Iterative Normalization: Beyond Standardization towards Efficient Whitening

Lei Huang Yi Zhou Fan Zhu Li Liu Ling Shao
Inception Institute of Artificial Intelligence (IIAI), Abu Dhabi, UAE
{lei.huang, yi.zhou, fan.zhu, li.liu, ling.shao} @inceptioniai.org

Abstract

Batch Normalization (BN) is ubiquitously employed for accelerating neural network training and improving the generalization capability by performing standardization within mini-batches. Decorrelated Batch Normalization (DBN) further boosts the above effectiveness by whitening. However, DBN relies heavily on either a large batch size, or eigen-decomposition that suffers from poor efficiency on GPUs. We propose Iterative Normalization (IterNorm), which employs Newton's iterations for much more efficient whitening, while simultaneously avoiding the eigen-decomposition. Furthermore, we develop a comprehensive study to show IterNorm has better trade-off between optimization and generalization, with theoretical and experimental support. To this end, we exclusively introduce Stochastic Normalization Disturbance (SND), which measures the inherent stochastic uncertainty of samples when applied to normalization operations. With the support of SND, we provide natural explanations to several phenomena from the perspective of optimization, e.g., why group-wise whitening of DBN generally outperforms full-whitening and why the accuracy of BN degenerates with reduced batch sizes. We demonstrate the consistently improved performance of IterNorm with extensive experiments on CIFAR-10 and ImageNet over BN and DBN.

1. Introduction

Centering, scaling and decorrelating the input data is known as data whitening, which has demonstrated enormous success in speeding up training [25]. Batch Normalization (BN) [19] extends the operations from the input layer to centering and scaling activations of each intermediate layer within a mini-batch so that each neuron has a zero mean and a unit variance (Figure 1 (a)). BN has been extensively used in various network architectures [11, 43, 12, 50, 42, 14] for its benefits in improving both the optimization efficiency [19, 9, 21, 5, 37] and generalization capability [19, 3, 5, 48]. However, instead of performing whitening, BN is only capable of performing standardization, which centers and scales the activations but does not decorrelate them [19]. On the

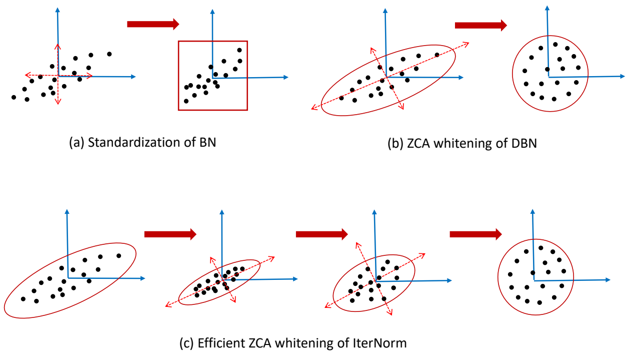

(a) Standardization of BN
(b) ZCA whitening of DBN
(c) Efficient ZCA whitening of IterNorm

Figure 1. Illustrations of multiple normalization methods on centered data. (a) BN [19] performs standardization by stretching/squeezing the data along the axes, such that each dimension has a unit variance; (b) DBN performs ZCA whitening by stretching/squeezing the data along the eigenvectors, such that the covariance matrix is identical. (c) IterNorm performs efficient ZCA whitening by progressively adjusting the data along the eigenvectors without eigen-decomposition.

other hand, previous works suggest that further decorrelating the activations is beneficial to both the optimization [9, 29] and generalization [6, 49]. To the end of improving BN with whitening, Decorrelated Batch Normalization (DBN) [17] is proposed to whiten the activations of each layer within a mini-batch, such that the output of each layer has an isometric diagonal covariance matrix (Figure 1 (b)). DBN improves over BN in regards to both training efficiency and generalization capability, but it relies heavily on a large batch size and eigen-decompositions or singular value decomposition (SVD), which suffers from poor efficiency on GPUs.

In order to address these issues, we propose Iterative Normalization (IterNorm) to further enhance BN with more efficient whitening. IterNorm avoids eigen-decomposition or SVD by employing Newton's iteration for approximating the whitening matrix. Thus, the capacity of GPUs can be effectively exploited. Eigenvalues of the covariance matrix are normalized prior to the iterations with guaranteed convergence condition of Newton's iteration. As illustrated in (Figure 1 (c)), IterNorm stretches the dimensions along the eigenvectors progressively, so that the associated eigenvalues converge to 1 after normalization. One desirable property is that the convergence speed of IterNorm along the eigenvectors is proportional to the associated eigenvalues [4]. This means the dimensions that correspond to small/zero (when a small batch size is applied) eigenvalues can be largely ignored, given a fixed number of iterations. As a consequence, the sensitivity of IterNorm against batch size can be significantly reduced.

When the data batch is undersized, it is known that the performance of both whitening and standardization on the test data can be significantly degraded [18, 48]. However, beyond our expectation, we observe that the performance on the training set also significantly degenerates under the same condition. We further observe such a phenomenon is caused by the stochasticity introduced by the mini-batch based normalization [44, 39]. To allow a more comprehensive understanding and evaluation about the stochasticity, we introduce Stochastic Normalization Disturbance (SND), which is discussed in Section 4. With the support of SND, we provide a thorough analysis regarding the performance of normalization methods, with respect to the batch size and feature dimensions, and show that IterNorm has better trade-off between optimization and generalization. Experiments on CIFAR-10 [22] and ILSVRC-2012 [8] demonstrate the consistent improvements of IterNorm over BN and DBN.

2. Related Work

Normalized activations [38, 33, 31, 46] have long been known to benefit neural networks training. Some research methodologies attempt to normalize activations by viewing the population statistics as parameters and estimating them directly during training [31, 46, 9]. Some of these methods include activations centering in Restricted Boltzmann Machine [31]/feed-forward neural networks [46] and activations whitening [9, 29]. This type of normalization may suffer from instability (such as divergence or gradient explosion) due to 1) inaccurate approximation to the population statistics with local data samples [46, 19, 18, 17] and 2) the internal-covariant shift problem [19].

Ioffe et al., [19] propose to perform normalization as a function over mini-batch data and back-propagate through the transformation. Multiple standardization options have been discovered for normalizing mini-batch data, including the L2 standardization [19], the L1-standardization [47, 13] and the $L\infty$-standardization [13]. One critical issue with these methods, however, is that it normally requires a reasonable batch size for estimating the mean and variance. In order to address such an issue, a significant number of standardization approaches are proposed [3, 48, 34, 30, 18, 27, 45, 24, 7]. Our work develops in an orthogonal direction to these approaches, and aims at improving BN with decorrelated activations.

Beyond standardization, Huang *et al.* [17] propose DBN, which uses ZCA-whitening by eigen-decomposition and back-propagates the transformation. Our approach aims at a much more efficient approximation of the ZCA-whitening matrix in DBN, and suggests that approximating whitening is more effective based on the analysis shown in Section 4.

Our approach is also related to works that normalize the network weights (e.g., either through re-parameterization [36, 16, 15] or weight regularization [23, 32, 35]), and that specially design either scaling coefficients & bias values [1] or nonlinear function [20], to normalize activation implicitly [39]. IterNorm differs from these work in that it is a data dependent normalization, while these normalization approaches are independent of the data.

Newton's iteration is also employed in several other deep neural networks. These methods focus on constructing bilinear [28] or second-order pooling [26] by constraining the power of the covariance matrix and are limited to producing fully-connected activations, while our work provides a generic module that can be ubiquitously built in various neural network frameworks. Besides, our method computes the square root inverse of the covariance matrix, instead of calculating the square root of the covariance matrix [28, 26].

3. Iterative Normalization

Let $\mathbf{X} \in \mathbf{R}^{d\times m}$ be a data matrix denoting the mini-batch input of size m in certain layer. BN [19] works by standardizing the activations over the mini-batch input:

$$\widehat{\mathbf{X}} = \phi_{Std}(\mathbf{X}) = \Lambda_{std}^{-\frac{1}{2}}(\mathbf{X} - \mu \cdot \mathbf{1}^T), \tag{1}$$

where $\mu = \frac{1}{m}\mathbf{X} \cdot \mathbf{1}$ is the mean of $\mathbf{X}$, $\Lambda_{std} = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_d^2) + \epsilon\mathbf{I}$, σ_i^2 is the dimension-wise variance corresponding to the *i*-th dimension, $\mathbf{1}$ is a column vector of all ones, and $\epsilon > 0$ is a small number to prevent numerical instability. Intuitively, standardization ensures that the normalized output gives equal importance to each dimension by multiplying the scaling matrix $\Lambda_{std}^{-\frac{1}{2}}$ (Figure 1 (a)).

DBN [17] further uses ZCA whitening to produce the whitened output as[1]:

$$\phi_{ZCA}(\mathbf{X}) = \mathbf{D}\Lambda^{-\frac{1}{2}}\mathbf{D}^T(\mathbf{X} - \mu \cdot \mathbf{1}^T), \tag{2}$$

where $\Lambda = \mathrm{diag}(\sigma_1, \ldots, \sigma_d)$ and $\mathbf{D} = [\mathbf{d}_1, ..., \mathbf{d}_d]$ are the eigenvalues and associated eigenvectors of Σ, *i.e.* $\Sigma = \mathbf{D}\Lambda\mathbf{D}^T$. $\Sigma = \frac{1}{m}(\mathbf{X} - \mu \cdot \mathbf{1}^T)(\mathbf{X} - \mu \cdot \mathbf{1}^T)^T + \epsilon\mathbf{I}$ is the covariance matrix of the centered input. ZCA whitening works by stretching or squeezing the dimensions along the eigenvectors such that the associated eigenvalues to be 1 (Figure 1 (b)). Whitening the activation ensures that all dimensions along the eigenvectors have equal importance in the subsequent linear layer.

One crucial problem of ZCA whitening is that calculating the whitening matrix requires eigen-decomposition or

[1]DBN and BN both use learnable dimension-wise scale and shift parameters to recover the possible loss of representation capability.

Algorithm 1 Whitening activations with Newton's iteration.

```
1:  Input: mini-batch inputs $\mathbf{X} \in \mathbb{R}^{d\times m}$.
2:  Hyperparameters: $\epsilon$, iteration number $T$.
3:  Output: the ZCA-whitened activations $\widehat{\mathbf{X}}$.
4:  calculate mini-batch mean: $\mu = \frac{1}{m}\mathbf{X}\cdot\mathbf{1}$.
5:  calculate centered activation: $\mathbf{X}_C = \mathbf{X} - \mu\cdot\mathbf{1}^T$.
6:  calculate covariance matrix: $\Sigma = \frac{1}{m}\mathbf{X}_C\mathbf{X}_C^T + \epsilon\mathbf{I}$.
7:  calculate trace-normalized covariance matrix $\Sigma_N$ by Eqn .4.
8:  $\mathbf{P}_0 = \mathbf{I}$.
9:  for $k = 1$ to $T$ do
10:     $\mathbf{P}_k = \frac{1}{2}(3\mathbf{P}_{k-1} - \mathbf{P}_{k-1}^3\Sigma_N)$
11: end for
12: calculate whitening matrix: $\Sigma^{-\frac{1}{2}} = \mathbf{P}_T/\sqrt{tr(\Sigma)}$.
13: calculate whitened output: $\widehat{\mathbf{X}} = \Sigma^{-\frac{1}{2}}\mathbf{X}_C$.
```

SVD, as shown in Eqn. 2, which heavily constrains its practical applications. We observe that Eqn. 2 can be viewed as the square root inverse of the covariance matrix denoted by $\Sigma^{-\frac{1}{2}}$, which multiplies the centered input. The square root inverse of one specific matrix can be calculated using Newton's iteration methods [4], which avoids executing eigen-decomposition or SVD.

3.1. Computing $\Sigma^{-\frac{1}{2}}$ by Newton's Iteration

Given the square matrix $\mathbf{A}$, Newton's method calculates $\mathbf{A}^{-\frac{1}{2}}$ by the following iterations [4]:

$$\begin{cases}\mathbf{P}_0 = \mathbf{I} \\ \mathbf{P}_k = \frac{1}{2}(3\mathbf{P}_{k-1} - \mathbf{P}_{k-1}^3\mathbf{A}), \quad k = 1,2,...,T,\end{cases} \tag{3}$$

where T is the iteration number. $\mathbf{P}_k$ will be converged to $\mathbf{A}^{-\frac{1}{2}}$ under the condition $\|\mathbf{A}-\mathbf{I}\|_2 < 1$.

In terms of applying Newton's methods to calculate the inverse square root of the covariance matrix $\Sigma^{-\frac{1}{2}}$, one crucial problem is Σ cannot be guaranteed to satisfy the convergence condition $\|\Sigma - \mathbf{I}\|_2 < 1$. That is because Σ is calculated over mini-batch samples and thus varies during training. If the convergence condition cannot be perfectly satisfied, the training can be highly instable [4, 26]. To address this issue, we observe that one sufficient condition for convergence is to ensure the eigenvalues of the covariance matrix are less than 1. We thus propose to construct a transformation $\Sigma_N = F(\Sigma)$ such that $\|\Sigma_N\|_2 < 1$, and ensure the transformation is differentiable such that the gradients can back-propagate through this transformation. One feasible transformation is to normalize the eigenvalue as follows:

$$\Sigma_N = \Sigma/tr(\Sigma), \tag{4}$$

where $tr(\Sigma)$ indicates the trace of Σ. Note that Σ_N is also a semi-definite matrix and thus all of its eigenvalues are greater than or equal to 0. Besides, Σ_N has the property that the sum of its eigenvalues is 1. Therefore, Σ_N can surely satisfy the convergence condition. We can thus calculate the inverse square root $\Sigma_N^{-\frac{1}{2}}$ by Newton's method as Eqn. 3. Given $\Sigma_N^{-\frac{1}{2}}$, we can compute $\Sigma^{-\frac{1}{2}}$ based on Eqn. 4, as follows:

$$\Sigma^{-\frac{1}{2}} = \Sigma_N^{-\frac{1}{2}}/\sqrt{tr(\Sigma)}. \tag{5}$$

Given $\Sigma^{-\frac{1}{2}}$, it's easy to whiten the activations by multiplying $\Sigma^{-\frac{1}{2}}$ with the centered inputs. In summary, Algorithm 1 describes our proposed methods for whitening the activations in neural networks.

Our method first normalizes the eigenvalues of the covariance matrix, such that the convergence condition of Newton's iteration is satisfied. We then progressively stretch the dimensions along the eigenvectors, such that the final associate eigenvalues are all "1", as shown in Figure 1 (c). Note that the speed of convergence of the eigenvectors is proportional to the associated eigenvalues [4]. That is, the larger the eigenvalue is, the faster its associated dimension along the eigenvectors converges. Such a mechanism is a remarkable property to control the extent of whitening, which is essential for the success of whitening activations, as pointed out in [17], and will be further discussed in Section 4.

3.2. Back-propagation

As pointed out by [19, 17], viewing standardization or whitening as functions over the mini-batch data and back-propagating through the normalized transformation are essential for stabilizing training. Here, we derive the back-propagation pass of IterNorm. Denoting L as the loss function, the key is to calculate $\frac{\partial L}{\partial \Sigma}$, given $\frac{\partial L}{\partial \Sigma^{-1/2}}$. Let's denote $\mathbf{P}_T = \Sigma_N^{-\frac{1}{2}}$, where T is the iteration number. Based on the chain rules, we have:

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{P}_T} &= \frac{1}{\sqrt{tr(\Sigma)}}\frac{\partial L}{\partial \Sigma^{-\frac{1}{2}}} \\ \frac{\partial L}{\partial \Sigma_N} &= -\frac{1}{2}\sum_{k=1}^{T}(\mathbf{P}_{k-1}^3)^T\frac{\partial L}{\partial \mathbf{P}_k} \\ \frac{\partial L}{\partial \Sigma} &= \frac{1}{tr(\Sigma)}\frac{\partial L}{\partial \Sigma_N} - \frac{1}{(tr(\Sigma))^2}tr(\frac{\partial L}{\partial \Sigma_N}^T\Sigma)\mathbf{I} \\ &- \frac{1}{2(tr(\Sigma))^{3/2}}tr((\frac{\partial L}{\partial \Sigma^{-1/2}})^T\mathbf{P}_T)\mathbf{I},\end{aligned} \tag{6}$$

where $\frac{\partial L}{\partial \mathbf{P}_k}$ can be calculated by following iterations:

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{P}_{k-1}} &= \frac{3}{2}\frac{\partial L}{\partial \mathbf{P}_k} - \frac{1}{2}\frac{\partial L}{\partial \mathbf{P}_k}(\mathbf{P}_{k-1}^2\Sigma_N)^T - \frac{1}{2}(\mathbf{P}_{k-1}^2)^T\frac{\partial L}{\partial \mathbf{P}_k}\Sigma_N^T \\ &- \frac{1}{2}(\mathbf{P}_{k-1})^T\frac{\partial L}{\partial \mathbf{P}_k}(\mathbf{P}_{k-1}\Sigma_N)^T, \quad k = T,...,1.\end{aligned} \tag{7}$$

Note that in Eqn. 4 and 5, $tr(\Sigma)$ is a function for mini-batch examples and is needed to back-propagate through it to stabilize the training. Algorithm 2 summarizes the back-propagation pass of our proposed IterNorm. More details of back-propagation derivations are shown in *supplementary materials*.

Algorithm 2 The respective backward pass of Algorithm 1.

```
1: Input: mini-batch gradients respect to whitened activations: $\frac{\partial L}{\partial \hat{\mathbf{X}}}$. auxiliary data from respective forward pass: (1) $\mathbf{X}_C$; (2) $\Sigma^{-\frac{1}{2}}$; (3) $\{\mathbf{P}_k\}$.
2: Output: the gradients with respect to the inputs: $\frac{\partial L}{\partial \mathbf{X}}$.
3: calculate the gradients with respect to $\Sigma^{-\frac{1}{2}}$: $\frac{\partial L}{\partial \Sigma^{-\frac{1}{2}}} = \frac{\partial L}{\partial \hat{\mathbf{X}}} \mathbf{X}_C^T$.
4: calculate $\frac{\partial L}{\partial \Sigma}$ based on Eqn. 6 and 7.
5: calculate: $\mathbf{f} = \frac{1}{m} \frac{\partial L}{\partial \hat{\mathbf{X}}} \cdot \mathbf{1}$.
6: calculate: $\frac{\partial L}{\partial \mathbf{X}} = \Sigma^{-\frac{1}{2}} (\frac{\partial L}{\partial \hat{\mathbf{X}}} - \mathbf{f} \cdot \mathbf{1}^T) + \frac{1}{m} (\frac{\partial L}{\partial \Sigma} + \frac{\partial L}{\partial \Sigma}^T) \mathbf{X}_C$.
```

3.3. Training and Inference

Like the previous normalizing activation methods [19, 3, 17, 48], our IterNorm can be used as a module and inserted into a network extensively. Since IterNorm is also a method for mini-batch data, we use the running average to calculate the population mean $\hat{\mu}$ and whitening matrix $\widehat{\Sigma}^{-\frac{1}{2}}$, which is used during inference. Specifically, during training, we initialize $\hat{\mu}$ as $\mathbf{0}$ and $\widehat{\Sigma}^{-\frac{1}{2}}$ as $\mathbf{I}$ and update them as follows:

$$\begin{aligned} \hat{\mu} &= (1-\lambda)\, \hat{\mu} + \lambda\, \mu \\ \widehat{\Sigma}^{-\frac{1}{2}} &= (1-\lambda) \widehat{\Sigma}^{-\frac{1}{2}} + \lambda \Sigma^{-\frac{1}{2}}, \end{aligned} \tag{8}$$

where μ and $\Sigma^{-\frac{1}{2}}$ are the mean and whitening matrix calculated within each mini-batch during training, and λ is the momentum of running average.

Additionally, we also use the extra learnable parameters γ and β, as in previous normalization methods [19, 3, 17, 48], since normalizing the activations constrains the model's capacity for representation. Such a process has been shown to be effective [19, 3, 17, 48].

Convolutional Layer For a CNN, the input is $\mathbf{X}_C \in \mathbb{R}^{h \times w \times d \times m}$, where h and w indicate the height and width of the feature maps, and d and m are the number of feature maps and examples, respectively. Following [19], we view each spatial position of the feature map as a sample. We thus unroll $\mathbf{X}_C$ as $\mathbf{X} \in \mathbb{R}^{d \times (mhw)}$ with $m \times h \times w$ examples and d feature maps. The whitening operation is performed over the unrolled $\mathbf{X}$.

Computational Cost The main computation of our IterNorm includes calculating the covariance matrix, the iteration operation and the whitened output. The computational costs of the first and the third operation are equivalent to the 1×1 convolution. The second operation's computational cost is Td^3. Our method is comparable to the convolution operation. To be specific, given the internal activation $\mathbf{X}_C \in \mathbb{R}^{h \times w \times d \times m}$, the 3×3 convolution with the same input and output feature maps costs $9hwmd^2$, while our IterNorm costs $2hwmd^2 + Td^3$. The relative cost of IterNorm for 3×3 convolution is $2/9 + Td/mhw$. Further, we can use group-wise whitening, as introduced in [17] to improve the efficiency when the dimension d is large. We also compare the wall-clock time of IterNorm, DBN [17] and 3×3 convolution in *supplementary materials*.

During inference, IterNorm can be viewed as a 1×1 convolution and merged to adjacent convolutions. Therefore, IterNorm does not introduce any extra costs in memory or computation during inference.

4. Stochasticity of Normalization

Mini-batch based normalization methods are sensitive to the batch size [18, 17, 48]. As described in [17], fully whitening the activation may suffer from degenerate performance while the number of data in a mini-batch is not sufficient. They [17] thus propose to use group-wise whitening [17]. Furthermore, standardization also suffers from degenerated performance under the scenario of micro-batch [45]. These works argue that undersized data batch makes the estimated population statistics highly noisy, which results in a degenerating performance during inference [3, 18, 48].

In this section, we will provide a more thorough analysis regarding the performance of normalization methods, with respect to the batch size and feature dimensions. We show that normalization (standardization or whitening) with undersized data batch not only suffers from degenerate performance during inference, but also encounter the difficulty in optimization during training. This is caused by the Stochastic Normalization Disturbance (SND), which we will describe.

4.1. Stochastic Normalization Disturbance

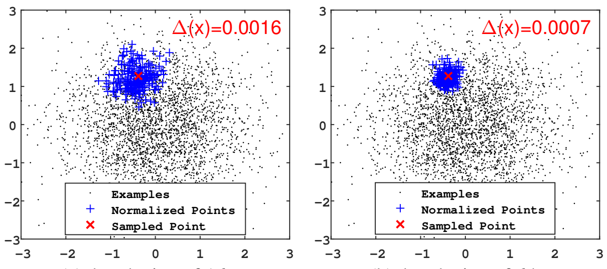
Given a sample $\mathbf{x} \in \mathbb{R}^d$ from a distribution P_{χ}, we take a sample set $\mathbf{X}^B = \{\mathbf{x}_1, ..., \mathbf{x}_B, \mathbf{x}_i \sim P_{\chi}\}$ with a size of B. We denote the normalization operation as $F(\cdot)$ and the normalized output as $\hat{\mathbf{x}} = F(\mathbf{X}^B; \mathbf{x})$. For a certain $\mathbf{x}$, $\mathbf{X}^B$ can be viewed as a random variable [2, 44]. $\hat{\mathbf{x}}$ is thus a random variable which shows the stochasticity. It's interesting to explore the statistical momentum of $\mathbf{x}$ to measure the magnitude of the stochasticity. Here we define the *Stochastic Normalization Disturbance* (SND) for the sample $\mathbf{x}$ over the normalization $F(\cdot)$ as:

$$\mathbf{\Delta}_F(\mathbf{x}) = \mathbf{E}_{\mathbf{X^B}}(\|\hat{\mathbf{x}} - \mathbf{E}_{\mathbf{X}^B}(\hat{\mathbf{x}})\|_2). \tag{9}$$

It's difficult to accurately compute this momentum if no further assumptions are made over the random variable $\mathbf{X}^B$, however, we can explore its empirical estimation over the sampled sets as follows:

$$\widehat{\mathbf{\Delta}}_F(\mathbf{x}) = \frac{1}{s} \sum_{i=1}^{s} \|F(\mathbf{X}_i^B; \mathbf{x}) - \frac{1}{s} \sum_{j=1}^{s} F(\mathbf{X}_j^B; \mathbf{x})\|, \tag{10}$$

where s denotes the time of sampling. Figure 2 gives the illustration of sample $\mathbf{x}$'s SND with respect to the operation of BN. We can find that SND is closely related to the batch size. When batch size is large, the given sample $\mathbf{x}$ has a small value of SND and the transformed outputs have a compact



(a) batch size of 16 (b) batch size of 64

Figure 2. Illustration of SND with different batch sizes. We sample 3000 examples (black points) from Gaussian distribution. We show a given example $\mathbf{x}$ (red cross) and its BN outputs (blue plus sign), when normalized over different sample sets $\mathbf{X}^B$. (a) and (b) show the results with batch sizes B of 16 and 64, respectively.

distribution. As a consequence, the stochastic uncertainty $\mathbf{x}$ can be low.

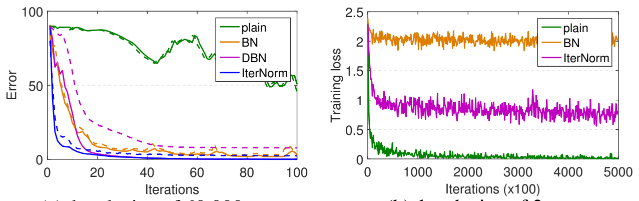
SND can be used to evaluate the stochasticity of a sample after the normalization operation, which works like the dropout rate [41]. We can further define the normalization operation $F(\cdot)$'s SND as: $\mathbf{\Delta}_F = \mathbf{E}_{\mathbf{x}}(\mathbf{\Delta}(\mathbf{x}))$ and it's empirical estimation as $\widehat{\mathbf{\Delta}}_F = \frac{1}{N}\sum_{i=1}^{N}\widehat{\mathbf{\Delta}}(\mathbf{x})$ where N is the number of sampled examples. $\mathbf{\Delta}_F$ describes the magnitudes of stochasticity for corresponding normalization operations.

Exploring the exact statistic behavior of SND is difficult and out of the scope of this paper. We can, however, explore the relationship of SND related to the batch size and feature dimension. We find that our defined SND gives a reasonable explanation to why we should control the extent of whitening and why mini-batch based normalizations have a degenerate performance when given a small batch size.

4.2. Controlling the Extent of Whitening

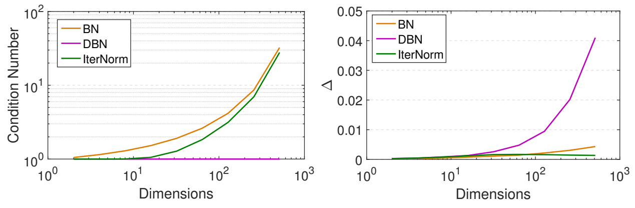
We start with experiments on multi-layer perceptron (MLP) over MNIST dataset, by using the full batch gradient (batch size =60,000), as shown in Figure 3 (a). We find that all normalization methods significantly improve the performance. One interesting observation is that full-whitening the activations with such a large batch size still underperforms the approximate-whitening of IterNorm, in terms of training efficiency. Intuitively, full-whitening the activations may lead to amplifying the dimension with small eigenvalues, which may correspond to the noise. Exaggerating this noise may be harmful to learning, especially lowering down the generalization capability as shown in Figure 3 (a) that DBN has diminished test performance. We provide further analysis based on SND, along with the conditioning analysis. It has been shown that improved conditioning can accelerate training [25, 9], while increased stochasticity can slow down training but likely to improve generalization [41] .

We experimentally explore the consequent effects of improved conditioning [25] with SND through BN (standardization), DBN (full-whitening) and IterNorm (approximate-



(a) batch size of 60,000 (b) batch size of 2

Figure 3. Ablation study in training a 4-layer MLP on MNIST. The number of neurons in each hidden layer is 100. We use full batch gradient and report the best results with respect to the training loss among learning rates=$\{0.2, 0.5, 1, 2, 5\}$, and stochastic gradient with batch size of 2 among learning rates= $\{0.005, 0.01, 0.02, 0.05, 0.1\}$. (a) shows the training (solid lines) and test (dashed lines) errors with respect to the iterations, and (b) shows the training loss. 'plain' is referred to as the network without normalization.



(a) Condition number (b) SND

Figure 4. Comparison of different normalization operations in condition number of covariance matrix of normalized output (a) and SND (b). We sample 60,000 examples from Gaussian distribution and choose a batch size of 1024, and observe the results with respect to the dimensions from 2^1 to 2^9, averaged over 10 times.

whitening with 5 iterations). We calculate the condition number of covariance matrix of normalized output, and the SND for different normalization methods (as shown in Figure 4). We find that DBN has the best conditioning with an exact condition number as 1, however it significantly enlarges SND, especially in a high-dimensional space. Therefore, full-whitening can not consistently improve the training efficiency, even for the highly improved conditioning, which is balanced out by the larger SND. Such an observation also explains why group-based whitening [17] (by reducing the number of dimensions that will be whitened) works better from the training perspective.

IterNorm has consistently improved conditioning over different dimensions compared to BN. Interestingly, IterNorm has a reduced SND in a high-dimensional space, since it can adaptively normalize the dimensions along different eigenvalues based on the convergence theory of Newton's iteration [4]. Therefore, IterNorm possesses a better trade-off between the improved conditioning and SND, which naturally illustrates IterNorm can be more efficiently trained. We also provide the results of IterNorm when applying different iteration numbers in *supplementary materials*.

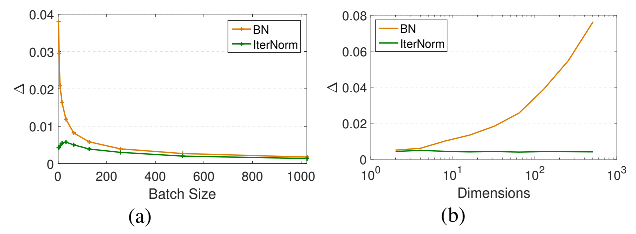


Figure 5. Illustration of the micro-batch problem of BN from the perspective of SND. (a) shows the SND with respect to batch sizes under the dimension of 128. (b) shows the SND with respect to dimensions under the batch size of 2.

4.3. Micro-batch Problem of BN

BN suffers from degenerate test performance if the batch data is undersized [48]. We also show that BN suffers from the optimization difficulty with a small batch size. We show the experimental results on MNIST dataset with a batch size of 2 in Figure 3 (b). We find that BN can hardly learn and produces random results, while the naive network without normalization learns well. Such an observation clearly shows that BN suffers from more difficulties in training with undersized data batch.

For an in-depth investigation, we sample the data from the dimension of 128 (Figure 5 (a)), and find that BN has a significantly increased SND. With increasing batch sizes, the SND of BN can be gradually reduced. Meanwhile, reduced SND leads to more stable training. When we fix the batch size to 2 and vary the dimension, (as shown in Figure 5 (b)), we observe that the SND of BN can be reduced with a low dimension. On the contrary, the SND of BN can be increased in a high-dimensional space. Thus, it can be explained why BN suffers from the difficulty during with a small batch, and why group-based normalization [48] (by reducing the dimension and adding the examples to be standardized implicitly) alleviates the problem.

Compared to BN, IterNorm is much less sensitive to a small batch size in producing SND. Besides, the SND of IterNorm is more stable, even with a significantly increased dimension. Such characteristics of IterNorm are mainly attributed to its adaptive mechanism in normalization, that it stretches the dimensions along large eigenvalues and correspondingly ignores small eigenvalues, given a fixed number of iterations [4].

5. Experiments

We evaluate IterNorm with CNNs on CIFAR datasets [22] to show that the better optimization efficiency and generalization capability, compared to BN [19] and DBN [17]. Furthermore, IterNorm with residual networks will be applied to show the performance improvement on CIFAR-10 and ImageNet [8] classification tasks. The code to reproduce the experiments is available at https://github.com/huangleiBuaa/IterNorm.

5.1. Sensitivity Analysis

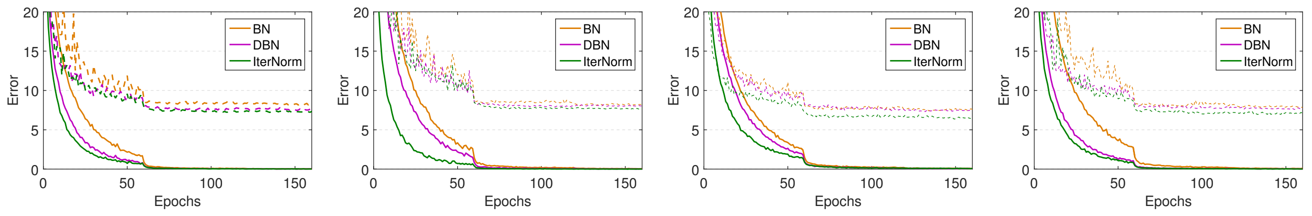
We analyze the proposed methods on CNN architectures over the CIFAR-10 dataset [22], which contains 10 classes with $50k$ training examples and $10k$ test examples. The dataset contains 32×32 color images with 3 channels. We use the VGG networks [40] tailored for 32×32 inputs (16 convolution layers and 1 fully-connected layers), and the details of the networks are shown in *supplementary materials*.

The datasets are preprocessed with a mean-subtraction and variance-division. We also execute normal data augmentation operation, such as a random flip and random crop with padding, as described in [11].

Experimental Setup We use SGD with a batch size of 256 to optimize the model. We set the initial learning rate to 0.1, then divide the learning rate by 5 at 60 and 120 epochs, and finish the training at 160 epochs. All results are averaged over 3 runs. For DBN, we use a group size of 16 as recommend in [17], and we find that DBN is unstable for a group size of 32 or above, due to the fact that the eigen-decomposition operation cannot converge. The main reason is that the batch size is not sufficient for DBN to full-whiten the activation for each layer. For IterNorm, we don't use group-wise whitening in the experiments, unless otherwise stated.

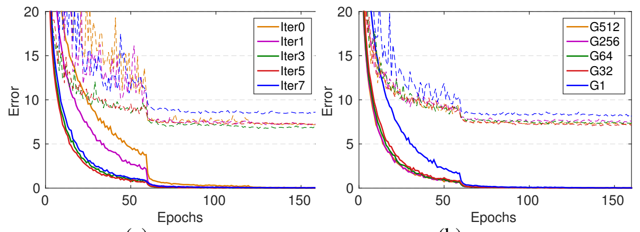
Effect of Iteration Number The iteration number T of our IterNorm controls the extent of whitening. Here we explore the effects of T on performance of IterNorm, for a range of $\{0, 1, 3, 5, 7\}$. Note that when $T = 0$, our method is reduced to normalizing the eigenvalues such that the sum of the eigenvalues is 1. Figure 7 (a) shows the results. We find that the smallest ($T = 0$) and the largest ($T = 7$) iteration number both have the worse performance in terms of training efficiency. Further, when $T = 7$, IterNorm has significantly worse test performance. These observations show that (1) whitening within an mini-batch can improve the optimization efficiency, since IterNorm progressively stretches out the data along the dimensions of the eigenvectors such that the corresponding eigenvalue towards 1, with increasing iteration T; (2) controlling the extent of whitening is essential for its success, since stretching out the dimensions along small eigenvalue may produce large SND as described in Section 4, which not only makes estimating the population statistics difficult — therefore causing higher test error — but also makes optimization difficult. Unless otherwise stated, we use an iteration number of 5 in subsequent experiments.

Effects of Group Size We also investigate the effects of group size. We vary the group size in $\{256, 64, 32, 1\}$, compared to the full-whitening operation of IterNorm (group size of 512). Note that our IterNorm with group size of 1, like DBN, is also reduced to Batch Normalization [19], which is ensured by Eqn. 4 and 5. The results are shown in Figure 7



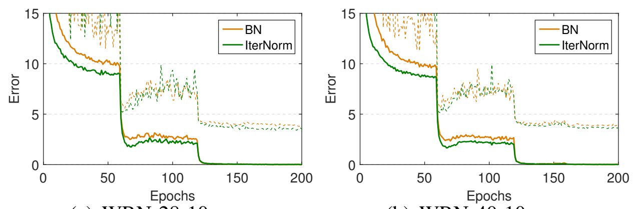
(a) basic configuration (b) batch size of 1024 (c) batch size of 16 (d) 10x larger learning rate

Figure 6. Comparison among BN, DBN and IterNorm on VGG over CIFAR-10 datasets. We report the training (solid lines) and test (dashed lines) error with respect to epochs.



(a) (b)

Figure 7. Ablation studies on VGG over CIFAR-10 datasets. We report the training (solid lines) and test (dashed lines) error curves. (a) shows the effects of different iteration number for IterNorm; (b) show the effects of different group size of IterNorm.



(a) WRN-28-10 (b) WRN-40-10

Figure 8. Comparison on Wide Residual Networks over CIFAR-10 datasets. The solid line indicates the training errors and the dashed line indicates the test errors. (a) shows the results on WRN-28-10 and (b) on WRN-40-10.

(b). We can find that our IterNorm, unlike DBN, is not sensitive to the large group size, not only in training, but also in testing. The main reason is that IterNorm gradually stretches out the data along the dimensions of eigenvectors such that the corresponding eigenvalue towards 1, in which the speed of convergence for each dimension is proportional to the associated eigenvalues [4]. Even though there are many small eigenvalue or zero in high-dimension space, IterNorm only stretches the dimension along the associate eigenvector a little, given small iteration T, which introduces few SND. In practice, we can use a smaller group size, which can reduce the computational costs. We recommend using a group size of 64, which is proposed in the experiments of Section 5.2 and 5.3 for IterNorm.

Comparison of Baselines We compare our IterNorm with $T = 5$ to BN and DBN. Under the basic configuration, we also experiment with other configurations, including (1) using a large batch size of 1024; (2) using a small batch size of 16; and (3) increasing the learning rate by 10 times and considering mini-batch based normalization is highly dependent on the batch size and their benefits comes from improved conditioning and therefore larger learning rate. All experimental setups are the same, except that we search a different learning rate in $\{0.4, 0.1, 0.0125\}$ for different batch sizes, based on the linear scaling rule [10]. Figure 6 shows the results.

We find that our proposed IterNorm converges the fastest with respect to the epochs, and generalizes the best, compared to BN and DBN. DBN also has better optimization and generalization capability than BN. Particularly, IterNorm reduces the absolute test error of BN by 0.79%, 0.53%, 1.11%, 0.75% for the four experiments above respectively, and DBN by 0.22, 0.37, 1.05, 0.58. The results demonstrate that our IterNorm outperforms BN and DBN in terms of optimization quality and generalization capability.

5.2. Results on CIFAR-10 with Wide Residual Networks

We apply our IterNorm to Wide Residual Network (WRN) [50] to improve the performance on CIFAR-10. Following the conventional description in [50], we use the abbreviation WRN-d-k to indicate a WRN with depth *d* and width *k*. We adopt the publicly available Torch implementation[2] and follow the same setup as in [50]. We apply IterNorm to WRN-28-10 and WRN-40-10 by replacing all the BN modules with our IterNorm. Figure 8 gives the training and testing errors with respect to the training epochs. We clearly find that the wide residual network with our proposed IterNorm improves the original one with BN, in terms of optimization efficiency and generalization capability. Table 1 shows the final test errors, compared to previously reported results for the baselines and DBN [17].

The results show IterNorm improves the original WRN with BN and DBN on CIFAR-10. In particular, our methods reduce the test error to 3.56% on WRN-28-10, a relatively improvement of 8.5% in performance over 'Baseline'.

[2]https://github.com/szagoruyko/wide-residual-networks

Method	WRN-28-10	WRN-40-10
Baseline* [50]	3.89	3.80
DBN [17]	3.79 ± 0.09	3.74 ± 0.11
Baseline	3.89 ± 0.13	3.82 ± 0.11
IterNorm	**3.56 ± 0.12**	**3.59 ± 0.07**

Table 1. Test errors (%) on wide residual networks over CIFAR-10. All results are computed over 5 random seeds, and shown in the format of 'mean $\pm std$'. We replicate the 'Baseline' results based on the released code in [50], which computes the median of 5 runs on WRN-28-10 and only performs one run onWRN-40-10.

Method	Top-1	Top-5
Baseline* [11]	30.43	10.76
DBN-L1* [17]	29.87	10.36
Baseline	29.76	10.39
DBN-L1	29.50	10.26
IterNorm-L1	29.34	10.22
IterNorm-Full	29.30	10.21
IterNorm-L1 + DF	**28.86**	**10.08**

Table 2. Comparison of validation errors (%, single model and single-crop) on 18-layer residual networks on ILSVRC-2012. 'Baseline*' and 'DBN-L1*' indicate that the results are reported in [17] with training of 90 epochs.

5.3. Results on ImageNet with Residual Network

We validate the effectiveness of our methods on residual networks for ImageNet classification with 1000 classes [8]. We use the given official 1.28M training images as a training set, and evaluate the top-1 and top-5 classification errors on the validation set with 50k images.

Ablation Study on Res-18 We first execute an ablation study on the 18-layer residual network (Res-18) to explore multiple positions for replacing BN with IterNorm. The models used are as follows: (a) 'IterNorm-L1': we only replace the first BN module of ResNet-18, so that the decorrelated information from previous layers can pass directly to the later layers with the identity connections described in [17]; (b) We also replace all BN modules indicated as 'IterNorm-full'; We follow the same experimental setup as described in [11], except that we use 1 GPU and train over 100 epochs. We apply SGD with a mini-batch size of 256, momentum of 0.9 and weight decay of 0.0001. The initial learning rate is set to 0.1 and divided by 10 at 30, 60 and 90 epochs, and end the training at 100 epochs.

We find that only replacing the first BN effectively improves the performance of the original residual network, either by using DBN or IterNorm. Our IterNorm has marginally better performance than DBN. We find that replacing all the layers of IterNorm has no significant improvement over only replacing the first layer. We conjecture that the reason might be that the learned residual functions tend to have small response as shown in [11], and stretching this small response to the magnitude as the previous one may lead to negative effects. Based on 'IterNorm-L1', we further plug-in the IterNorm after the last average pooling (before the last linear layer) to learn the decorrelated feature representation. We find this significantly improves the performance, as shown in Table 2, referred to as 'IterNorm-L1 + DF'. Such a way to apply IterNorm can improve the original residual networks and introduce negligible computational cost. We also attempt to use DBN to decorrelate the feature representation. However, it always suffers the problems of that the eigen-decomposition can not converge.

Method	Res-50		Res-101	
	Top-1	Top-5	Top-1	Top-5
Baseline* [11]	24.70	7.80	23.60	7.10
Baseline	23.95	7.02	22.45	6.29
IterNorm-L1	23.28	6.72	21.95	5.99
IterNorm-L1 + DF	**22.91**	**6.47**	**21.77**	**5.94**

Table 3. Comparison of test errors (%, single model and single-crop) on 50/101-layer residual networks on ILSVRC-2012. 'Baseline*' indicates that the results are obtained from the website: https://github.com/KaimingHe/deep-residual-networks.

Results on Res-50/101 We further apply our method on the 50 and 101-layer residual network (ResNet-50 and ResNet-101) and perform single model and single-crop testing. We use the same experimental setup as before, except that we use 4 GPUs and train over 100 epochs. The results are shown in Table 3. We can see that the 'IterNorm-L1' achieves lower test errors compared to the original residual networks. 'IterNorm-L1 + DF ' further improves the performance.

6. Conclusions

In this paper, we proposed Iterative Normalization (IterNorm) based on Newton's iterations. It improved the optimization efficiency and generalization capability over standard BN by decorrelating activations, and improved the efficiency over DBN by avoiding the computationally expensive eigen-decomposition. We introduced Stochastic Normalization Disturbance (SND) to measure the inherent stochastic uncertainty in normalization. With the support of SND, we provided a thorough analysis regarding the performance of normalization methods, with respect to the batch size and feature dimensions, and showed that IterNorm has better trade-off between optimization and generalization. We demonstrated consistent performance improvements of IterNorm on the CIFAR-10 and ImageNet datasets. The analysis of combining conditioning and SND, can potentially lead to novel visions for future normalization work, and our proposed IterNorm can potentially to be used in designing network architectures.

References

[1] Devansh Arpit, Yingbo Zhou, Bhargava Urala Kota, and Venu Govindaraju. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. In *ICML*, 2016. 2

[2] Andrei Atanov, Arsenii Ashukha, Dmitry Molchanov, Kirill Neklyudov, and Dmitry Vetrov. Uncertainty estimation via stochastic batch normalization. In *ICLR Workshop*, 2018. 4

[3] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. 1, 2, 4

[4] Dario A. Bini, Nicholas J. Higham, and Beatrice Meini. Algorithms for the matrix pth root. *Numerical Algorithms*, 39(4):349–378, Aug 2005. 2, 3, 5, 6, 7

[5] Johan Bjorck, Carla Gomes, and Bart Selman. Understanding batch normalization. In *NIPS*, 2018. 1

[6] Michael Cogswell, Faruk Ahmed, Ross B. Girshick, Larry Zitnick, and Dhruv Batra. Reducing overfitting in deep networks by decorrelating representations. In *ICLR*, 2016. 1

[7] Tim Cooijmans, Nicolas Ballas, César Laurent, and Aaron C. Courville. Recurrent batch normalization. In *ICLR*, 2017. 2

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009. 2, 6, 8

[9] Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, and koray kavukcuoglu. Natural neural networks. In *NIPS*, 2015. 1, 2, 5

[10] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. 7

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 6, 8

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 1

[13] Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. *arXiv preprint arXiv:1803.01814*, 2018. 2

[14] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 1

[15] Lei Huang, Xianglong Liu, Bo Lang, Adams Wei Yu, Yongliang Wang, and Bo Li. Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. In *AAAI*, 2018. 2

[16] Lei Huang, Xianglong Liu, Yang Liu, Bo Lang, and Dacheng Tao. Centered weight normalization in accelerating training of deep neural networks. In *ICCV*, 2017. 2

[17] Lei Huang, Dawei Yang, Bo Lang, and Jia Deng. Decorrelated batch normalization. In *CVPR*, 2018. 1, 2, 3, 4, 5, 6, 7, 8

[18] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *NIPS*, 2017. 2, 4

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 1, 2, 3, 4, 6

[20] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *NIPS*. 2017. 2

[21] Jonas Kohler, Hadi Daneshmand, Aurelien Lucchi, Ming Zhou, Klaus Neymeyr, and Thomas Hofmann. Towards a theoretical understanding of batch normalization. *arXiv preprint arXiv:1805.10694*, 2018. 1

[22] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 2, 6

[23] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *NIPS*. 1992. 2

[24] César Laurent, Gabriel Pereyra, Philemon Brakel, Ying Zhang, and Yoshua Bengio. Batch normalized recurrent neural networks. In *ICASSP*, 2016. 2

[25] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Effiicient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–50, 1998. 1, 5

[26] Peihua Li, Jiangtao Xie, Qilong Wang, and Zilin Gao. Towards faster training of global covariance pooling networks by iterative matrix square root normalization. In *CVPR*, 2018. 2, 3

[27] Qianli Liao, Kenji Kawaguchi, and Tomaso Poggio. Streaming normalization: Towards simpler and more biologically-plausible normalizations for online and recurrent learning. *arXiv preprint arXiv:1610.06160*, 2016. 2

[28] Tsung-Yu Lin and Subhransu Maji. Improved bilinear pooling with cnns. In *BMVC*, 2017. 2

[29] Ping Luo. Learning deep architectures via generalized whitened neural networks. In *ICML*, 2017. 1, 2

[30] Ping Luo, Jiamin Ren, and Zhanglin Peng. Differentiable learning-to-normalize via switchable normalization. *arXiv preprint arXiv:1806.10779*, 2018. 2

[31] Grégoire Montavon and Klaus-Robert Müller. *Deep Boltzmann Machines and the Centering Trick*, volume 7700 of *LNCS*. Springer, 2nd edn edition, 2012. 2

[32] Behnam Neyshabur, Ruslan Salakhutdinov, and Nathan Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In *NIPS*, 2015. 2

[33] Tapani Raiko, Harri Valpola, and Yann LeCun. Deep learning made easier by linear transformations in perceptrons. In *AISTATS*, 2012. 2

[34] Mengye Ren, Renjie Liao, Raquel Urtasun, Fabian H. Sinz, and Richard S. Zemel. Normalizing the normalizers: Comparing and extending network normalization schemes. In *ICLR*, 2017. 2

[35] Pau Rodríguez, Jordi Gonzàlez, Guillem Cucurull, Josep M. Gonfaus, and F. Xavier Roca. Regularizing cnns with locally constrained decorrelations. In *ICLR*, 2017. 2

[36] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS*, 2016. 2

[37] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization?(no, it is not about internal covariate shift). In *NIPS*, 2018. 1

[38] Nicol N. Schraudolph. Accelerated gradient descent by factor-centering decomposition. Technical report, 1998. 2

[39] Alexander Shekhovtsov and Boris Flach. Normalization of neural networks using analytic variance propagation. In *Computer Vision Winter Workshop*, 2018. 2

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 6

[41] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014. 5

[42] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017. 1

[43] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 1

[44] Mattias Teye, Hossein Azizpour, and Kevin Smith. Bayesian uncertainty estimation for batch normalized deep networks. In *ICML*, 2018. 2, 4

[45] Guangrun Wang, Jiefeng Peng, Ping Luo, Xinjiang Wang, and Liang Lin. Kalman normalization: Normalizing internal representations across network layers. In *NIPS*, 2018. 2, 4

[46] Simon Wiesler, Alexander Richard, Ralf Schlüter, and Hermann Ney. Mean-normalized stochastic gradient for large-scale deep learning. In *ICASSP*, 2014. 2

[47] Shuang Wu, Guoqi Li, Lei Deng, Liu Liu, Yuan Xie, and Luping Shi. L1-norm batch normalization for efficient training of deep neural networks. *CoRR*, 2018. 2

[48] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018. 1, 2, 4, 6

[49] Wei Xiong, Bo Du, Lefei Zhang, Ruimin Hu, and Dacheng Tao. Regularizing deep convolutional neural networks with a structured decorrelation constraint. In *ICDM*, 2016. 1

[50] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016. 1, 7, 8