Constrained Weight Optimization for Learning without Activation Normalization

Daiki Ikami Go Irie Takashi Shibata
NTT Corporation
daiki.ikami.ef@hco.ntt.co.jp goirie@ieee.org t.shibata@ieee.org

Abstract

Weight Normalization (WN) is an essential building block in deep learning. However, even state-of-the-art WN methods need to be combined with activation normalization methods, such as Batch Normalization (BN), to provide the same classification accuracy as BN. In this paper, we aim to circumvent this issue with a weight normalization approach that can be used on its own to provide a classification accuracy competitive to BN. Our approach mimics three fundamental properties of BN, namely, keeping the norm of the weights constant, setting the mean of the weights to zero, and simulating stochastic perturbations due to batch sampling bias. Unlike most of the existing WN methods that rely on "reparametrization", our method directly optimizes the weights with proper constraints and thus can circumvent its serious drawback, gradient explosion. Moreover, we propose an efficient and easy-to-implement algorithm to solve our constrained optimization problem without sacrificing its benefits. The results of classification experiments on three popular benchmark datasets demonstrate that our method is highly competitive with or even better than the state-of-the-art normalization methods.

1. Introduction

Deep neural networks have become a standard tool in many pattern recognition tasks. A considerable amount of effort has revealed the essential ingredients for successful deep neural network training. Batch Normalization (BN) [12] is currently a representative building block in general deep neural network training to improve the training stability and final generalization performance. With its success, significant shortcomings of BN (e.g., performance is heavily dependent on mini-batch size [15]) have also been revealed, and a number of extensions in this vein have been proposed [32, 2, 30, 21, 19]. In contrast to these "activation normalization" methods, another approach called Weight Normalization (WN) has also been explored [24, 1, 10, 22]. It aims to achieve the same effect as activation normalization by normalizing only the weights of the network instead of the activation values. The advantage of WN is that the normalization is input- and output-independent, which naturally avoids some of the significant problems plaguing BN, e.g., dependence on mini-batch size. Moreover, normalized weights are precomputable, which means that WN has the potential to provide a simpler and more efficient scheme. In this work, we investigate WN for stable and effective training of deep neural networks for classification tasks.

Stemming from the pioneering work by [24, 1], several WN methods have been proposed [10, 22]. The most common approach is to "reparameterize" each weight vector in the network as a normalized version of another parameter vector and optimize the parameter vector instead of the original weight vector. For example, Salimans et al. [24] reparameterize a weight vector using two types of parameters representing its length and direction, respectively. Weight Standardization (WS) [22] models each weight vector as the result of the standard normalization of the parameter vector. However, the issue with these methods is that their performance is still lower than BN [24, 22]. To achieve the same level of accuracy, they need to be combined with activation normalization methods such as BN or Group Normalization (GN) [32]. Consequently, the network no longer enjoys the potential benefit of the framework of WN, namely, being able to avoid additional computations for inference.

In this work, we propose a weight normalization method that can solely achieve the same performance level as BN. The processing of BN is characterized by three fundamental properties: making the variance of the activation values constant, making the mean of the activation values zero, and sampling mini-batches stochastically. Our method aims to reproduce these properties as much as possible within the framework of WN. Some existing WN methods [24, 1, 10, 22] have also attempted to reproduce these properties (in part) based on the reparameterization approach. However, reparameterization has a serious drawback in that the variance of the unnormalized parameter vectors appears in the gradients, resulting in a gradient ex-

plosion problem. We argue that this is one of the main reasons the existing WN methods perform unsatisfactorily unless combined with activation normalization methods. As an alternative, we formulate an optimization problem constrained to satisfy the above BN properties and thereby avoid the gradient explosion by directly obtaining normalized weights, without relying on reparameterization. We propose a simple algorithm that can efficiently solve the constrained optimization problem. Experiments in classification tasks using three popular benchmark datasets demonstrate that our method can achieve highly competitive or even better performance compared to the state-of-the-art normalization methods.

2. Related Works

In machine learning, it has long been known that normalization of input data contributes to improving the stability, speed, and final accuracy of the training process. The impact of normalization is also prominent in deep learning, which often uses unbounded activation functions (e.g., ReLU), and its importance has been widely recognized since its dawn. Early examples like Local Contrast Normalization (LCN) [13] and Local Response Normalization (LRN) [16] aim to extract stable features by normalizing the contrast of the input image (or feature map) locally.

BN [12] is the first example of trainable normalization layers that normalizes the layer output in a global sense. Despite excellent performance on a variety of tasks, BN is not always desirable. First, it is known that when the batch size is small, the batch statistics used for normalization become unstable and cause serious performance degradation [32]. A more universal property is that there is no mini-batch processing during inference. There is always a stochastic gap between the models during training and inference, so while they might match asymptotically they never match perfectly. Second, the reason for the success has not been fully clarified [25]. While many studies have analyzed the reasons BN contributes to training stability, there are still different viewpoints on this, such as 1) suppressing internal covariate shifts [12], 2) preventing the mean and variance of the network output from increasing exponentially as the layers get deeper [3], and 3) preventing the vanishing/exploding gradient problem during training through the suppression of Lipschitz constant [25].

Motivated by these weaknesses and insights, several normalization methods have been proposed. The majority of these can be grouped into two approaches, activation normalization and weight normalization (WN).

Activation Normalization. Activation normalization was originally designed to suppress the adverse effect of the internal covariate shift [12]. The difference between the existing methods mainly lies in which dimensions are normalized (batch, channel, height, and width) and at what granularity. For example, Layer Normalization (LN) [2] normalizes the set of activations along the channel direction. Instance Normalization (IN) [30] resembles LN but normalizes the input across each channel in each training sample. In Group Normalization (GN) [32], the normalization is performed over groups of channels to address the issue induced by the small batch size. To deal with the issue of small batches, Batch Renormalization (BRN) [11] has also been proposed. Other variants [26] or their combinations [21, 19, 36, 21] have also been explored. Unfortunately, these methods often do not perform as well as BN. Furthermore, as with BN, normalization depends on the input and output, which requires additional computations for normalization during inference.

Weight Normalization. Instead of normalizing the layer input/output, a few recent methods aim to achieve similar effects by normalizing only the weights of the network. In the original weight normalization method presented by Salimans et al. [24], each weight is decomposed into a length and a vector and is updated during optimization on the basis of both of these gradients. In Weight Standardization (WS) [22] and Centered Weight Normalization [10], the weight is renormalized by the mean and the variance of each weight. WN is generally much simpler than activation normalization, as there is no additional computation for normalization, even during inference, and there is no disadvantage in terms of depending on the batch size or mini-batch statistics. However, it is widely known that the performance of WN alone is inferior to that of BN, and it can only achieve accuracy comparable to BN when combined with some activation normalization method. For example, WN can only achieve the same level of accuracy as BN when combined with mean-only batch normalization [24]. In other words, WS needs to be combined with GN to be competitive with BN [22].

Unlike existing WN methods, the method proposed in this paper does not require any activation normalization method to compete with BN. Our proposed method is, to the best of our knowledge, the first that achieves the same level of accuracy as BN in classification tasks without a combination of activation normalization methods.

3. Proposed Method

We begin our discussion with the standard learning process of deep convolutional neural networks. Let us denote a convolution operation performed in a single layer by $\mathbf{y} = \mathbf{W}\mathbf{x}$, where $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{W}$ are the input, output, and trainable weights of a layer, respectively. Without loss of generality, we assume that the bias is zero. The weights $\mathbf{W}$ are typically learned by optimizing some loss function

$\mathcal{L}$, e.g., softmax cross entropy, with respect to them by using some gradient descent method, e.g., Stochastic Gradient Descent (SGD).

$$\min_{\{\mathbf{W}^{(l)}\}} \mathcal{L}(\{\mathbf{W}^{(l)}\}), \ \forall l. \tag{1}$$

where $\mathbf{W}^{(l)}$ is the weights of the l-th layer.

In this paper, we aim to normalize the weight parameters for stable and effective training. To this end, instead of solving Eq. (1), we obtain the normalized weights directly by solving the following constrained optimization problem.

$$\begin{aligned} &\min_{\{\mathbf{W}^{(l)}\}} \mathcal{L}(\{\mathbf{W}^{(l)}\}), \ \forall l \\ &\text{s.t.} \ \ \|\mathbf{w}_i^{(l)}\|_2^2 = c^{(l)}, \mathbf{1}^\top \mathbf{w}_i^{(l)} = 0, \forall i, l, \end{aligned} \tag{2}$$

where $\mathbf{w}_i^{(l)}$ is a weight vector corresponding to i-th output in l-th layer and $\mathbf{1}$ is the all-one vector. The two constraints which we call **sphere constraint** and **centralization constraint** are imposed to imitate the fundamental property of BN: the former keeps the weight of the norm constant, and the latter constrains the weights to be centered. We also introduce another concept, **noise injection**, to replicate the gap between testing and training caused by stochastic mini-batch sampling. As discussed later in Section 3.5, one key benefit of our optimization-based approach is that it does not suffer from the gradient explosion problem, unlike existing WN methods based on the reparameterization approach [24, 1, 10, 22].

Various constraints for weight optimization $\mathbf{W}$ have been proposed to date. For example, weight decay is an implementation of a ridge regularization and is widely used to improve generalization performance. Spectral Normalization [20] is another example proposed to stabilize the training of GANs. In addition to these, there are several other methods aimed at learning unbiased [34] or normalized weights [9] to improve the stability of training. Our method is designed to mimic BN and therefore uses a different set of constraints from these methods.

3.1. Sphere Constraint

It is widely known that keeping the distribution of the activation values fixed before and after a layer is effective in stabilizing learning; in BN, the efficient stabilization is achieved by controlling the distribution statistics of the input, especially the first and second moments [12]. Moreover, it is known that the variance of the output $\mathbf{y}$ is equal to the norm of the weights $\|\mathbf{w}_i^{(l)}\|_2^2$, under the assumption that x is whitened [24]. Hence, applying BN to $\mathbf{y}$ is equivalent to normalizing the weights. Following these observations, we impose the first constraint named Sphere Constraint to strictly control the second moment of the weights $\mathbf{W}^{(l)}$ for each layer to a constant value $c^{(l)}$.

While $c^{(l)}$ can be an arbitrary bounded constant, it is possible to ensure the uniformity of the distributions over multiple layers under certain assumptions [7]. Specifically, let us assume that $\mathbf{W}^{(l)}$'s are mutually independent, $\mathbf{x}^{(l)}$'s are also mutually independent, and $\mathbf{W}^{(l)}$ and $\mathbf{x}^{(l)}$ are independent of each other. As given in [7], under these assumptions, the variance of the activation of the l-th layer $\mathbf{y}^{(l)}$ can be written as a function of that of $\mathbf{y}^{(l-1)}$ as

$$\text{Var}[\mathbf{y}^{(l)}] = \frac{n_l}{2}\text{Var}[\mathbf{w}^{(l)}]\text{Var}[\mathbf{y}^{(l-1)}], \tag{3}$$

where n_l is the number of input dimensions of the l-th layer. In order to equalize the variance of these two outputs, a sufficient condition is

$$\frac{n_l}{2}\text{Var}[\mathbf{w}^{(l)}] = 1, \tag{4}$$

which can be satisfied by setting the constant as $c^{(l)} = 2, \forall l$. Unless otherwise stated, we will use this setting hereafter.

3.2. Centralization Constraint

The second constraint, named Centralization Constraint, is introduced to suppress the harmful effects stemming from the bias of the activation values by controlling the first moment of the weight distribution. The presence of the bias of the activation values is known to cause an undesirable increase in the range of the activation values as the layers get deeper, which also has an undesirable effect on the gradient values [24, 10]. By making the average of the weights zero, i.e., $\mathbf{w}^\top \mathbf{1} = 0$, we have

$$\begin{aligned} \mathbf{w}^\top \left(\mathbf{x} - \frac{1}{n}(\mathbf{1}^\top \mathbf{x})\mathbf{1} \right) &= \mathbf{w}^\top \mathbf{x} - \frac{1}{n}(\mathbf{1}^\top \mathbf{x})\mathbf{w}^\top \mathbf{1} \\ &= \mathbf{w}^\top \mathbf{x}. \end{aligned} \tag{5}$$

Here, $\frac{1}{n}(\mathbf{1}^\top \mathbf{x})\mathbf{1}$ gives the mean of the input $\mathbf{x}$. This suggests that by introducing the Centralization Constraint on the weights, we can make the output activation values unbiased even if the input is biased.

These two constraints, Sphere Constraint and Centralization Constraint, require that the variance and the mean be the same in all the layers, which may reduce the representation ability of the entire network. In BN (and other activation normalization methods like GN), new scale and bias parameters are introduced and learned to restore the representation ability deteriorated by the normalization. Following this idea, we introduce two parameters for improving the representation ability. Formally, the l-th layer's output value $y_i^{(l)}$ is given by

$$y_i = \gamma_i^{(l)} x_i + \beta_i^{(l)}, \tag{6}$$

where $\gamma_i^{(l)}$ and $\beta_i^{(l)}$ are the scale and bias parameters, respectively.

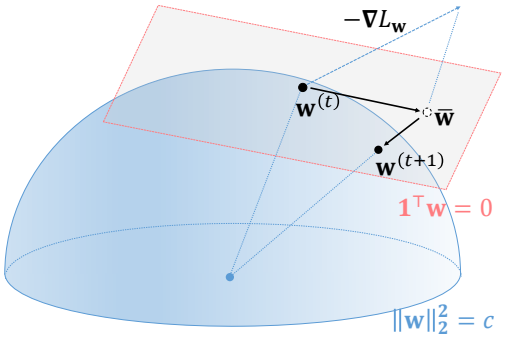


Figure 1: An illustration of our projected gradient descent. The solution is updated along the zero-mean gradient direction and then normalized to satisfy the constraint.

3.3. Noise injection

During the training, BN uses the mini-batch statistics (mean and standard deviation) to normalize the activation values. These are sample statistics for the entire training data and vary randomly, depending on the mini-batch samples. In contrast, in the case of WN, such randomness never exist because the normalization does not depend on the mini-batch. In general, it is known that introducing reasonable randomness often improves the final generalization performance of a network. Typical examples include dropout, data augmentation, and, more recently, NoisyStudent [28]. Even in BN, this discrepancy due to the above randomness is believed to impact performance [27].

Assuming that the randomness of BN is a factor in improving performance, we inject random noise into our WN training to mimic the behavior of BN. Specifically, instead of using Eq. (6), we compute the output y_i during the training as

$$y_i = (1 + n_i)\gamma_i x_i + \beta_i, \tag{7}$$

where $n_i \sim N(0, \sigma_l^2)$ is the noise injected to simulate the randomness from the mini-batch statistics of BN. A simple selection of σ_l^2 is to use a certain constant σ uniformly in every layer. However, we find that linearly increasing σ_l in the number of layers L achieves better performance for residual-type networks (e.g., ResNet):

$$\sigma_l = \frac{l}{L}\sigma. \tag{8}$$

We use $\sigma_l = \sigma$ for non-residual-type networks.

3.4. Optimization

For solving our problem in Eq. (2), we utilize projected gradient descent through solving the following constrained optimization problem:

$$\begin{aligned}\mathbf{w}^{(t+1)} = &\min_{\mathbf{w}} \left\|\mathbf{w} - \left(\mathbf{w}^{(t)} - \alpha\nabla_{\mathbf{w}}L(\mathbf{w})\right)\right\|_2^2 \\ &\text{s.t. } \|\mathbf{w}\|_2^2 = c, \mathbf{1}^\top\mathbf{w} = \mathbf{0},\end{aligned} \tag{9}$$

where α is the learning rate. For simplicity, we drop the subscripts l and i. The Lagrangian function associated with Eq. (9) is

$$\left\|\mathbf{w} - \left(\mathbf{w}^{(t)} - \alpha\nabla_{\mathbf{w}}L(\mathbf{w})\right)\right\|_2^2 + \lambda_1\left(\|\mathbf{w}\|_2^2 - c\right) + \lambda_2\mathbf{1}^\top\mathbf{w}, \tag{10}$$

where λ_1 and λ_2 are Lagrange multipliers. From the derivative of Eq. (10), we can obtain the solution of the optimization problem in Eq. (9):

$$\mathbf{w} = \sqrt{c}\frac{\mathbf{w}^{(t)} - \alpha\left(\nabla_{\mathbf{w}}L(\mathbf{w}) - (\mathbf{1}^\top\nabla_{\mathbf{w}}L(\mathbf{w}))\mathbf{1}/n\right)}{\left\|\mathbf{w}^{(t)} - \alpha\left(\nabla_{\mathbf{w}}L(\mathbf{w}) - (\mathbf{1}^\top\nabla_{\mathbf{w}}L(\mathbf{w}))\mathbf{1}/n\right)\right\|}, \tag{11}$$

where n is the number of input dimensions. The update rule of $\mathbf{w}$ by gradient decent is

$$\mathbf{w}^{(t+1)} = (1-\beta)\mathbf{w}^{(t)} - \alpha'\left(\nabla_{\mathbf{w}}L(\mathbf{w}) - (\mathbf{1}^\top\nabla_{\mathbf{w}}L(\mathbf{w}))\mathbf{1}/n\right), \tag{12}$$

where coefficients β and α' are given by

$$\beta = 1 - \frac{1}{\|\mathbf{w}^{(t)} - \alpha(\nabla_{\mathbf{w}}L(\mathbf{w}) - (\mathbf{1}^\top\nabla_{\mathbf{w}}L(\mathbf{w}))\mathbf{1})/n\|}, \tag{13}$$

$$\alpha' = \frac{\alpha}{\|\mathbf{w}^{(t)} - \alpha(\nabla_{\mathbf{w}}L(\mathbf{w}) - (\mathbf{1}^\top\nabla_{\mathbf{w}}L(\mathbf{w}))\mathbf{1})/n\|}. \tag{14}$$

The proposed optimization in Eq. (12) can be easily implemented by a two-step approach in popular deep learning libraries such as PyTorch and TensorFlow. Figure 1 shows the image of our projected gradient descent. First, the parameters are updated along the centralized gradient direction. Then we obtain updated parameters by normalizing weights to satisfy the norm constraint. This is as efficient as the standard SGD because it requires only mean subtraction and scaling. The pseudo-code is shown in Algorithm 1.

Algorithm 1 Weight update rules of our method

Input: initial weights of a neural network $\mathbf{w}^{(1)}$, learning rate α,
for $t = 1$ to T **do**
 # Compute forward and backward and obtain the gradient $\nabla_{\mathbf{w}}L$:
 # update with zero-mean gradient:
$\tilde{\mathbf{w}} \leftarrow \mathbf{w}^{(t)} - \alpha\left(\nabla_{\mathbf{w}}L - (\mathbf{1}^\top\nabla_{\mathbf{w}}L)\mathbf{1}/n\right)$
 # update with norm-constrained weight:
$\mathbf{w}^{(t+1)} \leftarrow \sqrt{c^{(l)}}\tilde{\mathbf{w}}/\|\tilde{\mathbf{w}}\|$
end for

Table 1: Comparison of top-1 error on the CIFAR-10 and CIFAR-100 datasets with various network architectures. The best and second-best results are denoted in bold and italics, respectively.

	Augmentation	Network	Ours	BN	GN	WS+BN	WS+GN
CIFAR-10	Standard	CNN	5.50	**5.36**	6.45	*5.43*	5.71
		ResNet-110	*5.65*	5.68	6.34	**5.52**	6.69
		WideResNet-28-10	*4.19*	**3.70**	5.01	4.78	4.67
	FastAA	CNN	**4.18**	4.40	4.93	4.65	*4.37*
		ResNet-110	4.26	*4.11*	6.76	**3.92**	5.85
		WideResNet-28-10	**2.59**	*2.64*	3.68	3.42	3.29
CIFAR-100	Standard	CNN	**24.26**	*25.11*	29.28	25.51	26.27
		ResNet-110	**25.16**	26.28	29.75	*25.55*	28.51
		WideResNet-28-10	**18.31**	*18.48*	22.63	19.82	22.62
	FastAA	CNN	**21.03**	22.23	25.34	*21.87*	22.31
		ResNet-110	*21.87*	**21.84**	28.90	22.10	27.83
		WideResNet-28-10	**16.40**	*16.57*	19.96	19.18	19.41

3.5. Discussion: Gradient explosion problem caused by reparameterization

We discuss the weakness of the weight normalization approaches based on reparametrization [24, 1, 10, 22]. In existing WN methods based on reparameterization, the weights are modeled as normalized versions of some parameters, making the output values aligned during the forward propagation. For example, WS[1] replaces the convolution operation $\mathbf{y} = \mathbf{W}\mathbf{x}$ with $\mathbf{y} = \hat{\mathbf{W}}\mathbf{x}$, where $\hat{\mathbf{W}} = [\hat{\mathbf{w}}_1^\top, \cdots, \hat{\mathbf{w}}_{n_o}^\top]$ are reparametrized weights as a function of $\mathbf{W}$, specifically defined as

$$\hat{\mathbf{w}}_i = \frac{\mathbf{w}_i - \mu_{\mathbf{w}_i}}{\sigma_{\mathbf{w}_i}}, \tag{15}$$

where $\mu_{\mathbf{w}_i}$ and $\sigma_{\mathbf{w}_i}$ are the mean and the standard deviation of a weight vector given by

$$\mu_{\mathbf{w}_i} = \frac{1}{n_i}\mathbf{w}_i^\top \mathbf{1}, \tag{16}$$

$$\sigma_{\mathbf{w}_i} = \sqrt{\frac{1}{n_i}\sum_j^{n_i}(w_{ij} - \mu_{\mathbf{w}_i})^2}. \tag{17}$$

By this reparameterization, the output $\mathbf{y}$ for the forward propagation becomes scaling-invariant [28]; even if the weight $\mathbf{W}$ is scaled by the factor α as $\mathbf{W} \leftarrow \alpha\mathbf{W}$, the output $\mathbf{y}$ is invariant by using $\hat{\mathbf{W}}$. In general, however, this possibly induces the gradient explosion problem, i.e., the gradient of the loss function may grow extremely large in the backpropagation step. The gradient of the loss function $\mathcal{L}$ is formally given by

$$\nabla_{\mathbf{w}_i}\mathcal{L} = \nabla_{\hat{\mathbf{w}}_i}\mathcal{L} - \frac{1}{n_i}(\mathbf{1}^\top \nabla_{\hat{\mathbf{w}}_i}\mathcal{L})\mathbf{1}, \tag{18}$$

$$\nabla_{\hat{\mathbf{w}}_i}\mathcal{L} = \frac{1}{\sigma_{\mathbf{w}_i}}\left(\nabla_{\hat{\mathbf{w}}_i}\mathcal{L} - \frac{1}{n_i}\langle \hat{\mathbf{w}}_i, \nabla_{\hat{\mathbf{w}}_i}\mathcal{L}\rangle \hat{\mathbf{w}}_i\right), \tag{19}$$

where $<\dot{>}$ denotes the dot product. The update rule by gradient descent with weight decay parameter β is

$$\mathbf{w}^{(t+1)} = (1-\beta)\mathbf{w}^{(t)} - \alpha'\left(\nabla_{\hat{\mathbf{w}}}L(\mathbf{w}) - (\mathbf{1}^\top \nabla_{\hat{\mathbf{w}}}L(\mathbf{w}))\mathbf{1}/n_i\right) \tag{20}$$

As shown in Eq. (19), the scale of the gradient magnitude is controlled by the standard deviation of the unnormalized weights $\sigma_{\mathbf{w}_i}$, which means that the gradient $\nabla_{\mathbf{W}}\mathcal{L}$ becomes dramatically large when $\sigma_{\mathbf{w}_i}$ becomes small during training. This is a reason why the existing WN methods need to be coupled with activation normalization methods like BN or GN to reduce the dependence of the gradients on the scale of the weights.

In contrast, our approach constrains the norm of the weights $\mathbf{w}$ and updates them without dividing the gradient $\nabla_{\mathbf{W}}\mathcal{L}$ by the standard deviation $\sigma_{\mathbf{w}_i}$ (see Eq. (12)), which means it can naturally avoid the gradient explosion problem and achieve stable training.

4. Experiments

In this section, we empirically investigate the performance of our proposed method in image classification and 3D point cloud classification.

4.1. Results on CIFAR-10/CIFAR-100

We first evaluate our proposed method with the CIFAR-10 and CIFAR-100 datasets. Both CIFAR-10 and CIFAR-

[1]We pick up WS [22] as an example here, but the same can be applied to other methods based on reparameterization (e.g., [24, 1, 10]).

Table 2: Results for small batch size training on the CIFAR-10 and CIFAR-100 datasets. WideResNet-28-10 and FastAutoAugment are used as the base network and data augmentation, respectively. B denotes the batch size.

	Ours			BN		
	$B = 4$	$B = 16$	$B = 128$	$B = 4$	$B = 16$	$B = 128$
CIFAR-10	2.65	2.64	2.59	3.08	2.80	2.64
CIFAR-100	15.12	14.80	16.40	17.98	16.37	16.57

100 consist of 32×32 RGB color images, with 50,000 images for training and 10,000 images for testing. We compare our method with major activation and weight normalization methods, specifically, BN [12], GN [32], WS with BN (WS+BN) [22], and WS with GN (WS+GN) [22].

4.1.1 Setup

We used all 50,000 training images for training the networks of three architectures from scratch: 9-layer CNN, ResNet-110 [8], and WideResNet-28-10 [35]. The configuration of the 9-layer CNN is: Conv(3,64) - Conv(3,64) - Conv(3,64) - AvgPool(2) - Conv(3,128) - Conv(3,128) - Conv(3,128) - AvgPool(2) - Conv(3,256) - Conv(3,256) - Conv(3,256) - GAP - Linear, where Conv(k,c) denotes a convolution-BN-ReLU layer with the kernel size $k \times k$ and output channel c, AvgPool(2) denotes average pooling with stride 2, and GAP is global average pooling. Our method is implemented by replacing all BN layers with scale and bias layers Eq. (6). In additio to the standard data augmentation strategy using mirroring and random cropping, we also test a more advanced data augmentation strategy using Fast AutoAugment [17] coupled with Cutout [4] (FastAA) for practical performance evaluation.

Except for the small batch size training discussed in Section 4.1.3, all DNN models are trained for 200 epochs using one GPU with batch size 128. We use SGD with a momentum of 0.9, whose learning rate is decayed by cosine shape learning rate scheduling [18]. For our method, the initial learning rate and weight decay are respectively set to 0.1 and 1.0×10^{-3} for CNN, 0.2 and 5.0×10^{-3} for WideResNet, and 0.2 and 1.0×10^{-3} for ResNet. We do not apply weight decay for the trainable scale and bias parameters (γ and β in Eq. (6)) for our method. For GN, we set the number of groups to 16.

4.1.2 Main results

We report the test errors at the end of the training. Table 1 shows the results of our method and all the baselines with the three network architectures. Overall, our method achieved a highly competitive or even better performance compared to all the baselines. It achieved the best performance when using WideResNet-28-10 and FastAA; 2.59 on CIFAR-10 and 16.40 on CIFAR-100. On the CIFAR-100 dataset, our method outperformed all the others except for the case using ResNet-110 with FastAA. Note that the proposed method is equal to or better than BN without combining any activation normalization methods. This is a clear difference from existing weight normalization methods that must always be coupled with an activation normalization method (e.g., WS+BN and WS+GN). These results demonstrate a clear advantage of our method.

Table 3: Ablation study of the proposed method on the CIFAR-10 and CIFAR-100 datasets. WideResNet-28-10 with standard data augmentation is used.

	centerize	spherize	$\sigma = 0.0$	$\sigma = 1.0$
CIFAR-10			5.36	4.97
	✓		5.58	4.66
		✓	5.68	5.29
	✓	✓	4.45	4.19
CIFAR-100			24.15	21.85
	✓		23.43	20.43
		✓	22.76	20.88
	✓	✓	20.61	18.31

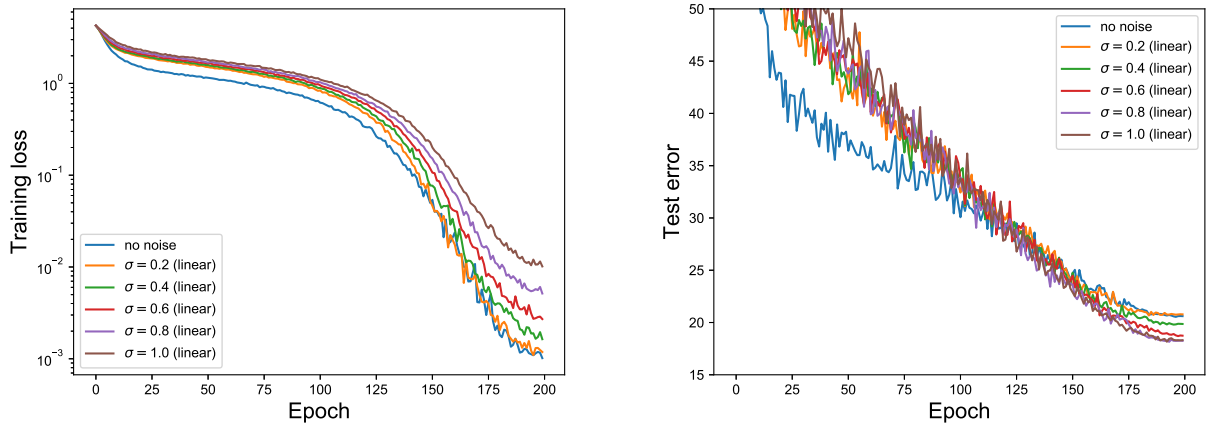
4.1.3 Small batch size training

As mentioned earlier, one major problem of BN is its performance degradation in the case of small mini-batch training because of inaccurate estimation of the batch statistics [11, 32]. We compare our method with BN under small batch sizes. In this set of experiments, we follow the linear learning rate scaling rule [5] to adapt to batch size changes. Specifically, we set the learning rate to $\eta B/128$ for the batch size of B, where η is the learning rate at the batch size of 128.

The results are shown in Table 2. Similar to reports in previous studies, the performance of BN severely degraded when training with small batch size, especially on CIFAR-100 for $B = 4$. In contrast, our method was robust to the batch size and stably achieved excellent performance even with the extremely small batch size of $B = 4$. Moreover,

Table 4: Experimental results on the CIFAR-10 and CIFAR-100 datasets with different levels of noise injection. WideResNet-28-10 with standard data augmentation is used. The best and second-best results are denoted in bold and italics, respectively.

	no noise	constant				linear				
		$\sigma = 0.1$	$\sigma = 0.2$	$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.2$	$\sigma = 0.4$	$\sigma = 0.6$	$\sigma = 0.8$	$\sigma = 1.0$
CIFAR-10	4.45	4.78	4.55	4.59	*4.25*	4.49	4.49	4.45	4.45	**4.19**
CIFAR-100	20.61	21.00	20.13	19.81	-	20.78	19.86	18.74	**18.25**	*18.31*



(a) Training loss

(b) Test error

Figure 2: Learning curves on the CIFAR-100 dataset with different levels of noise injection. WideResNet-28-10 with standard data augmentation is used.

our method sometimes performed better with smaller batch sizes. For example, on CIFAR-100, the error rate was 16.40 when $B = 128$, but it was reduced to 15.12 when $B = 4$. One possible reason would be a characteristic of SGD. That is, SGD with smaller batch sizes can find flatter minima and achieve better generalization performance [14]. Unlike BN, our method is not dependent on batch statistics and therefore enjoys this benefit of SGD for the case of small batch sizes without suffering from its disadvantage. This suggests another important advantage of our method that does not need to be combined with BN.

4.1.4 Ablation study

The three main components of the proposed method are centralization, spherization and noise injection. We conduct ablation experiments to investigate the effect of each of these components on the final performance.

Centralization and normalization. Table 3 shows the test errors on the CIFAR-10 and CIFAR-100 datasets. We found that using only spherization or only centralization can reduce the performance, while using both together can significantly improve the performance. Some existing WN methods have only considered either spherization or centralization. Our results suggest that it might be a source of the performance degradation of these methods.

Noise injection. We consider two noise injection strategies: constant noise injection (i.e., $\sigma_l = \sigma$) and the linearly increasing noise injection given in Eq. (8). The results are shown in Table 4. We found that the test accuracy improved over the baseline (no noise) by injecting the proper magnitude of noise. For both datasets, the linearly increasing noise injection performed better than the constant noise. In the range up to $\sigma = 1.0$, larger standard deviations tended to give higher accuracy. Figure 2(a) and (b) show the training loss and test error with respect to epochs on the CIFAR-100 dataset, respectively. Although applying noise injection increased the final training loss value, it also improved the test accuracy without a significant delay in learning. This indicates that our noise injection improves the generalization performance.

Table 5: Comparison of point cloud classification accuracy on the ModelNet40 dataset. The results other than ours are cited from [22].

Method	Mean class acc.	Overall acc.
Ours	89.0	91.4
GN	87.0	89.7
WS+GN	88.8	91.2
BN	89.3	91.7
WS+BN	89.6	92.0

Table 6: Comparison of error rates on the ImageNet dataset. ResNet-50 is used as the base network.

Method	Top-1 accuracy	Top-5 accuracy
Ours	75.15	92.37
Plain	73.76	91.15
GN	75.19	92.54
WS+GN	76.28	93.01
BN	75.70	92.81
WS+BN	76.24	92.87

4.2. Results on ModelNet40

To demonstrate the generality of our method, we perform experiments in a 3D object classification task. We apply our proposed method to point cloud classification on the ModelNet40 dataset [33], which contains 40 manually created object categories of CAD models. There are $12,311$ CAD models in total, split into $9,843$ for training and $2,468$ for testing.

4.2.1 Setup

We follow the same experimental settings as [31], i.e., $1,024$ points are uniformly sampled on mesh faces according to face area and then normalized into a unit sphere. In the training phase, we augment the point cloud on-the-fly by randomly rotating the object along the up-axis and jitter the position of each point. The network architecture and pre-processing for training are also the same as in [31]. In our proposed method, we replace all the BN layers in the same way as the previous image classification experiments. We set $\sigma = 0.4$ for our method.

4.2.2 Results

We show the mean class accuracy and overall accuracy for the proposed and existing methods in Table 5. Note that results other than ours are cited from [22]. Our method clearly outperformed GN and had comparable results to WS+GN and BN. This demonstrates that our proposed method can improve learning efficiency on the point cloud classification task.

4.3. Results on ImageNet

Finally, we evaluate our method on the ImageNet 2012 classification dataset [23], which consists of 1.28 million training images and $50,000$ validation images from $1,000$ classes.

4.3.1 Setup

In our evaluation, the validation set is used as the test set. We trained each method with the ResNet-50 [8] architecture. We follow the simple data augmentation in [6, 29]. More specifically, input images of size 224×224 are randomly cropped from a resized image using scale and aspect ratio augmentation. The networks are optimized using SGD with a momentum of 0.9 and the batch size of 256. We start from a learning rate of 0.1 and divide it by 10 at 30, 60, and 90 epochs.

4.3.2 Results

We compared our method with BN, GN, WS+BN, WS+GN, and plain ResNet which is a ResNet whose BN layers have been removed. Top-1 and top-5 error rates on the validation set are shown in Table 6. The results other than ours and for plain CNN are quoted from [22]. Our method significantly improved the top-1 accuracy from 73.76% to 75.15% and achieved a comparable performance to GN.

5. Conclusion

In this paper, we proposed a weight normalization approach that can be used on its own to provide a classification accuracy competitive to BN. The key concepts of our approach are that it 1) makes the norm of the weights constant and the mean of the weights zero and 2) mimics stochastic perturbations by injecting noise. Unlike existing WN methods based on reparametrization, which often suffer from the gradient explosion problem, our method directly optimizes the weights by solving a constrained optimization problem and thus can avoid that harmful effect. Our algorithm can efficiently solve the proposed optimization problem. We evaluated our proposed method in various classification tasks using three popular datasets and found that it was highly competitive with, or even better than, the state-of-the-art normalization methods without relying on any activation normalization methods.

References

[1] D. Arpit, Y. Zhou, B. Kota, and V. Govindaraju. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. In *Proc. ICML*, 2016.

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

[3] Nils Bjorck, Carla P. Gomes, Bart Selman, and Kilian Q. Weinberger. Understanding batch normalization. In *Proc. NeurIPS*, 2018.

[4] Terrance DeVries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. In *arXiv preprint arXiv:1708.04552*, 2017.

[5] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. In *arXiv preprint arXiv:1706.02677*, 2017.

[6] Sam Gross and Michael Wilber. Training and investigating residual nets. `https://github.com/facebook/fb.resnet.torch`, 2016.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. ICCV*, 2015.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. CVPR*, 2016.

[9] Lei Huang, Xianglong Liu, Bo Lang, and Bo Li. Projection based weight normalization for deep neural networks. In *arXiv preprint, arXiv:1710.02338*, 2017.

[10] Lei Huang, Xianglong Liu, Yang Liu, Bo Lang, and Dacheng Tao. Centered weight normalization in accelerating training of deep neural networks. In *Proc. ICCV*, 2017.

[11] Sergey Ioffe. Batch Renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Proc. NeurIPS*, 2017.

[12] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, 2015.

[13] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Proc. ICCV.*, 2009.

[14] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *Proc. ICLR*, 2017.

[15] Durk P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Proc. NeurIPS*, 2018.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Proc. NeurIPS*, 2012.

[17] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast AutoAugment. In *Proc. NeurIPS*, 2019.

[18] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *Proc. ICLR*, 2017.

[19] Ping Luo, Jiamin Ren, Zhanglin Peng, Ruimao Zhang, and Jingyu Li. Differentiable learning-to-normalize via switchable normalization. In *Proc. ICLR*, 2019.

[20] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *Proc. ICLR*, 2018.

[21] Hyeonseob Nam and Hyo-Eun Kim. Batch-instance normalization for adaptively style-invariant neural networks. In *NeulIPS*, pages 2558–2567, 2018.

[22] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Weight standardization. In *arXiv preprint arXiv:1903.10520*, 2019.

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015.

[24] Tim Salimans and Durk P. Kingma. Weight Normalization: A simple reparameterization to accelerate training of deep neural networks. In *Proc. NeurIPS*, 2016.

[25] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Proc. NeurIPS*, 2018.

[26] Cecilia Summers and Michael J Dinneen. Four things everyone should know to improve batch normalization. *arXiv preprint arXiv:1906.03548*, 2019.

[27] Cecilia Summers and Michael J. Dinneen. Four things everyone should know to improve batch normalization. In *Proc. ICLR*, 2020.

[28] Jiacheng Sun, Xiangyong Cao, Hanwen Liang, Weiran Huang, Zewei Chen, and Zhenguo Li. New interpretations of normalization methods in deep learning. In *Proc. AAAI*, 2020.

[29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. CVPR*, 2015.

[30] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance Normalization: The missing ingredient for fast stylization. In *arXiv preprint arXiv:1607.08022*, 2016.

[31] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Trans. on Graphics (TOG)*, 38(5):1–12, 2019.

[32] Yuxin Wu and Kaiming He. Group normalization. In *Proc. ECCV*, 2018.

[33] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *Proc. CVPR*, 2015.

[34] Hongwei Yong, Jianqiang Huang, Xiansheng Hua, and Lei Zhang. Gradient centralization: A new optimization technique for deep neural networks. In *Proc. ECCV*, 2020.

[35] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proc. BMVC*, 2016.

[36] Chaoning Zhang, In-So Kweon, Rameau Francois, and Gyumin Shim. Revisiting residual networks with nonlinear shortcuts. In *BMVC*, 2019.