini-batch B:

1.1 Forward:

1: $\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i$ //mini-batch mean

2: $\sigma_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} |x_i - \mu_B|$ //mini-batch L1 variance

3: $\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sigma_B + z}$ //normalize

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

1.2 Backward:

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

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

1.3 Update statistics:

7: $\mu \leftarrow \alpha\mu + (1 - \alpha)\mu_B$ //moving-average

8: $\sigma \leftarrow \alpha\sigma + (1 - \alpha)\sigma_B$ //moving-average

2. Inference with sample I:

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

in MLPs, each channel is normalized based on m samples in a training minibatch. 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 B be the set of all values across both the propagation training.

Other normalization processes of L1BN are just exactly the same as that in L2BN. For the fully connected layer

minibatch and spatial locations, so for a minibatch containing m samples with height h and width w, the effective number of data points for each channel is 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.

V. Experiments

A. L1BN on Classification Tasks

In order to verify the equivalence between L1BN and its original $L2$-norm version, we test both methods in various image classification tasks. In these tasks, we parameterize model complexity and task complexity, and then apply 2-D demonstrations with different CNN architectures on multiple data sets. 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, the SGD optimizer with momentum 0.9 is the default configuration. As suggested in [7], the trainable linear layer is applied in each BN 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 $(\pi/2)$ in the calculation of σ_{L_1}.

$|B|=$

1) Simple Tasks With Shallow Models: Fashion-MNIST [33] 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 [41] network and train for totally 90 epochs with minibatch 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 [34] data set, we use a Visual Geometry Group-like network with totally seven layers [32]. Input images are scaled and biased to the range of $[1, 4]$ and 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 [17] and densely concatenations [18] 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 [35] data sets. A DensetNet-100 [18] network is trained on CIFAR-100, as for the bottleneck architecture [26], a ResNet-110 [17], and a Wide-DenseNet ($L = 40$ and $K = 48$) [18] are trained on CIFAR-10. We follow the data augmentation in [42] 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 [17].

3) Complicated Tasks With Deep Wide Models: For ImageNet data set with 1000 categories [36], we adopt AlexNet [2] model but remove dropout and replace the local response normalization layers with L1BN or L2BN. Images are first 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 minibatch size of 256 and totally 80 epochs. Weight decay is set to $5e\text{-}4$, 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 [27] and depthwise convolution [28] draw extensive attention with equal performance but much fewer parameters and MACs. Therefore, we further reproduce MobileNet [19] and evaluate L1BN on ImageNet data set. At this time, weight decay decreases to $4e\text{-}5$, learning rate η is set to 0.1 initially, and linearly annealed to $1e\text{-}3$ after 60 epochs. We apply the Inception data argumentation defined in TensorFlow-Slim image classification model library [43]. The training is performed on two Titan-Xp GPUs and the population statistics $\{\mu, \sigma\}$ for BN are updated according to the calculations from single GPU, so the actual batchsize for BN is 128.

The main results are summarized in Table II, and we run each model for five times and show mean$\pm$std of the error rates. In addition, the training curves of two methods using the ResNet-110 model on CIFAR-10 data set are shown in Fig. 1. We have two major observations.

From the perspective of the final results, L1BN is approximately equivalent to the original L2BN with only marginal performance distinctions, which might be caused by the

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

Dataset	Model	L2BN	L1BN
Fashion	LeNet-5	7.66±0.24	7.62±0.21
SVHN	VGG-7	1.93±0.03	1.92±0.03
CIFAR-10	ResNet-110	6.24±0.09	6.12±0.15
CIFAR-10	Wide-DenseNet	4.09±0.12	4.13±0.10
CIFAR-100	DenseNet-100	22.5±0.39	22.4±0.36
ImageNet	AlexNet	42.5±0.62	42.1±0.52
ImageNet	MobileNet	29.5±0.46	29.7±0.59

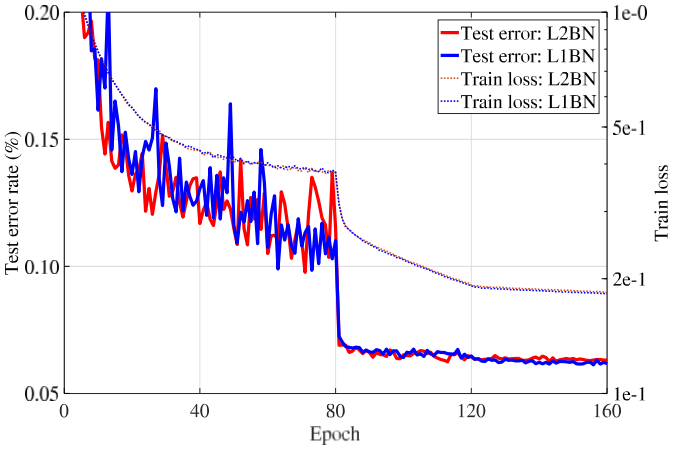


Fig. 1. Training curves of ResNet-110 network on CIFAR-10 data set using L2BN or L1BN. Training losses contain the sum of weight decay losses.

enlarged normalized statistics $\hat{x}_i$: according to (15), σ_{L_1} is smaller than σ_{L_2}, which results in a 1.25× amplification of $\hat{x}_i$. As mentioned earlier, the inherent randomization incurred by the minibatch 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, the $L1$-norm intuitively enhances this randomization and may further regularize large model. 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, the enhanced regularization may hurt the accuracy by a small margin.

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 $\sqrt{(2/\pi)} \approx 0.8$ ($L1$-norm), or following 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

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
UNSUPERVISED ISS ON CIFAR-10 (LARGER VALUES REPRESENT FOR BETTER GENERATION QUALITY)

Method	BN	AVG	BEST
DCGAN (in [15])	L2	6.16 ± 0.07	
Improved GAN [14]	L2	6.86 ± 0.06	
WGAN-GP [44]	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

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. First, we generate 32 × 32 CIFAR-10 images using deep convolutional generative adversarial network (DCGAN) [13] and wasserstein generative adversarial network with gradient penalty (WGAN-GP) [44]. In the reproduction of DCGAN, we use the nonsaturating loss function proposed in [45]. As for WGAN-GP, the network remains the same except that no BN is applied to discriminators as suggested in [44]. In addition, other training techniques, e.g., Wasserstein distance and gradient penalty, are adopted to improve the training process. Generated images are evaluated by the widely used metric inception score (IS) introduced in [14]. Since IS results fluctuate during training, we evaluate sampled images after every 10 000 generator iterations and report both the average score of the last 20 evaluations (AVG) and the overall best score (BEST). 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.

Second, we perform experiments on LSUN-Bedroom [37] high-resolution image generation task. The original DCGAN only deals with image size 64×64, so an additional upsample deconvolution 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 the LSUN-Bedroom data set only has single class (bedroom) of images, the generated images cannot be evaluated by IS that requires multiple categories of samples. Fig. 2 just intuitively shows generated images after 300 000 generator iterations. Still, both methods generate comparable samples containing detailed textures, and we observe no significant difference in artistic style.

C. Layerwise and Channelwise Comparison

We further offer a layerwise and channelwise perspective to demonstrate the equivalence between the $L1$-norm and $L2$-norm. 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. However, 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.

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

TABLE IV

TOTAL NUMBERS AND COMPUTATIONAL OVERHEADS OF BASIC ARITHMETIC OPERATIONS IN ASIC SYNTHESIS. HERE, m DENOTES THE TOTAL NUMBER OF INTRACHANNEL 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

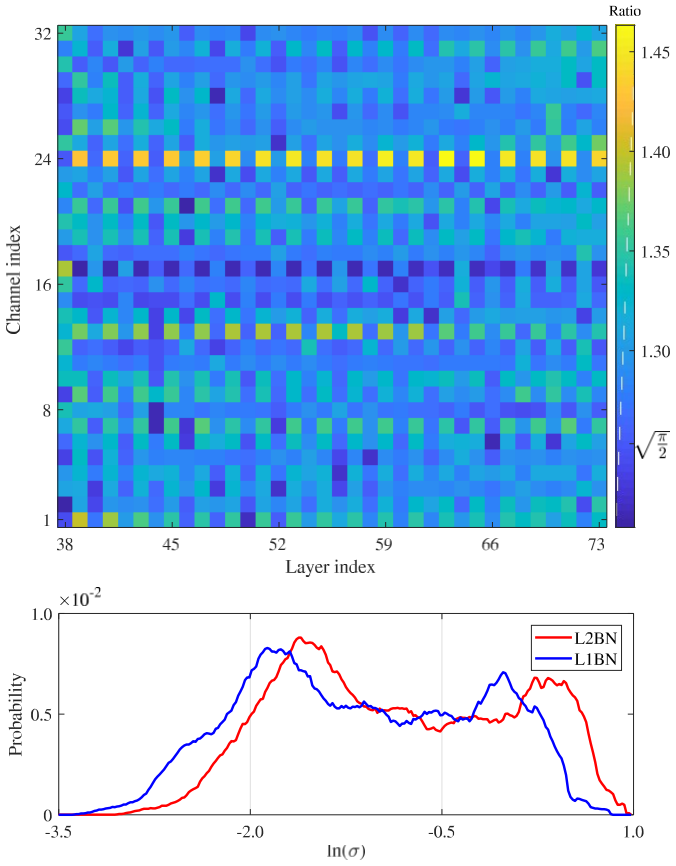


Fig. 3. Colormap of layerwise and channelwise ratios $\sigma_{L_2}/\sigma_{L_1}$ averaged across 100 minibatches (top). Probability histograms of σ_{L_2} and σ_{L_1}, the axis x is on a logarithmic scale (bottom).

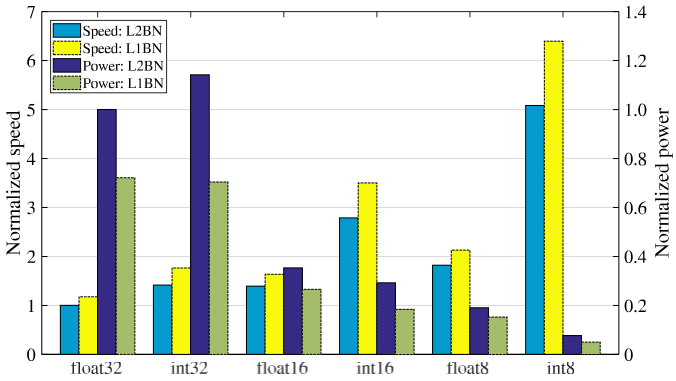


Fig. 4. Estimated time and power consumption for $L1$-norm and $L2$-norm.

on ASIC devices with reduced resources. In Section IV-A, we have pointed out that the effective number of data points for normalization equals to m and mhw for a fully connected layer and a convolution layer, respectively. Since spatial statistics are always coupled with minibatch statistics, we use m to denote the total number of intrachannel 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 (4), (8), (11), and (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 Fig. 4, the time and power consumption of L1BN and L2BN are estimated with CNN models in Table I. 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 digital signal processor and floating-point unit are not available.

VI. DISCUSSION AND CONCLUSION

To reduce the overheads of $L2$-norm based BN, we propose the $L1$-norm-based BN. L1BN is equivalent to L2BN by multiplying a scaling factor $\sqrt{(\pi/2)}$ on condition that 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 the ASIC synthesis, L1BN is able to obtain 25% speedup and 37% energy saving, as well as the reduction of hardware resources, e.g., silicon area and cost. Other than feed-forward models, $L1$-norm might be applied into recurrent BN [24] and decreases much more overheads. Inspired by L1BN, the exploration and interpretation of the intrinsic probabilistic principle behind the normalization statistics [11], [46] remain as intriguing directions for future research.

Previous deep learning hardware mainly target at accelerating the offline 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 the future artificial general intelligence. Thus, the 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 BN 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, pp. 436–444, May 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[3] D. Amodei *et al.*, "Deep speech 2: End-to-end speech recognition in English and mandarin," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 173–182.

[4] Y. Wu *et al.* (2016). "Google's neural machine translation system: Bridging the gap between human and machine translation." [Online]. Available: https://arxiv.org/abs/1609.08144

[5] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[6] A. Karpathy, A. Joulin, and L. Fei-Fei, "Deep fragment embeddings for bidirectional image sentence mapping," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 1889–1897.

[7] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.*, 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 *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1026–1034.

[9] D. J. C. MacKay, "Bayesian methods for adaptive models," Ph.D. dissertation, Dept. Comput. Neural Syst., California Inst. Technol., Pasadena, CA, USA, 1992.

[10] R. M. Neal, "Bayesian learning for neural networks," Ph.D. dissertation, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 1995.

[11] M. Teye, H. Azizpour, and K. Smith. (2018). "Bayesian uncertainty estimation for batch normalized deep networks." [Online]. Available: https://arxiv.org/abs/1802.06455

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[13] A. Radford, L. Metz, and S. Chintala. (2015). "Unsupervised representation learning with deep convolutional generative adversarial networks." [Online]. Available: https://arxiv.org/abs/1511.06434

[14] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2234–2242.

[15] X. Huang, Y. Li, O. Poursaeed, J. E. Hopcroft, and S. J. Belongie, "Stacked generative adversarial networks," in *Proc. CVPR*, vol. 2, 2017, p. 3.

[16] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 807–814.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[18] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2017, pp. 1–4.

[19] A. G. Howard *et al.* (2017). "MobileNets: Efficient convolutional neural networks for mobile vision applications." [Online]. Available: https://arxiv.org/abs/1704.04861

[20] R. Zhao *et al.*, "Accelerating binarized convolutional neural networks with software-programmable FPGAs," in *Proc. FPGA*, 2017, pp. 15–24.

[21] 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 *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2017, pp. 1–6.

[22] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 901–909.

[23] J. L. Ba, J. R. Kiros, and G. E. Hinton. (2016). "Layer normalization." [Online]. Available: https://arxiv.org/abs/1607.06450

[24] T. Cooijmans, N. Ballas, C. Laurent, Ç. Gülçehre, and A. Courville. (2016). "Recurrent batch normalization," [Online]. Available: http://arxiv.org/abs/1603.09025

[25] S. Ioffe, "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1945–1953.

[26] M. Lin, Q. Chen, and S. Yan. (2013). "Network in network." [Online]. Available: https://arxiv.org/abs/1312.4400

[27] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5987–5995.

[28] F. Chollet. (2016). "Xception: Deep learning with depthwise separable convolutions." [Online]. Available: https://arxiv.org/abs/1610.02357

[29] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4107–4115.

[30] Q. He *et al.* (2016). "Effective quantization methods for recurrent neural networks." [Online]. Available: https://arxiv.org/abs/1611.10176

[31] L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li, "GXNOR-Net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework," *Neural Netw.*, vol. 100, pp. 49–58, Apr. 2018.

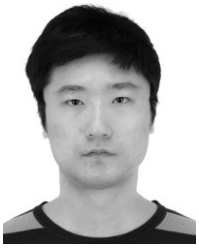
[32] S. Wu, G. Li, F. Chen, and L. Shi. (2018). "Training and inference with integers in deep neural networks." [Online]. Available: https://arxiv.org/abs/1802.04680

[33] H. Xiao, K. Rasul, and R. Vollgraf. (2017). "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms." [Online]. Available: https://arxiv.org/abs/1708.07747

[34] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, vol. 2011, no. 2, 2011, p. 5.

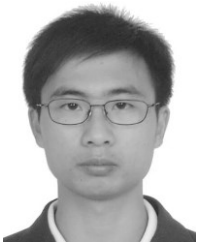
[35] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009, p. 7, vol. 1.

[36] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.
[37] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao. (2015). "LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop." [Online]. Available: https://arxiv.org/abs/1506.03365
[38] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, vol. 16, 2016, pp. 265–283.
[39] *Tensorflow Performance Guide*. Accessed: 2018. [Online]. Available: https://www.tensorflow.org/performance/performance_guide
[40] F. S. Leone, L. S. Nelson, and R. B. Nottingham, "The folded normal distribution," *Technometrics*, vol. 3, no. 4, pp. 543–550, 1961.
[41] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[42] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Proc. 18th Int. Conf. Artif. Intell. Statist.*, 2015, pp. 562–570.
[43] N. Silberman and S. Guadarrama. *TensorFlow-Slim Image Classification Model Library*. Accessed: 2016. [Online]. Available: https://github.com/tensorflow/models/tree/master/research/slim
[44] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of Wasserstein GANs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5767–5777.
[45] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.
[46] J. Bjorck, C. Gomes, B. Selman, and K. Q. Weinberger. (2018). "Understanding batch normalization." [Online]. Available: https://arxiv.org/abs/1806.02375



Shuang Wu received the B.E. degree in mechanical engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2015. He is currently pursuing the Ph.D. degree with the Department of Precision Instrument, Tsinghua University, Beijing, China.

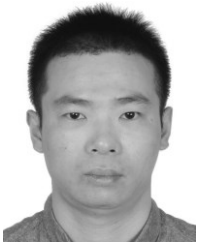
His current research interests include deep learning, neuromorphic chip architecture, and reinforcement learning.



Guoqi Li (M'12) received the B.Eng. degree from the Xi'an University of Technology, Xi'an, China, in 2004, the M.Eng. degree from Xi'an Jiaotong University, Xi'an, China, in 2007, and the Ph.D. degree from Nanyang Technological University, Singapore, in 2011.

From 2011 to 2014, he was a Scientist with the Data Storage Institute and the Institute of High Performance Computing, Agency for Science, Technology and Research, Singapore. Since 2014, he has been with the Department of Precision Instrument, Tsinghua University, Beijing, China, where he is currently an Associate Professor. He has authored or co-authored more than 80 journal and conference papers. His current research interests include brain-inspired computing, complex systems, machine learning, neuromorphic computing, and system identification.

Dr. Li has been actively involved in professional services such as serving as an International Technical Program Committee Member and a Track Chair for international conferences. He is an Editorial-Board Member and a Guest Associate Editor for *Frontiers in Neuroscience* (Neuromorphic Engineering section). He serves as a reviewer for a number of international journals.



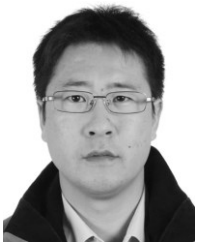
Lei Deng (M'18) received the B.E. degree from the University of Science and Technology of China, Hefei, China, in 2012, and the Ph.D. degree from Tsinghua University, Beijing, China, in 2017.

He is currently a Post-Doctoral Researcher with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA, USA. His current research interests include computer architecture, machine learning, computational neuroscience, tensor analysis, and complex systems.

Dr. Deng is a Guest Associate Editor for *Frontiers in Neuroscience*.

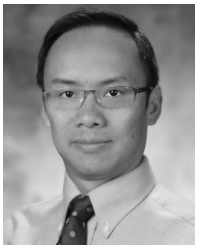
Liu Liu received the B.S. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2013, and the Ph.D. degree from the University of California at Santa Barbara, Santa Barbara, CA, USA, in 2015, where he is currently pursuing the Ph.D. degree with the Department of Computer Science.

His current research interests include deep learning, computer architecture, and emerging nonvolatile memory.



Dong Wu received the B.S. degree in electronic engineering from Xi'an Jiaotong University, Xi'an, China, in 2011, and the Ph.D. degree in microelectronics from Tsinghua University, Beijing, China, in 2006.

He is currently an Associate Professor with the Institute of Microelectronics, Tsinghua University. His current research interests include circuit design for sensors and memories.



Yuan Xie (F'15) received the B.S. degree from the Electrical Engineering Department, Tsinghua University, Beijing, China, in 1997, and the M.S. and Ph.D. degrees from the Electrical Engineering Department, Princeton University, Princeton, NJ, USA, in 1999 and 2002, respectively.

From 2002 to 2003, he was with IBM, Armonk, NY, USA. From 2012 to 2013, he was with the AMD Research China Laboratory, Beijing, China. From 2003 to 2014, he was a Professor with Pennsylvania State University, State College, PA, USA. He is currently a Professor with the Electrical and Computer Engineering Department, University of California at Santa Barbara, Santa Barbara, CA, USA. His current research interests include computer architecture, electronic design automation, and very large scale integration design.

Dr. Xie is an Expert in computer architecture who has been inducted to ISCA/MICRO/HPCA Hall of Fame. He served as the TPC Chair for HPCA 2018. He is the Editor-in-Chief for the *ACM Journal on Emerging Technologies in Computing Systems*, a Senior Associate Editor for *ACM Transactions on Design Automation for Electronics Systems*, and an Associate Editor for the IEEE TRANSACTIONS ON COMPUTERS.

Luping Shi (M'02) received the Ph.D. degree in physics from the University of Cologne, Cologne, Germany, in 1992.

In 1993, he was a Post-Doctoral Fellow with the Fraunhofer Institute for Applied Optics and Precision Instrument, Jena, Germany. From 1994 to 1996, he was a Research Fellow with the Department of Electronic Engineering, City University of Hong Kong, Hong Kong. From 1996 to 2013, he was with the Data Storage Institute, Singapore, as a Senior Scientist and a Division Manager, and led nonvolatile solid-state memory, artificial cognitive memory, and optical storage researches. In 2013, he joined Tsinghua University, Beijing, China, as a National Distinguished Professor. By integrating seven departments, he established the Center for Brain Inspired Computing Research, Tsinghua University, in 2014, and served as the Director. He has authored or co-authored more than 200 papers in prestigious journals including *Science*, *Nature Photonics*, *Advanced Materials*, and *Physical Review Letters*. His current research interests include brain-inspired computing, neuromorphic chip, memory devices, and so on.

Dr. Shi is an SPIE Fellow. He was a recipient of the National Technology Award 2004 Singapore.