Normalization Techniques in Training DNNs: Methodology, Analysis and Application

Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, Ling Shao

Abstract—Normalization techniques are essential for accelerating the training and improving the generalization of deep neural networks (DNNs), and have successfully been used in various applications. This paper reviews and comments on the past, present and future of normalization methods in the context of DNN training. We provide a unified picture of the main motivation behind different approaches from the perspective of optimization, and present a taxonomy for understanding the similarities and differences between them. Specifically, we decompose the pipeline of the most representative normalizing activation methods into three components: the normalization area partitioning, normalization operation and normalization representation recovery. In doing so, we provide insight for designing new normalization technique. Finally, we discuss the current progress in understanding normalization methods, and provide a comprehensive review of the applications of normalization for particular tasks, in which it can effectively solve the key issues.

Index Terms—Deep neural networks, batch normalization, weight normalization, image classification, survey

1 Introduction

Deep neural networks (DNNs) have been extensively used across a broad range of applications, including computer vision (CV), natural language processing (NLP), speech and audio processing, robotics, bioinformatics, *etc*. [1]. They are typically composed of stacked layers/modules, the transformation between which consists of a linear mapping with learnable parameters and a nonlinear activation function [2]. While their deep and complex structure provides them powerful representation capacity and appealing advantages in learning feature hierarchies, it also makes their training difficult [3], [4]. In fact, the success of DNNs heavily depends on breakthroughs in training techniques [5], [6], [7], [8], which has been witnessed by the history of deep learning [1].

One milestone technique in addressing the training issues of DNNs was batch normalization (BN) [8], which standardizes the activations of intermediate DNN layers within a mini-batch of data. BN improves DNNs' training stability, optimization efficiency and generalization ability. It is a basic component in most state-of-the-art architectures [9], [10], [11], [12], [13], [14], [15], [16], and has successfully proliferated throughout various areas of deep learning [17], [18], [19]. Further, a significant number of other normalization techniques have been proposed to address the training issues in particular contexts, further evolving the DNN architectures and their applications [20], [21], [22], [23], [24]. For example, layer normalization (LN) [20] is an essential module in Transformer [25], which has advanced the state-of-the-art architectures for NLP [25], [26], [27], [28], while spectral normalization [23] is a basic component in the discriminator of generative adversarial networks (GANs) [23], [29], [30]. Importantly, the ability of most normalization techniques to stabilize and accelerate training has helped to simplify the process of designing network architectures—training is no longer the main concern, enabling more focus to be given to developing components that can effectively encode prior/domain knowledge into the architectures.

• *Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu and Ling Shao are with the Inception Institute of Artificial Intelligence, Abu Dhabi, UAE.*

Corresponding author: Lei Huang (huanglei36060520@gmail.com).

However, despite the abundance and ever more important roles of normalization techniques, we note that there is an absence of a unifying lens with which to describe, compare and analyze them. This paper provides a review and commentary on normalization techniques in the context of training DNNs. To the best of our knowledge, our work is the first survey paper to cover normalization methods, analyses and applications. We attempt to provide answers for the following questions:

(1) What are the main motivations behind different normalization methods in DNNs, and how can we present a taxonomy for understanding the similarities and differences between a wide variety of approaches?

(2) How can we reduce the gap between the empirical success of normalization techniques and our theoretical understanding of them?

(3) What recent advances have been made in designing/tailoring normalization techniques for different tasks, and what are the main insights behind them?

We answer the first question by providing a unified picture of the main motivations behind different normalization methods, from the perspective of optimization (Section 3). We show that most normalization methods are essentially designed to satisfy nearly equal statistical distributions of layer input/output-gradients across different layers during training, in order to avoid the ill-conditioned landscape of optimization. Based on this, we provide a comprehensive review of the normalization methods, including normalizing activations by population statistics (Section 4), normalizing activations as functions (Section 5), normalizing weights (Section 6) and normalizing gradients (Section 7). Specifically, we decompose the most representative normalizing-activations-as-functions framework into three components: the normalization area partitioning (NAP), normalization operation (NOP) and normalization representation recovery (NRR). We unify most normalizing-activations-as-function methods into this framework, and provide insights for designing new normalization methods.

To answer the second question, we discuss the recent progress in our theoretical understanding of BN in Section 8. It is difficult to fully analyze the inner workings of BN in a unified framework, but

our review ultimately provides clear guidelines for understanding why BN stabilizes and accelerates training, and further improves generalization, through a scale-invariant analysis, condition analysis and stochasticity analysis, respectively.

We answer the third question in Section 9 by providing a review of the applications of normalization for particular tasks, and illustrating how normalization methods can be used to solve key issues. To be specific, we mainly review the applications of normalization in domain adaptation, style transfer, training GANs and efficient deep models. We show that the normalization methods can be used to 'edit' the statistical properties of layer activations. These statistical properties, when designed well, can represent the style information for a particular image or the domain-specific information for a distribution of a set of images. This characteristic of normalization methods has been thoroughly exploited in CV tasks and potentially beyond them.

We conclude the paper with additional thoughts about certain open questions in the research of normalization techniques.

2 Denotations and Definitions

In this paper, we use a lowercase letter $x \in \mathbb{R}$ to denote a scalar, boldface lowercase letter $\mathbf{x} \in \mathbb{R}^d$ for a vector, boldface uppercase letter for a matrix $\boldsymbol{X} \in \mathbb{R}^{d\times m}$, and boldface sans-serif notation for a tensor $\mathbf{X}$, where $\mathbb{R}$ is the set of real-valued numbers, and d, m are positive integers. Note that a tensor is a more general entity. Scalars, vectors and matrices can be viewed as 0th-order, 1st-order and 2nd-order tensors. Here, $\mathbf{X}$ denotes a tensor with an order larger than 2. We will provide a more precise definition in the later sections. We follow matrix notation where the vector is in column form, except that the derivative is a row vector.

2.1 Optimization Objective

Consider a true data distribution $p_*(\mathbf{x}, \mathbf{y}) = p(\mathbf{x})p(\mathbf{y}|\mathbf{x})$ and the sampled training sets $\mathbb{D} \sim p_*(\mathbf{x}, \mathbf{y})$ of size N. We focus on a supervised learning task aiming to learn the conditional distribution $p(\mathbf{y}|\mathbf{x})$ using the model $q(\mathbf{y}|\mathbf{x})$, where $q(\mathbf{y}|\mathbf{x})$ is represented as a function $f_\theta(\mathbf{x})$ parameterized by θ. Training the model can be viewed as tuning the parameters to minimize the discrepancy between the desired output $\mathbf{y}$ and the predicted output $f(\mathbf{x};\theta)$. This discrepancy is usually described by a loss function $\ell(\mathbf{y}, f(\mathbf{x};\theta))$ for each sample pair $(\mathbf{x}, \mathbf{y})$. The empirical risk, averaged over the sample loss in training sets $\mathbb{D}$, is defined as:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(\ell(\mathbf{y}^{(i)}, f_\theta(\mathbf{x}^{(i)}))). \tag{1}$$

This paper mainly focuses on discussing the empirical risk from the perspective of optimization. We do not explicitly analyze the risk under the true data distribution $\mathcal{L}^*(\theta) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p_*(\mathbf{x},\mathbf{y})}(\ell(\mathbf{y}^{(i)}, f_\theta(\mathbf{x}^{(i)})))$ from the perspective of generalization.

2.2 Neural Networks

The function $f(\mathbf{x};\theta)$ adopted by neural networks usually consists of stacked layers. For a multilayer perceptron (MLP), $f_\theta(\mathbf{x})$ can be represented as a layer-wise linear and nonlinear transformation, as follows:

$$\mathbf{h}^l = \boldsymbol{W}^l\mathbf{x}^{l-1}, \tag{2}$$
$$\mathbf{x}^l = \phi(\mathbf{h}^l), \quad l = 1, ..., L, \tag{3}$$

where $\mathbf{x}^0 = \mathbf{x}$, $\boldsymbol{W}^l \in \mathbb{R}^{d_l\times d_{l-1}}$ and d_l indicates the number of neurons in the l-th layer. The learnable parameters $\theta = \{\boldsymbol{W}^l, l = 1, ..., L\}$. Typically, $\mathbf{h}^l$ and $\mathbf{x}^l$ are referred to as the pre-activation and activation, respectively, but in this paper, we refer to both as activations for simplicity. We also set $\mathbf{x}^L = \mathbf{h}^L$ as the output of the network $f_\theta(\mathbf{x})$ to simplify denotations.

Convolutional Layer: The convolutional layer parameterized by weights $\mathbf{W} \in \mathbb{R}^{d_l\times d_{l-1}\times F_h\times F_w}$, where F_h and F_w are the height and width of the filter, takes feature maps (activations) $\mathbf{X} \in \mathbb{R}^{d_{l-1}\times h\times w}$ as input, where h and w are the height and width of the feature maps, respectively. We denote the set of spatial locations as Δ and the set of spatial offsets as Ω. For each output feature map k and its spatial location $\delta \in \Delta$, the convolutional layer computes the pre-activation $\{\mathit{H}_{k,\delta}\}$ as: $\mathit{H}_{k,\delta} = \sum_{i=1}^{d_{l-1}}\sum_{\tau\in\Omega}\mathit{W}_{k,i,\tau}\mathit{X}_{i,\delta+\tau} =< \mathbf{w}_k, \mathbf{x}_\delta >$. Therefore, the convolution operation is a linear (dot) transformation. Here, $\mathbf{w}_k \in \mathbb{R}^{d_{l-1}\cdot F_h\cdot F_w}$ can eventually be viewed as an unrolled filter produced by $\mathbf{W}$.

2.3 Training DNNs

From an optimization perspective, we aim to minimize the empirical risk $\mathcal{L}$, as:

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta). \tag{4}$$

In general, the gradient descent (GD) update is used to minimize $\mathcal{L}$, seeking to iteratively reduce the loss as:

$$\theta_{t+1} = \theta_t - \eta\frac{\partial\mathcal{L}}{\partial\theta}, \tag{5}$$

where η is the learning rate. For large-scale learning, stochastic gradient descent (SGD) is extensively used to approximate the gradients $\frac{\partial\mathcal{L}}{\partial\theta}$ with a mini-batch gradient. One essential step is to calculate the gradients. This can be done by backpropagation for calculating $\frac{\partial\mathcal{L}}{\partial\mathbf{x}^{l-1}}$:

$$\frac{\partial\ell}{\partial\mathbf{x}^{l-1}} = \frac{\partial\ell}{\partial\mathbf{h}^l}\boldsymbol{W}^l, \tag{6}$$
$$\frac{\partial\ell}{\partial\mathbf{h}^{l-1}} = \frac{\partial\ell}{\partial\mathbf{x}^{l-1}}\phi'(\mathbf{h}^{l-1}),\ l = L, ..., 2, \tag{7}$$

and $\frac{\partial\ell}{\partial\boldsymbol{W}^l}$:

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{W}^l} = \mathbb{E}_\mathbb{D}[(\mathbf{x}^{l-1}\frac{\partial\ell}{\partial\mathbf{h}^l})^T], \quad l = L, ..., 1. \tag{8}$$

2.4 Normalization

Normalization is widely used in data-preprocessing [9], [31], [32], data mining and other areas. The definition of normalization may vary among different topics. In this paper, we define normalization as a general transformation, which ensures that the transformed data has certain statistical properties. To be more specific, we provide the following formal definition.

Definition 2.1. ***Normalization:*** *Given a set of data* $\mathbb{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N$, *the normalization operation is a function* $\Phi : \mathbf{x} \longmapsto \hat{\mathbf{x}}$, *which ensures that the transformed data* $\widehat{\mathbb{D}} = \{\hat{\mathbf{x}}^{(i)}\}_{i=1}^N$ *has certain statistical properties.*

We consider five main normalization operations (Figure 1) in this paper: centering, scaling, decorrelating, standardizing and whitening [33].

Centering formulates the transformation as:

$$\hat{\mathbf{x}} = \Phi_C(\mathbf{x}) = \mathbf{x} - \mathbb{E}_\mathbb{D}(\mathbf{x}). \tag{9}$$

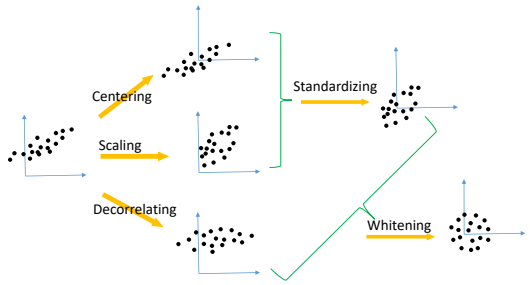

Fig. 1. Illustration of normalization operations discussed in this paper.

This ensures that the normalized output $\hat{\mathbf{x}}$ has a zero-mean property, which can be represented as: $\mathbb{E}_{\hat{\mathbb{D}}}(\mathbf{x}) = \mathbf{0}$.

Scaling formulates the transformation as:

$$\hat{\mathbf{x}} = \Phi_{SC}(\mathbf{x}) = \Lambda^{-\frac{1}{2}}\mathbf{x}. \tag{10}$$

Here, $\Lambda = \text{diag}(\sigma_1^2, \ldots, \sigma_d^2)$, where σ_j^2 is the mean square over data samples for the i-th dimension: $\sigma_j^2 = \mathbb{E}_{\mathbb{D}}(\mathbf{x}_j^2)$. Scaling ensures that the normalized output $\hat{\mathbf{x}}$ has a unit-variance property, which can be represented as: $\mathbb{E}_{\hat{\mathbb{D}}}(\hat{\mathbf{x}}_j^2) = 1$ for all $j = 1, ..., d$.

Decorrelating formulates the transformation as:

$$\hat{\mathbf{x}} = \Phi_D(\mathbf{x}) = \boldsymbol{D}\mathbf{x}, \tag{11}$$

where $\boldsymbol{D} = [\mathbf{d}_1, ..., \mathbf{d}_d]$ are the eigenvectors of Σ and $\Sigma = \mathbb{E}_{\mathbb{D}}(\mathbf{x}\mathbf{x}^T)$ is the covariance matrix. Decorrelating ensures that the correlation between different dimensions of the normalized output $\hat{\mathbf{x}}$ is zero (the covariance matrix $\mathbb{E}_{\hat{\mathbb{D}}}(\hat{\mathbf{x}}\hat{\mathbf{x}}^T)$ is a diagonal matrix).

Standardizing is a composition operation that combines centering and scaling, as:

$$\hat{\mathbf{x}} = \Phi_{ST}(\mathbf{x}) = \Lambda^{-\frac{1}{2}}(\mathbf{x} - \mathbb{E}_{\mathbb{D}}(\mathbf{x})). \tag{12}$$

Standardizing ensures that the normalized output $\hat{\mathbf{x}}$ has zero-mean and unit-variance properties.

Whitening formulates the transformation as[1]:

$$\hat{\mathbf{x}} = \Phi_W(\mathbf{x}) = \tilde{\Lambda}^{-\frac{1}{2}}\boldsymbol{D}\mathbf{x}, \tag{13}$$

where $\tilde{\Lambda} = \text{diag}(\tilde{\sigma}_1, \ldots, \tilde{\sigma}_d)$ and $\boldsymbol{D} = [\mathbf{d}_1, ..., \mathbf{d}_d]$ are the eigenvalues and associated eigenvectors of covariance matrix Σ. Whitening ensures that the normalized output $\hat{\mathbf{x}}$ has a spherical Gaussian distribution, which can be represented as: $\mathbb{E}_{\mathbb{D}}(\hat{\mathbf{x}}\hat{\mathbf{x}}^T) = \mathbf{I}$. The whitening transformation, defined in Eqn. 13, is called principal components analysis (PCA) whitening, where the whitening matrix $\Sigma_{PCA}^{-\frac{1}{2}} = \tilde{\Lambda}_d^{-\frac{1}{2}}\mathbf{D}$. There are an infinite number of whitening matrices since a whitened input stays whitened after an arbitrary rotation [33], [35], which will be discussed in the subsequent sections.

3 Motivation and Overview of Normalization in DNNs

Input normalization is extensively used in machine learning models. Intuitively, normalizing an input removes the difference in magnitude between different features, which benefits learning. There are also theoretical advantages to normalization for linear models.

1. Whitening usually requires the input to be centered [33], [34], which means it also includes the centering operation. In this paper, we unify the operation as whitening regardless of whether it includes centering or not.

Consider a linear regression model with a scalar output $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, and mean square error loss $\ell = (y - f_\theta(\mathbf{x}))^2$. As shown in [31], [36], the learning dynamics for such a quadratic surface are fully controlled by the spectrum of the Hessian matrix $\mathbf{H} = \mathbb{E}_{\mathbb{D}}(\mathbf{x}\mathbf{x}^T)$. There are two statistical momentums that are essential for evaluating the convergence behaviors of the optimization problem. One is the maximum eigenvalue of the curvature matrix λ_{max}, and the other is the condition number of the curvature matrix, denoted by $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$, where λ_{min} is the minimum nonzero eigenvalue of the curvature matrix. Specifically, λ_{max} controls the upper bound and the optimal learning rate (*e.g.*, the training will diverge if $\eta \geq \frac{2}{\lambda_{max}(\mathbf{H})}$). Meanwhile, κ controls the number of iterations required for convergence (*e.g.*, the lower bound of the iterations is $\kappa(\mathbf{H})$ [31], [36], [37]). If $\mathbf{H}$ is an identity matrix that can be obtained by whitening the input through Eqn. 13, the GD can converge within only one iteration. Therefore, normalizing the input can surely accelerate convergence during the optimization of linear models.

These theoretical results do not apply to neural networks directly, since the input $\mathbf{x}$ is only directly connected to the first weight matrix $\boldsymbol{W}^1$, and it is not clear how input $\mathbf{x}$ affects the sub-landscapes of optimization with respect to other weight matrices $\boldsymbol{W}^l, l = 2, ..., L$. Luckily, the layer-wise structure (Eqn. 2) of neural networks can be exploited to approximate the curvature matrix, *e.g.*, the Fisher information matrix (FIM). One successful example is approximating the FIM of DNNs using the Kronecker product (K-FAC) [38], [39], [40], [41]. In the K-FAC approach, there are two assumptions: 1) weight-gradients in different layers are assumed to be uncorrelated; 2) the input and output-gradient in each layer are approximated as independent. Thus, the full FIM can be represented as a block diagonal matrix, $\mathbf{F} = diag(F_1, ..., F_L)$, where F_l is the sub-FIM (the FIM with respect to the parameters in a certain layer) and computed as:

$$F_l \approx \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\mathbf{x}^{l-1}(\mathbf{x}^{l-1})^T] \otimes \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p(\mathbf{x})q(\mathbf{y}|\mathbf{x})}[\frac{\partial \ell}{\partial \mathbf{h}^l}^T \frac{\partial \ell}{\partial \mathbf{h}^l}]. \tag{14}$$

Note that [38], [42], [43] have provided empirical evidence to support their effectiveness in approximating the full FIM with block diagonal sub-FIMs. We denote the covariance matrix of the layer input as $\Sigma_{\mathbf{x}}^l = \mathbb{E}_{p(\mathbf{x})}[\mathbf{x}^{l-1}(\mathbf{x}^{l-1})^T]$ and the covariance matrix of the layer output-gradient as $\Sigma_{\nabla\mathbf{h}}^l = \mathbb{E}_{q(\mathbf{y}|\mathbf{x})}[\frac{\partial \ell}{\partial \mathbf{h}^l}^T \frac{\partial \ell}{\partial \mathbf{h}^l}]$. Based on the K-FAC, it is clear that the conditioning of the FIM can be improved, if:

- *Criteria 1*: The statistics of the layer input (*e.g.*, $\Sigma_{\mathbf{x}}$) and output-gradient (*e.g.*, $\Sigma_{\nabla\mathbf{h}}$) across different layers are equal.
- *Criteria 2*: $\Sigma_{\mathbf{x}}$ and $\Sigma_{\nabla\mathbf{h}}$ are well conditioned.

A variety of techniques for training DNNs have been designed to satisfy *Criteria 1* and/or *Criteria 2*, essentially. For example, the weight initialization techniques aim to satisfy *Criteria 1*, obtaining nearly equal variances for layer input/output-gradients across different layers [3], [44], [45], [46], [47] by designing initial weight matrices. However, the equal-variance property across layers can be broken down and is not necessarily sustained throughout training, due to the update of weight matrices. From this perspective, it is important to normalize the activations in order to produce better-conditioned optimization landscapes, similar to the benefits of normalizing the input.

Normalizing activations is more challenging than normalizing an input with a fixed distribution, since the distribution of layer activations $\mathbf{x}^l$ varies during training. Besides, DNNs are usually optimized over stochastic or mini-batch gradients, rather than the full gradient, which requires more efficient statistical estimations

for activations. This paper discusses three types of normalization methods for improving the performance of DNN training:

(1) Normalizing the activations directly to (approximately) satisfy Criteria 1 and/or Criteria 2. Generally speaking, there are two strategies for normalizing the activations of DNNs. One is to normalize the activations using the population statistics estimated over the distribution of activations [48], [49], [50]. The other strategy is to normalize the activations as a function transformation, which requires backpropagation through this transformation [8], [20], [34].

(2) Normalizing the weights with a constrained distribution, such that the activations/output-gradients (Eqns 2 and 6) can be implicitly normalized. This normalization strategy is inspired by weight initialization methods, but extends them towards satisfying the desired property during training [21], [23], [51], [52].

(3) Normalizing gradients to exploit the curvature information for GD/SGD, even though the optimization landscape is ill-conditioned [53], [54]. This involves performing normalization solely on the gradients, which may effectively remove the negative effects of the ill-conditioned landscape caused by the diversity of magnitude in gradients from different layers (*i.e.*, Criteria 1 is not well satisfied) [3].

4 Normalizing Activations by Population Statistics

In this section, we will discuss the methods that normalize activations using the population statistics estimated over their distribution. This normalization strategy views the population statistics as constant during backpropagation. To simplify the notation, we remove the layer index l of activations $\mathbf{x}^l$ in the subsequent sections, unless otherwise stated.

Gregoire *et al*. [48] proposed to center the activations (hidden units) in a Boltzmann machine to improve the conditioning of the optimization problems, based on the insight that centered inputs improve the conditioning [31], [55], [56]. Specifically, given the activation in a certain layer $\mathbf{x}$, they perform the normalization as:

$$\hat{\mathbf{x}} = \mathbf{x} - \hat{\mathbf{u}}, \tag{15}$$

where $\hat{\mathbf{u}}$ is the mean of activations over the training dataset. Note that $\hat{\mathbf{u}}$ indicates the population statistics that need to be estimated during training, and is considered as constant during backpropagation. In [48], $\hat{\mathbf{u}}$ is estimated by running averages. Wiesler *et al*. [49] also considered centering the activations to improve the performance of DNNs, reformulating the centering normalization by re-parameterization. This can be viewed as a pre-conditioning method. They also used running average to estimate $\hat{\mathbf{u}}$ based on the mini-batch activations. One interesting observation in [49] is that the scaling operation does not yield improvements in this case. One likely reason is that the population statistics estimated by running average are not accurate, and thus cannot adequately exploit the advantages of standardization.

Desjardins *et al*. [50] proposed to whiten the activations using the population statistics as:

$$\hat{\mathbf{x}} = \widehat{\Sigma}^{-\frac{1}{2}}(\mathbf{x} - \hat{\mathbf{u}}), \tag{16}$$

where $\widehat{\Sigma}^{-\frac{1}{2}}$ is the population statistics of the whitening matrix. One difficulty is to accurately estimate $\widehat{\Sigma}^{-\frac{1}{2}}$. In [50], [57], $\widehat{\Sigma}^{-\frac{1}{2}}$ is updated over T intervals, and the whitening matrix is further pre-conditioned by one hyperparameter ϵ, which balances the natural gradient (produced by the whitened activations) and the naive gradient. With these two techniques, the networks with whitened activations can be trained by finely adjusting T/ϵ. Luo [57] investigated the effectiveness of whitening the activations (pre-whitening) and pre-activations (post-whitening). They also addressed the computational issues by using online singular value decomposition (SVD) when calculating the whitening matrix.

Although several improvements in performance have been achieved, normalization by population statistics still faces some drawbacks. The main disadvantage is the training instability, which can be caused by the inaccurate estimation of population statistics: 1) These methods usually use a limited number of data samples to estimate the population statistics, due to computational concerns. 2) Even if full data is available and an accurate estimation is obtained for a current iteration, the activation distribution (and thus the population statistics) will change due to the updates of the weight matrix, which is known as internal covariant shift (ICS) [8]. 3) Finally, an inaccurate estimation of population statistics will be amplified as the number of layers increases, so these methods are not suitable for large-scale networks. As pointed out in [50], [57], additional batch normalization [8] is needed to stabilize the training for large-scale networks.

5 Normalizing Activations as Functions

BN [8] paved the way to viewing normalization statistics as functions over mini-batch inputs, and addressing backpropagation through normalization operations. Let x denote the activation for a given neuron in one layer of a DNN. BN [8] standardizes the neuron within m mini-batch data by:

$$\hat{x}^{(i)} = \frac{x^{(i)} - u}{\sqrt{\sigma^2 + \epsilon}}, \tag{17}$$

where $\epsilon > 0$ is a small number to prevent numerical instability, and $u = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$ and $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - u)^2$ are the mean and variance, respectively.[2] During inference, the population statistics $\{\hat{u}, \hat{\sigma}^2\}$ are required for deterministic inference, and they are usually calculated by running average over the training iterations, as follows:

$$\begin{cases} \hat{u} = (1-\lambda)\hat{u} + \lambda u, \\ \hat{\sigma}^2 = (1-\lambda)\hat{\sigma}^2 + \lambda\sigma^2. \end{cases} \tag{18}$$

Compared to the normalization methods based on population statistics, introduced in Section 4, this normalization strategy provides several advantages: 1) It avoids using the population statistics to normalize the activations, thus avoiding the instability caused by inaccurate estimations. 2) The normalized output for each mini-batch has a zero-mean and unit-variance constraint that stabilizes the distribution of the activations, and thus benefits training. For more discussions please refer to the subsequent Section 8.

Due to the constraints introduced by standardization, BN also uses an additional learnable scale parameter $\gamma \in \mathbb{R}$ and shift parameter $\beta \in \mathbb{R}$ to recover a possible reduced representation capacity [8]:

$$\tilde{x} = \gamma\hat{x} + \beta. \tag{19}$$

In this paper, we also refer to the scale parameter and shift parameter as affine parameters. BN has been shown to be a milestone in the deep learning community [9], [12], [22]. It is

2. Note that here u and σ^2 are functions over the mini-batch data.

Algorithm 1 Framework of algorithms normalizing activations as functions.

1: **Input**: mini-batch inputs $\mathbf{X} \in \mathbb{R}^{d\times m\times h\times w}$.
2: **Output**: $\widetilde{\mathbf{X}} \in \mathbb{R}^{d\times m\times h\times w}$.
3: Normalization area partitioning: $\boldsymbol{X} = \Pi(\mathbf{X})$.
4: Normalization operation: $\widehat{\boldsymbol{X}} = \Phi(\boldsymbol{X})$.
5: Normalization representation recovery: $\widetilde{\boldsymbol{X}} = \Psi(\widehat{\boldsymbol{X}})$.
6: Reshape back: $\widetilde{\mathbf{X}} = \Pi^{-1}(\widetilde{\boldsymbol{X}})$.

widely used in different networks [9], [10], [11], [12], [13], [15], [16], [58] and various applications [59]. However, despite its great success in deep learning, BN still faces several issues in particular contexts: 1) The inconsistent operation of BN between training and inference limits its usage in complex networks (*e.g.* recurrent neural networks (RNNs) [20], [60], [61]) and tasks [29], [62], [63]; 2) BN suffers from the small-batch-size problem — its error increases rapidly as the batch size becomes smaller [22]. To address BN's weaknesses and further extend its functionality, plenty of works related to feature normalization have been proposed.

In the following sections, we first propose a framework to describe normalizing-activations-as-function methods in Algorithm 1, and review the basic single-mode normalization methods, which ensure that the normalized output has a single-mode (Gaussian) distribution. We then introduce the approaches that extend single-mode method to multiple modes, and that further combine different normalization methods. Lastly, we discuss the more robust estimation methods that address the small-batch-size problem of BN.

5.1 A Framework for Decomposing Normalization

We divide the normalizing-activations-as-function framework into three abstract processes: normalization area partitioning (NAP), normalization operation (NOP), and normalization representation recovery (NRR). We consider the more general mini-batch (of size m) activations in a convolutional layer $\mathbf{X} \in \mathbb{R}^{d\times m\times h\times w}$, where d, h and w are the channel number, height and width of the feature maps, respectively.[3] NAP transforms the activations $\mathbf{X}$ into $\boldsymbol{X} \in \mathbb{R}^{S_1\times S_2}$[4], where S_2 indexes the set of samples used to compute the estimators. NOP denotes the specific normalization operation (see main operations in Section 2) on the transformed data $\boldsymbol{X}$. NRR is used to recover the possible reduced representation capacity.

Take BN as an example. BN [8] considers each spatial position in a feature map as a sample [8], [66] and the NAP is:

$$\boldsymbol{X} = \Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d\times mhw}, \tag{20}$$

which means that the statistics are calculated along the batch, height, and width dimensions. The NOP is the standardization, represented in the form of a matrix as:

$$\widehat{\boldsymbol{X}} = \Phi_{SD}(\boldsymbol{X}) = \Lambda^{-\frac{1}{2}}(\boldsymbol{X} - \mathbf{u}\mathbf{1}^T). \tag{21}$$

Here, $\mathbf{u}$ is the mean of data samples, $\mathbf{1}$ is a column vector of all ones, and $\Lambda_d = \text{diag}(\sigma_1^2, \ldots, \sigma_d^2) + \epsilon\boldsymbol{I}$, where σ_j^2 is the variance over data samples for the j-th neuron/channel. The NRR is the affine transformation with channel-wise learnable affine parameters $\gamma, \beta \in \mathbb{R}^d$, defined as:

$$\widetilde{\boldsymbol{X}} = \Psi_{AF}(\widehat{\boldsymbol{X}}) = \widehat{\boldsymbol{X}} \odot (\gamma\mathbf{1}^T) + (\beta\mathbf{1}^T). \tag{22}$$

In the following sections, we will discuss the research progress along these three lines.

5.1.1 Normalization Area Partitioning

In this section, the default NOP is the standardization operation (Eqn. 21), and the NRR is the affine transform (Eqn. 22).

LN [20] proposes to standardize the layer input within the neurons for each training sample, to avoid the drawbacks of normalization along batch dimensions. Specifically, the NAP of LN is $\boldsymbol{X} = \Pi_{LN}(\mathbf{X}) \in \mathbb{R}^{m\times dhw}$, where the normalization statistics are calculated along the channel, height and width dimensions. LN has the same formulation during training and inference, and is extensively used in NLP tasks [25], [26], [27].

Group normalization (GN) [22] generalizes LN, dividing the neurons into groups and standardizing the layer input within the neurons of each group for each sample independently. Specifically, the NAP of GN is $\boldsymbol{X} = \Pi_{GN}(\mathbf{X}) \in \mathbb{R}^{mg\times shw}$, where g is the group number and $d = gs$. LN is clearly a special case of GN with $g = 1$. By changing the group number g, GN is more flexible than LN, enabling it to achieve good performance on visual tasks limited to small-batch-size training (*e.g.*, object detection and segmentation [22]).

Instance normalization (IN) [67] proposes to normalize each single image to remove instance-specific contrast information. Specifically, the NAP of IN is $\boldsymbol{X} = \Pi_{IN}(\mathbf{X}) \in \mathbb{R}^{md\times hw}$. Due to its ability to remove style information from the inputs, IN is widely used in image style transfer tasks [68], [69], [70].

Position normalization (PN) [71] standardizes the activations at each position independently across the channels. The NAP of PN is $\boldsymbol{X} = \Pi_{PN}(\mathbf{X}) \in \mathbb{R}^{mhw\times d}$. PN is designed to deal with spatial information, and has the potential to enhance the performance of generative models [71], [72].

Batch group normalization (BGN) [73] expands the grouping mechanism of GN from being over only channels to being over both channels and batch dimensions. The NAP of BGN is $\boldsymbol{X} = \Pi_{BGN}(\mathbf{X}) \in \mathbb{R}^{g_m g\times s_m shw}$, where $m = g_m s_m$. BGN also normalizes over batch dimensions and needs to estimate the population statistics, similar to BN in Eqn. 18. However, the group mechanism adds 'examples' for normalization, thus relieving the small-batch problem of BN to some degree.

Local Normalization: In the above normalization methods, the statistics are shared by all examples/positions in the same area that are used to calculate these statistics. There are also methods in which the statistics for each example/position are calculated over the neighboring regions and thus vary. This kind of normalization is called *local normalization* [74]. Jarrett *et al.* proposed local contrast normalization (LCN) to standardize each example's feature using the statistics calculated by its neighbors in a window of size 9×9. Local response normalization (LRN) [75] proposes to scale the activation across n 'adjacent' kernel maps at the same spatial position. Local contex normalization [74] extends the neighborhood partition, where the normalization is performed within a window of size $p \times q$, for groups of filters with a size predefined by the number of channels per group (c_groups) along the channel axis. Divisive normalization (DN) [76] generalizes the neighborhood partition of these local normalization methods as the choices of the summation and suppression fields [76], [77].

3. Note that the convolutional activation is reduced to the MLP activation, when setting $h = w = 1$.

4. NAP can be implemented by the reshape operation of PyTorch [64] or Tensorflow [65].

5.1.2 Normalization Operation

As previously discussed, current normalization methods usually use a standardization operation. However, other operations can also be used to normalize the data. We divide these operations into three categories: 1) Extending standardization towards the whitening operation, which is a more general operation; 2) Variations of standardization; 3) Reduced standardizations that use only centering or scaling for some special situations. Unless otherwise stated, the NAP is Π_{BN}, the data transferred after the NAP is denoted as $\boldsymbol{X} \in \mathbb{R}^{d\times m}$, and the NRR is the affine transform as shown in Eqn. 22.

Beyond Standardization Towards Whitening: Huang *et al.* proposed decorrelated BN [34], which extends BN to batch whitening (BW). The NOP of BW is whitening, represented as:

$$\widehat{\boldsymbol{X}} = \Phi_W(\boldsymbol{X}) = \Sigma^{-\frac{1}{2}}(\boldsymbol{X} - \mathbf{u}\mathbf{1}^T). \tag{23}$$

Here, $\Sigma^{-\frac{1}{2}}$ is the whitening matrix, which is calculated from the corresponding mini-batch covariance matrix $\Sigma = \frac{1}{m}(\boldsymbol{X} - \mathbf{u}\mathbf{1}^T)(\boldsymbol{X} - \mathbf{u}\mathbf{1}^T)^T + \epsilon\boldsymbol{I}$.

One main challenge for extending standardization to whitening is how to back-propagate through the inverse square root of a matrix (*i.e.* $\partial\Sigma^{-\frac{1}{2}}/\partial\Sigma$) [8], [34]. This can be achieved by using matrix differential calculus [78], as proposed in [34]. One interesting question is the choice of how to compute the whitening matrix $\Sigma^{-\frac{1}{2}}$. PCA based BW with $\Sigma_{PCA}^{-\frac{1}{2}} = \tilde{\Lambda}^{-\frac{1}{2}}\boldsymbol{D}$ suffers significant instability in training DNNs and hardly converges, due to the so-called stochastic axis swapping (SAS), as explained in [34]. Zero-phase component analysis (ZCA) whitening, using $\Sigma_{ZCA}^{-\frac{1}{2}} = \boldsymbol{D}\tilde{\Lambda}^{-\frac{1}{2}}\boldsymbol{D}^T$, is advocated for in [34], where the PCA-whitened input is rotated back by the corresponding rotation matrix $\boldsymbol{D}$. ZCA whitening has been shown to avoid the SAS issue and achieve better performance over standardization (used in BN) on discriminative classification tasks [34]. Siarohin *et al.* [79] used Cholesky decomposition (CD) based whitening $\Sigma_{CD}^{-\frac{1}{2}} = \boldsymbol{L}^{-1}$, where $\boldsymbol{L}$ is a lower triangular matrix from the CD, with $\boldsymbol{L}\boldsymbol{L}^T = \Sigma$. CD whitening has been shown to achieve state-of-the-art performance in training GANs. For more details on comparing different whitening methods for training DNNs, please refer to [35].

Given a particular batch size, BW may not have enough samples to obtain a suitable estimate for the full covariance matrix, which can heavily harm the performance. Group-based BW—where features are divided into groups and whitening is performed within each one—was proposed [34], [80] to control the extent of the whitening. One interesting property is that group-based BW reduces to BN if the channel number in each group is set to 1. Besides, group-based BW also has the added benefit of reducing the computational cost of whitening. Later, Huang *et al.* proposed iterative normalization (IterNorm) [81] to improve the computational efficiency and numerical stability of ZCA whitening, since it can avoid eigen-decomposition or SVD by employing Newton's iteration for approximating the whitening matrix $\Sigma^{-\frac{1}{2}}$. An similar idea was also used in [80] by coupled Newton-Schulz iterations [82] for whitening. One interesting property of IterNorm is that it stretches the dimensions along the eigenvectors progressively, so that the associated eigenvalues converge to 1 after normalization. Therefore, IterNorm can effectively control the extent of whitening by its iteration number.

There also exist works that impose extra penalties on the loss function to obtain approximately whitened activations [83], [84], [85], [86], [87], or exploit the whitening operation to improve the network's generalization [88], [89].

Variations of Standardization: There are several variations of the standardization operation for normalizing the activations. As an alternative to the L^2 normalization used to control the activation scale in a BN layer [8], the L^1 normalization was proposed in [90], [91], [92] for standardization. Specifically, the dimension-wise standardization deviation of L^1 normalization is: $\sigma = \frac{1}{m}\sum_{i=1}^{m}|x^{(i)} - u|$. Note that L^2 normalization is made equivalent to L^1 normalization (under mild assumptions) by multiplying it by a scaling factor $\sqrt{\frac{\pi}{2}}$ [91], [92].

L^1 normalization can improve numerical stability in a low-precision implementation, as well as provide computational and memory benefits, over L^2 normalization. The merits of L^1 normalization originate from the fact that it avoids the costly square and root operations of L^2 normalization. Specifically, Wu *et al.* [91] showed that the proposed sign and absolute operations in L^1 normalization can achieve a $1.5\times$ speedup and reduce the power consumption by 50% on an FPGA platform. Similar merits also exist when using L^∞, as discussed in [92], where the standardization deviation is: $\sigma = \max_i |x^{(i)}|$. The more generalized L^p was investigated in [90] and [92], where the standardization deviation is: $\sigma = \frac{1}{m}\sqrt[p]{\sum_{i=1}^{m}(x^{(i)})^p}$.

Yuan *et al.* [93] proposed generalized batch normalization (GBN), in which the mean statistics for centering and the deviation measures for scaling are more general operations, the choice of which can be guided by the risk theory. They provided some optional asymmetric deviation measures for networks with ReLU, such as, the Right Semi-Deviation (RSD) [93].

Reduced Standardization: As stated in Section 2, the standardizing operation usually includes centering and scaling. However, some works only consider one or the other, for specific situations. Note that either centering or scaling along the batch dimension can benefit optimization, as shown in [8], [31]. Besides, the scaling operation is important for scale-invariant learning, and has been shown useful for adaptively adjusting the learning rate to stabilize training [94].

Salimans *et al.* proposed mean-only batch normalization (MoBN) [21], which only performs centering along the batch dimension, and works well when combined with weight normalization [21]. Yan *et al.* [95] proposed to perform scaling only in BN for small-batch-size training, which also works well when combined with weight centralization. Shen *et al.* [96] also proposed to perform scaling only in BN to improve the performance for NLP tasks.

Karras *et al.* [72] proposed pixel normalization, where the scaling only operation is performed along the channel dimension for each position of each image. This works like PN [71] but only uses the scaling operation. Pixel normalization works well for GANs [71] when used in the generator.

Zhang *et al.* [97] hypothesized that the re-centering invariance produced by centering in LN [20] is dispensable and proposed to perform scaling only for LN, which is referred to as root mean square layer normalization (RMSLN). RMSLN takes into account the importance of the scale-invariant property for LN. RMSLN works as well as LN on NLP tasks but reduces the running time [97]. This idea was also used in the variance-only layer normalization for the click-through rate (CTR) prediction task [98]. Chiley *et al.* [99] also proposed to perform scaling along the channel dimension

(like RMSLN) in the proposed online normalization to stabilize the training.

Singh and Krishnan proposed filter response normalization (FRN) [100], which performs the scaling-only operation along each channel (filter response) for each sample independently. This is similar to IN [67] but without performing the centering operation. The motivation is that the benefits of centering for normalization schemes that are batch independent are not really justified.

5.1.3 Normalization Representation Recovery

Normalization constrains the distribution of the activations, which can benefit optimization. However, these constraints may hamper the representation ability, so an additional affine transformation is usually used to recover the possible representation, as shown in Eqn. 22. There are also other options for constructing the NRR.

Siarohin *et al*. [79] proposed a coloring transformation to recover the possible loss in representation ability caused by the whitening operation, which is formulated as:

$$\widetilde{\boldsymbol{X}} = \Psi_{LR}(\widehat{\boldsymbol{X}}) = \widehat{\boldsymbol{X}}\mathbf{W} + (\beta \mathbf{1}^T), \tag{24}$$

where $\mathbf{W}$ is a $d \times d$ learnable matrix. The coloring transformation can be viewed as a linear layer in neural networks.

In Eqn. 22, the NRR parameters are both learnable through backpropagation. Several works have attempted to generalize these parameters by using a hypernetwork to dynamically generate them, which is formulated as:

$$\widetilde{\boldsymbol{X}} = \Psi_{DC}(\widehat{\boldsymbol{X}}) = \widehat{\boldsymbol{X}} \odot \Gamma_{\phi^\gamma} + B_{\phi^\beta}, \tag{25}$$

where $\Gamma_{\phi^\gamma} \in \mathbb{R}^{d\times m}$ and $B_{\phi^\beta} \in \mathbb{R}^{d\times m}$ are generated by the subnetworks $\phi^\gamma_{\theta_\gamma}(\cdot)$ and $\phi^\beta_{\theta_\beta}(\cdot)$, respectively. The affine parameters generated depend on the original inputs themselves, making them different from the affine parameters shown in Eqn. 22, which are learnable by backpropagation.

Kim *et al*. proposed dynamic layer normalization (DLN) [101] in a long short-term memory (LSTM) architecture for speech recognition, where $\phi^\gamma_{\theta_\gamma}(\cdot)$ and $\phi^\beta_{\theta_\beta}(\cdot)$ are separate utterance-level feature extractor subnetworks, which are jointly trained with the main acoustic model. The input of the subnetworks is the output of the corresponding hidden layer of the LSTM. Similar ideas are also used in adaptive instance normalization (AdaIN) [69] and adaptive layer-instance normalization (AdaLIN) [102] for unsupervised image-to-image translation, where the subnetworks are MLPs and the inputs of the subnetworks are the embedding features produced by one encoder. Jia *et al*. [103] proposed instance-level meta normalization (ILMN), which utilizes an encoder-decoder subnetwork to generate affine parameters, given the instance-level mean and variance as input. Besides, ILMN also combines the learnable affine parameters shown in Eqn. 22. Rather than using the channel-wise affine parameters shared across spatial positions, spatially adaptive denormalization (SPADE) uses the spatially dependent $\beta, \gamma \in \mathbb{R}^{d\times h\times w}$, which are dynamically generated by a two-layer CNN with raw images as inputs.

The mechanism for generating affine parameter Γ_{ϕ^γ} shown in Eqn. 25 resembles the squeeze-excitation (SE) block [104], when the input of the subnetwork $\phi^\gamma_{\theta_\gamma}(\cdot)$ is $\boldsymbol{X}$ itself, *i.e*., $\Gamma_{\phi^\gamma} = \phi^\gamma_{\theta_\gamma}(\boldsymbol{X})$. Inspired by this, Liang *et al*. proposed instance enhancement batch normalization (IEBN)) [105], which combines the channel-wise affine parameters in Eqn. 22 and the instance-specific channel-wise affine parameters in Eqn. 25 using SE-like subnetworks, with fewer parameters. IEBN can effectively regulate noise by introducing instance-specific information for BN. This idea is further generalized by attentive normalization [106], proposed by Li *et al*., where the affine parameters are modeled by K mixture components.

Rather than using a subnetwork to generate the affine parameters, Xu *et al*. proposed adaptive normalization (AdaNorm) [27], where the affine parameters depend on the standardized output $\widehat{\boldsymbol{X}}$ of layer normalization:

$$\widetilde{\boldsymbol{X}} = \widehat{\boldsymbol{X}} \odot \phi(\widehat{\boldsymbol{X}}). \tag{26}$$

Here, $\phi(\widehat{\boldsymbol{X}})$ used in [27] is: $\phi(\hat{\mathbf{x}}) = C(1 - k\hat{\mathbf{x}})$, $C = \frac{1}{d}\sum_{i=1}^{d}\hat{\mathbf{x}}_i$ and k is a constant that satisfies certain constraints. Note that $\phi(\widehat{\boldsymbol{X}})$ is treated as a changing constant (not a function) and the gradient of $\phi(\widehat{\boldsymbol{X}})$ is detached in the implementation [27]. We also note that the side information can be injected into the NRR operation for conditional generative models. The typical works are conditional BN (CBN) [107] and conditional IN (CIN) [68]. We will elaborate on these in Section 9 when we discuss applications of normalization.

In summary, we list the main single-mode normalization methods under our proposed framework in Table 1.

5.2 Multi-Mode and Combinational Normalization

In previous sections, we focused on single-mode normalization methods. In this section, we will introduce the methods that extend to multiple modes, as well as combinational methods.

Multiple Modes: Kalayeh and Shah proposed mixture normalizing (MixNorm) [110], which performs normalization on subregions that can be identified by disentangling the different modes of the distribution, estimated via a Gaussian mixture model (GMM). MixNorm requires a two-stage process, where the GMM is first fitted by expectation-maximization (EM) [111] with K-means++ [112] for initialization, and the normalization is then performed on samples with respect to the estimated parameters. MixNorm is not fully differentiable due to the K-means++ and EM iterations.

Deecke *et al*. proposed mode normalization (ModeNorm), which also extends the normalization to more than one mean and variance to address the heterogeneous nature of complex datasets. MN is formulated in a mixture of experts (MoE) framework, where a set of simple gate functions is introduced to assign one example to groups with a given probability. Each sample in the mini-batch is then normalized under voting from its gate assignment. The gate functions are trained jointly by backpropagation.

Combination: Since different normalization strategies have different advantages and disadvantages for training DNNs, some methods try to combine them. Luo *et al*. proposed switchable normalization (SN) [113], which combines three types of statistics, estimated channel-wise, layer-wise, and mini-batch-wise, by using IN, LN, and BN, respectively. SN switches between the different normalization methods by learning their importance weights, computed by a softmax function. SN was designed to address the learning-to-normalize problem and obtains good results on several visual benchmarks [113]. Shao *et al*. [114] further introduced sparse switchable normalization (SSN), which selects different normalizations using the proposed SparsestMax function, which is a sparse version of softmax. Pan *et al*. [115] proposed switchable whitening (SW), which provides a general way to switch between different whitening and standardization methods under the SN

TABLE 1
Summary of the main single-mode normalization methods, based on our proposed framework for describing normalizing-activations-as-functions methods. The order is based on the time of publication.

Method	NAP	NOP	NRR	Published In
Local contrast normalization [108]	$\Pi_{IN}(\mathbf{X}) \in \mathbb{R}^{md\times hw}$: local spatial positions	Standardizing	No	ICCV, 2009
Local response normalization [75]	$\Pi_{PN}(\mathbf{X}) \in \mathbb{R}^{mhw\times d}$: local channels	Scaling	No	NeurIPS, 2012
Batch normalization (BN) [8]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d\times mhw}$	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	ICML, 2015
Mean-only BN [21]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d\times mhw}$	Centering	No	NeurIPS, 2016
Layer normalization (LN) [20]	$\Pi_{LN}(\mathbf{X}) \in \mathbb{R}^{m\times dhw}$	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	Arxiv, 2016
Instance normalization (IN) [67]	$\Pi_{IN}(\mathbf{X}) \in \mathbb{R}^{md\times hw}$	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	Arxiv, 2016
L^p-Norm BN [90]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d\times mhw}$	Standardizing with L^p-Norm divided	Learnable $\gamma, \beta \in \mathbb{R}^d$	Arxiv, 2016
Divisive normalization [76]	$\Pi_{LN}(\mathbf{X}) \in \mathbb{R}^{m\times dhw}$: local spatial positions and channels	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	ICLR, 2017
Conditional IN [68]	$\Pi_{IN}(\mathbf{X}) \in \mathbb{R}^{md\times hw}$	Standardizing	Side information	ICLR, 2017
Dynamic LN [101]	$\Pi_{LN}(\mathbf{X}) \in \mathbb{R}^{m\times dhw}$	Standardizing	Generated $\gamma, \beta \in \mathbb{R}^d$	INTERSPEECH, 2017
Conditional BN [107]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d\times mhw}$	Standardizing	Side information	NeurIPS, 2017
Pixel normalization [72]	$\Pi_{PN}(\mathbf{X}) \in \mathbb{R}^{mhw\times d}$	Scaling	No	ICLR, 2018
Decorrelated BN [34]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d\times mhw}$	ZCA whitening	Learnable $\gamma, \beta \in \mathbb{R}^d$	CVPR, 2018
Group normalization (GN) [22]	$\Pi_{GN}(\mathbf{X}) \in \mathbb{R}^{mg\times shw}$	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	ECCV, 2018
Adaptive IN [69]	$\Pi_{IN}(\mathbf{X}) \in \mathbb{R}^{md\times hw}$	Standardizing	Generated $\gamma, \beta \in \mathbb{R}^d$	ECCV, 2018
L^1-Norm BN [91], [92]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d\times mhw}$	Standardizing with L^1-Norm divided	Learnable $\gamma, \beta \in \mathbb{R}^d$	NeurIPS, 2018
Whitening and coloring BN [79]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d\times mhw}$	CD whitening	Color transformation	ICLR, 2019
Generalized BN [93]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d\times mhw}$	General standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	AAAI, 2019
Iterative normalization [81]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d\times mhw}$	ZCA whitening by Newton's iteration	Learnable $\gamma, \beta \in \mathbb{R}^d$	CVPR, 2019
Instance-level meta normalization [103]	$\Pi_{LN}(\mathbf{X})/\Pi_{IN}(\mathbf{X})/\ \Pi_{GN}(\mathbf{X})$	Standardizing	Learnable & generated $\gamma, \beta \in \mathbb{R}^d$	CVPR, 2019
Spatially adaptive denormalization [109]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d\times mhw}$	Standardizing	Generated $\gamma, \beta \in \mathbb{R}^{d\times h\times w}$	CVPR, 2019
Position normalization (PN) [71]	$\Pi_{PN}(\mathbf{X}) \in \mathbb{R}^{mhw\times d}$	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	NeurIPS, 2019
Root mean square LN [97]	$\Pi_{LN}(\mathbf{X}) \in \mathbb{R}^{m\times dhw}$	Scaling	Learnable $\gamma \in \mathbb{R}^d$	NeurIPS, 2019
Online normalization [99]	$\Pi_{LN}(\mathbf{X}) \in \mathbb{R}^{m\times dhw}$	Scaling	No	NeurIPS, 2019
Batch group normalization [73]	$\Pi_{BGN}(\mathbf{X}) \in \mathbb{R}^{g_m g\times s_m shw}$	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	ICLR, 2020
Instance enhancement BN [105]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d\times mhw}$	Standardizing	Learnable & generated $\gamma, \beta \in \mathbb{R}^d$	AAAI, 2020
PowerNorm [96]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d\times mhw}$	Scaling	Learnable $\gamma, \beta \in \mathbb{R}^d$	ICML, 2020
Local contex normalization [74]	$\Pi_{GN}(\mathbf{X}) \in \mathbb{R}^{mg\times shw}$: local spatial positions and channels	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	CVPR, 2020
Filter response normalization [100]	$\Pi_{IN}(\mathbf{X}) \in \mathbb{R}^{md\times hw}$	Scaling	Learnable $\gamma, \beta \in \mathbb{R}^d$	CVPR, 2020
Attentive normalization [106]	$\Pi_{BN}(\mathbf{X})/\Pi_{IN}(\mathbf{X})/$ $\Pi_{LN}(\mathbf{X})/\Pi_{GN}(\mathbf{X})$	Standardizing	Generated $\gamma, \beta \in \mathbb{R}^d$	ECCV, 2020

framework. Zhang *et al*. [116] introduced exemplar normalization (EN) to investigate a dynamic 'learning-to-normalize' problem. EN learns different data-dependent normalizations for different image samples, while SN fixes the importance ratios for the entire dataset. Besides, Luo *et al*. [117] proposed dynamic normalization (DN), which generalizes IN, LN, GN and BN in a unified formulation and can interpolate them to produce new normalization methods.

Considering that IN can learn style-invariant features [67], Nam *et al*. [118] introduced batch-instance normalization (BIN) to normalize the styles adaptively to the task and selectively to individual feature maps. It learns to control how much of the style information is propagated through each channel by leveraging a learnable gate parameter to balance between IN and BN. A similar idea was also used in the adaptive layer-instance normalization (AdaLIN) [102] for image-to-image translation tasks, where a learnable gate parameter is leveraged to balance between LN and IN. Bronskill *et al*. [119] introduced TaskNorm, which combines LN/IN with BN for meta-learning scenarios. Rather than designing a combinational normalization module, Pan *et al*. proposed IBN-Net, which carefully integrates IN and BN as building blocks, and can be wrapped into several deep networks to improve their performances. Qiao *et al*. introduced batch-channel normalization (BCN), which integrates BN and channel-based normalizations (e.g., LN and GN) sequentially as a wrapped module.

Recently, Liu *et al*. [120] searched for a combination of normalization-activation layers using AutoML [121], leading to the discovery of EvoNorms, a set of new normalization-activation layers with sometimes surprising structures that go beyond existing

design patterns.

5.3 BN for More Robust Estimation

As illustrated in previous sections, BN introduces inconsistent normalization operations during training (using mini-batch statistics, as shown in Eqn. 17) and inference (using population statistics estimated in Eqn. 18). This means that the upper layers are trained on representations different from those computed during inference. These differences become significant if the batch size is too small, since the estimates of the mean and variance become less accurate. This leads to significantly degenerated performance [22], [122], [123], [124]. To address this problem, some normalization methods avoid normalizing along the batch dimension, as introduced in previous sections. Here, we will discuss the more robust estimation methods that also address this problem of BN.

5.3.1 Normalization as Functions Combining Population Statistics

One way to reduce the discrepancy between training and inference is to combine the estimated population statistics for normalization during training.

Ioffe *et al*. [122] proposed batch renormalization (BReNorm), augmenting the normalized output for each neuron with an affine transform, as:

$$\hat{x} = \frac{x-\mu}{\sigma} \cdot r + z, \tag{27}$$

where $r = \frac{\sigma}{\hat{\sigma}}$ and $z = \frac{\mu-\hat{\mu}}{\hat{\sigma}}$. Note that r and z are bounded between $(\frac{1}{r_{max}}, r_{max})$ and $(\frac{1}{z_{max}}, z_{max})$, respectively. Besides, r and z are treated as constants when performing gradient computation. Eqn. 27 is reduced to standardizing the activation using the estimated population (which ensures that the training and inference are consistent) if r and z are between their bounded values. Otherwise, Eqn. 27 implicitly exploits the benefits of mini-batch statistics.

Dinh *et al*. [125] were the first to experiment with batch normalization using population statistics, which were weighted averages of the old population statistics and current mini-batch statistics, as shown in Eqn. 18. The experimental results demonstrate that, combining population and mini-batch statistics can improve the performance of BN in small-batch-size scenarios. This idea is also used in diminishing batch normalization [126], full normalization (FN) [127], online normalization [99], moving average batch normalization (MABN) [95], PowerNorm [96] and momentum batch normalization (MBN) [128]. One challenge for this type of method is how to calculate the gradients during backpropagation, since the population statistics are computed by all the previous mini-batches, and it is impossible to obtain their exact gradients [90]. One straightforward strategy is to view the population statistics as constant and only back-propagate through current mini-batches, as proposed in [125], [126] and [127]. However, this may introduce training instability, as discussed in Section 4. Chiley *et al*. [99] proposed to compute the gradients by maintaining the property of BN during backpropagation. Yan *et al*. [95] and Shen *et al*. [96] proposed to view the backpropagation gradients as statistics to be estimated, and approximate these statistics by moving averages.

Rather than explicitly using the population statistics, Guo *et al*. [129] introduced memorized BN, which considers data information from multiple recent batches (or all batches in an extreme case) to produce more accurate and stable statistics. A similar idea is exploited in cross-iteration batch normalization [130], where the mean and variance of examples from recent iterations are approximated for the current network weights via a low-order Taylor polynomial. Besides, Wang *et al*. proposed Kalman normalization [131], which treats all the layers in a network as a whole system, and estimates the statistics of a certain layer by considering the distributions of all its preceding layers, mimicking the merits of Kalman filtering. Another practical approach for relieving the small-batch-size issue of BN in engineering systems is the synchronized batch normalization [132], [133], [134], which performs a synchronized computation of BN statistics across GPUs (Cross-GPU BN) to obtain better statistics.

5.3.2 Robust Inference Methods for BN

Some works address the small-batch-size problem of BN by finely estimating corrected normalization statistics during inference only. This strategy does not affect the training scheme of the model.

In fact, even the original BN paper [8] recommended estimating the population statistics after the training has finished (Algorithm 2 in [8]), rather than using the estimation calculated by running average, as shown in Eqn. 18. However, while this can benefit a model trained with a small batch size, where estimation is the main issue [8], [113], [135], it may lead to degenerated generalization when the batch size is moderate.

Singh *et al*. analyzed how a small batch size hampers the estimation accuracies of BN when using running averages, and proposed EvalNorm [123], which optimizes the sample weight during inference to ensure that the activations produced by normalization are similar to those provided during training. A similar idea is also exploited in [73], where the sample weights are viewed as hyperparameters, which are optimized on a validation set.

Compared to estimating the BN statistics (population mean and standardization deviation), Huang *et al*. [35] showed that estimating the whitening matrix of BW is more challenging. They demonstrated that, in terms of estimating the population statistics of the whitening matrix, it is more stable to use the mini-batch covariance matrix indirectly (the whitening matrix can be calculated after training) than the mini-batch whitening matrix directly.

6 Normalizing Weights

As stated in Section 3, normalizing the weights can implicitly normalize the activations by imposing constraints on the weight matrix, which can contribute to preserving the activations (gradients) during forward (backpropagation). Several seminal works have analyzed the distributions of the activations, given normalized inputs, under the assumption that the weights have certain properties or are under certain constraints, *e.g*., normalization propagation [136], variance propagation [137], self normalization [138], bidirectional self-normalization [139]. The general idea of weight normalization is to provide layer-wise constraints on the weights during optimization, which can be formulated as:

$$\begin{aligned} \theta^* &= \arg\min_\theta \mathbb{E}_{(\mathbf{x},\mathbf{y})\in D}[\mathcal{L}(\mathbf{y}, f(\mathbf{x};\theta))] \\ &s.t. \quad \Upsilon(\boldsymbol{W}), \end{aligned} \tag{28}$$

where $\Upsilon(\boldsymbol{W})$ are the layer-wise constraints imposed on the weights $\boldsymbol{W} \in \mathbb{R}^{d_{out}\times d_{in}}$. It has been shown that the imposed constraints can benefit generalization [51], [52], [140]. In the following sections, we will introduce different constraints and discuss how to train a model with the constraints satisfied.

6.1 Constraints on Weights

Salimans *et al*. proposed weight normalization (WN), which requires the input weight of each neuron to be unit norm. Specifically, given one neuron's input weight $\boldsymbol{W}_i \in \mathbb{R}^{d_{in}}$, the constraints imposed on $\boldsymbol{W}$ are:

$$\Upsilon(\boldsymbol{W}) = \{\|\boldsymbol{W}_i\| = 1, i = 1, ..., d_{out}\}. \tag{29}$$

Weight normalization has a scale-invariant property like BN, which is important for stabilizing training.

Inspired by the practical weight initialization technique [3], [44], where weights are sampled from a distribution with zero mean and a standard deviation for initialization, Huang *et al*. [51] further proposed centered weight normalization (CWN), constraining the input weight of each neuron to have zero mean and unit norm, as:

$$\Upsilon(\boldsymbol{W}) = \{\boldsymbol{W}_i^T\mathbf{1} = 0 \ \& \ \|\boldsymbol{W}_i\| = 1, i = 1, ..., d_{out}\}. \tag{30}$$

CWN can theoretically preserve the activation statistics between different layers under certain assumptions (Proposition 1 in [51]), which can benefit optimization. Weight centering is also advocated for in [95], [141]. Qiao *et al*. proposed weight standardization (WS), which imposes the constraints on the weights [142] with $\Upsilon(\boldsymbol{W}) = \{\boldsymbol{W}_i^T\mathbf{1} = 0 \ \& \ \|\boldsymbol{W}_i\| = \sqrt{d_{out}}, i = 1, ..., d_{out}\}$. Note that WS cannot effectively preserve the activation statistics between different layers, since the weight norm is $\sqrt{d_{out}}$, which may cause exploding activations. Therefore, WS usually needs to be combined with activation normalization methods (*e.g*., BN/GN) to relieve this issue [141].

Another widely used constraint on weights is orthogonality, which is represented as

$$\Upsilon(\boldsymbol{W}) = \{\boldsymbol{W}\boldsymbol{W}^T = \boldsymbol{I}\}. \tag{31}$$

Orthogonality was first used in the square hidden-to-hidden weight matrices of RNNs [143], [144], [145], [146], [147], [148], [149], and then further extended to the more general rectangular matrices in DNNs [52], [150], [151], [152], [153]. Orthogonal weight matrices can theoretically preserve the norm of activations/output-gradients between linear transformations [24], [152], [153]. Further, the distributions of activations/output-gradients can also be preserved under mild assumptions [24], [52]. These properties of orthogonal weight matrices are beneficial for the optimization of DNNs. Furthermore, orthogonal weight matrices can avoid learning redundant filters, benefitting generalization.

Rather than bounding all singular values as 1, like in orthogonal weight matrices, Miyato *et al*. [23] proposed spectral normalization, which constrains the spectral norm (the maximum singular value) of a weight matrix to 1, in order to control the Lipschitz constant of the discriminator when training GANs. Huang *et al*. [24] proposed orthogonalization by Newton's iterations (ONI), which controls the orthogonality through the iteration number. They showed that it is possible to bound the singular values of a weight matrix between $(\sigma_{min}, 1)$ during training. ONI effectively interpolates between spectral normalization and full orthogonalization, by altering the iteration number.

Note that the constraints imposed on the weight matrix (Eqns. 29, 30, 31) may harm the representation capacity and result in degenerated performance. An extra learnable scalar parameter is usually used for each neuron to recover the possible loss in representation capacity, which is similar to the idea of the affine parameters proposed in BN.

6.2 Training with Constraints

It is clear that training a DNN with constraints imposed on the weights is a constraint optimization problem. Here, we summarize three kinds of strategies for solving this.

Re-Parameterization One stable way to solve constraint optimization problems is to use a re-parameterization method. Re-parameterization constructs a fine transformation ψ over the proxy parameter $\boldsymbol{V}$ to ensure that the transformed weight $\boldsymbol{W}$ has certain beneficial properties for the training of neural networks. Gradient updating is executed on the proxy parameter $\boldsymbol{V}$ by back-propagating the gradient information through the normalization process. Re-parameterization was first used in learning the square orthogonal weight matrices in RNNs [143], [144], [145]. Salimans *et al*. [21] used this technique to learn a unit-norm constraint as shown in Eqn. 29. Huang *et al*. [51] formally described the re-parameterization idea in training with constraints on weight matrices, and applied it to solve the optimization with zero-mean and unit-norm constraints as shown in Eqn. 30. This technique was then further used in other methods for learning with different constraints, *e.g*., orthogonal weight normalization [52], weight standardization [141], spectral normalization [23] and weight centralization [95]. Re-parameterization is a main technique for optimizing constrained weights in DNNs. Its main merit is that the training is relatively stable, because it updates $\boldsymbol{V}$ based on the gradients computed by backpropagation, while simultaneously maintaining the constraints on $\boldsymbol{W}$. The downside is that the backpropagation through the designed transformation may increase the computational cost.

Regularization with an Extra Penalty Some works have tried to maintain the weight constraints using an additional penalty on the objective function, which can be viewed as a regularization. This regularization technique is mainly used for learning the weight matrices with orthogonality constraints, for its efficiency in computation [4], [16], [146], [154], [155]. Orthogonal regularization methods have demonstrated improved performance in image classification [16], [154], [156], [157], resisting attacks from adversarial examples [158], neural photo editing [159] and training GANs [23], [30]. However, the introduced penalty works like a pure regularization, and whether or not the constraints are truly maintained or training benefited is unclear. Besides, orthogonal regularization usually requires to be combined with activation normalization, when applied on large-scale architectures, since it cannot stabilize training.

Riemannian Optimization A weight matrix $\boldsymbol{W}$ with constraints can be viewed as an embedded submanifold [52], [160], [161], [162]. For example, a weight matrix with an orthogonality constraint (Eqn.31) is a real Stiefel manifold [52], [160]. One possible way of maintaining these constraints when training DNNs is to use Riemannian optimization [163], [164]. Conventional Riemannian optimization techniques are based on a gradient descent method over a manifold, which iteratively seeks updated points. In each iteration, there are two main phases: 1) Computing the Riemannian gradient based on the inner dot product defined in the tangent space of the manifold; 2) Finding the descent direction and ensuring that the new point is on the manifold [52], [164]. Most Riemannian optimization methods in the deep learning community address the second phase by either QR-decomposition-type retraction [164], [165] or Cayley transformation [144], [146], [162], [166]. The main difficulties of applying Riemannian optimization in training

DNNs are: 1) The optimization space covers multiple embedded submanifolds; 2) The embedded submanifolds are inter-dependent since the optimization of the current weight layer is affected by those of preceding layers. To stabilize the training, activation normalizations (*e.g*. BN) [162] or gradient clips [160] are usually required. One interesting observation is that using BN will probably improve the performance of projection-based methods (where the gradient is calculated based on the Euclidean space) [161], [167].

6.3 Combining Activation Normalization

Normalizing weights has its own advantages in training DNNs, compared to normalizing activations. For example, it is data-independent, and is more convenient to use for theoretical analysis [168], [169]. However, normalizing weights has some drawbacks when used in large-scale networks in practice: 1) It may not effectively improve the optimization efficiency when residual connections are introduced or when the nonlinearity does not satisfy the assumptions required for preserving distributions between different layers, since the Criteria 1 in Section 3 is not readily satisfied in these situations; 2) Normalizing the weight usually has significantly lower test accuracy than BN on large-scale image classification [170]. Huang *et al*. showed that CWN combined with BN can improve the original networks with only BN. The idea of combining normalizing weights and activations to improve performance has been widely studied [24], [52], [141], [171]. Moreover, Luo *et al*. proposed cosine normalization [172], which merges layer normalization and weight normalization together.

7 Normalizing Gradients

As stated previously, normalizing activations and weights aims to provide a better optimization landscape for DNNs, by satisfying Criteria 1 and 2 in Section 3. Rather than providing a good optimization landscape by design, normalizing gradients in DNNs aims to exploit the curvature information for GD/SGD, even though the optimization landscape is ill-conditioned. It performs normalization solely on the gradients, which may effectively remove the negative effects of an ill-conditioned landscape caused by the diversity in magnitude of gradients from different layers [3]. Generally speaking, normalizing gradients is similar to second-order optimization [173], [174], [175], [176] or coordinate-wise adaptive learning rate based methods [7], [177], [178], but with the goal of exploiting the layer-wise structural information in DNNs.

Yu *et al*. [53] were the first to propose block-wise (layer-wise) gradient normalization for training DNNs to front the gradient explosion or vanishing problem. Specifically, they perform scaling over the gradients w.r.t. the weight in each layer, ensuring the norm to be unit-norm. This technique can decrease the magnitude of a large gradient to a certain level, like gradient clipping [4], and also boost the magnitude of a small gradient. However, the net-gain of this approach degenerates in the scale-invariant DNNs (*e.g*. with BN). In [53], an extra ratio factor that depends on the norm of the layer-wise weight was used to adaptively adjust the magnitude of the gradients. A similar idea was also introduced in the layer-wise adaptive rate scaling (LARS), proposed by You *et al*. [54], for large-batch training. LARS and its follow-up works [179], [180], [181] are essential techniques in training large-scale DNNs using large batch sizes, significantly reducing the training times without degradation of performance.

Rather than using the scaling operation, Yong *et al*. [182] recently proposed gradient centralization (GC), which performs centering over the gradient w.r.t. the input weight of each neuron in each layer. GC implicitly imposes constraints on the input weight, and ensures that the sum of elements in the input weight is a constant during training. GC effectively improves the performances of DNNs with activation normalization (*e.g*. BN or GN).

8 Analysis of Normalization

In Section 3, we provided high-level motivation of normalization in benefiting network optimization. In this section, we will further discuss other properties of normalization methods in improving DNNs' training performance. We mainly focus on BN, because it displays nearly all the benefits of normalization in improving the performance of DNNs, *e.g*., stabilizing training, accelerating convergence and improving the generalization.

8.1 Scale Invariance in Stabilizing Training

One essential functionality of BN is its ability to stabilize training. This is mainly due to its scale-invariant property [20], [183], [184], [185], *i.e*., it does not change the prediction when rescaling parameters and works by adaptively adjusting the learning rate in a layer-wise manner [92], [94], [160]. Specifically,

$$BN(\mathbf{x}; a\boldsymbol{W}) = BN(\mathbf{x}; \boldsymbol{W}) \tag{32}$$

$$\frac{\partial BN(\mathbf{x}; a\boldsymbol{W})}{\partial (a\boldsymbol{W})} = \frac{1}{a}\frac{\partial BN(\mathbf{x}; \boldsymbol{W})}{\partial \boldsymbol{W}}, \tag{33}$$

where a is a constant factor. The scale-invariant property also applies to other methods that normalize the activations [20], [22], [34], [67] or weights [21], [51], [52]. This property was first shown in the original BN paper, and then investigated in [20] to compare different normalization methods, and further extended for rectifier networks in [43]. Specifically, in [43], Huang *et al*. showed how the scaled factor a of the weight in a certain layer will lead to exponentially increased/decreased gradients for unnormalized rectifier networks, and how normalization can avoid this problem with its scale-invariant property.

The scale-invariant weight vector in a network is always perpendicular to its gradient [21], [94], [183], [184], [186], which has an auto-tuning effect [91], [94], [187], [188]. Wu *et al*. proposed WNgrad [169], which introduces an adaptive step-size algorithm based on this fact. In [94], Arora *et al*. proved that GD/SGD with BN can arrive a first-order stationary point with any fixed learning rate, under certain mild assumptions. Cai *et al*. [187] showed that, for the simple problem of ordinary least squares (OLS), GD with BN converges under arbitrary learning rates for the weights, and the convergence remains linear under mild conditions.

Another research direction is to analyze the effect of weight decay [189] when combined with scale-invariant normalization methods [92], [142], [161], [185], [190], [191], [192]. In this case, weight decay causes the parameters to have smaller norms, and thus the effective learning rate is larger. In [192], Li and Arora showed that the original learning rate schedule and weight decay can be folded into a new exponential schedule, when scale-invariant normalization methods are used.

8.2 Improved Conditioning in Optimization

As stated in previous sections, one motivation behind BN is that whitening the input can improve the conditioning of the optimization [8] and thus accelerate training [34], [50]. This motivation is theoretically supported for linear models [36], [192],

but is difficult to further extend to DNNs. In [193], Santurkar *et al*. argue that BN may improve optimization by enhancing the smoothness of the Hessian of the loss. However, this conclusion is based on a layer-wise analysis [43], [193], which corresponds to the diagonal blocks of the overall Hessian. Ghorbani *et al*. [194] further empirically investigated the conditioning of the optimization problem by computing the spectrum of the Hessian for a large-scale dataset. It is believed that the improved conditioning enables large learning rates for training, thus improving the generalization, as shown in [195]. Karakida *et al*. [196] investigated the conditioning of the optimization problem by analyzing the geometry of the parameter space determined by the Fisher information matrix (FIM), which also corresponds to the local shape of the loss landscape under certain conditions.

One intriguing phenomenon is that the theoretical benefits of whitening the input for optimization only hold when BN is placed before the linear layer, while, in practice, BN is typically placed after the linear layer, as recommended in [8]. In [43], Huang *et al*. experimentally observed, through a layer-wise conditioning analysis, that BN (placed after the linear layer) not only improves the conditioning of the activation's covariance matrix, but also improves the conditioning of the output-gradient's covariation. Similar observations were made in [197], where BN prevents the rank collapse of pre-activation matrices. Some works have also empirically investigated the position, at which BN should be plugged in [34], [46], [198]. Results have shown that placing it after the linear layer may work better, in certain situations.

Other analyses of normalization in optimization include an investigation into the signal propagation and gradient backpropagation [199], [200], [201], based on the mean field theory [199], [200], [202]. Besides, the work of [203] demonstrated that BN obtains an accelerated convergence on the (possibly) nonconvex problem of learning half-spaces with Gaussian inputs, from a length-direction decoupling perspective. Dukler *et al*. [204] further provided the first global convergence result for two-layer neural networks with ReLU [6] activations trained with weight normalization.

8.3 Stochasticity for Generalization

One important property of BN is its ability to improve the generalization of DNNs. It is believed such an improvement is obtained from the stochasticity/noise introduced by normalization over batch data [8], [105], [205]. It is clear that both the normalized output (Eqn.17) and the population statistics (Eqn.18) can be viewed as stochastic variables, because they depend on the mini-batch inputs, which are sampled over datasets. Therefore, the stochasticity comes from the normalized output during training [81], and the discrepancy of normalization between training (using estimated population statistics) and inference (using estimated population statistics) [35], [206].

Ioffe and Szegedy [8] were the first to show the advantages of this stochasticity for the generalization of networks, like dropout [207], [208]. Teye *et al*. [209] demonstrated that training a DNN using BN is equivalent to approximating inference in Bayesian models, and that uncertainty estimates can be obtained from any network using BN through Monte Carlo sampling during inference. This idea was further efficiently approximated by stochastic batch normalization [210] and exploited in prediction-time batch normalization [211]. Alexander *et al*. jointly formulate the stochasticity of the normalized output and the discrepancy of normalization between training and inference in a mathematical way, under the assumptions that the distribution of activations over the full dataset is approximately Gaussian and *i.i.d*. In [81], Huang *et al*. proposed an empirical evaluation for the stochasticity of normalization over batch data, called stochastic normalization disturbance (SND), and investigated how the batch size affects the stochasticity of BN. This empirical analysis was further extended to the more general BW in [35].

Some studies exploit the stochasticity of BN to improve the generalization for large-batch training, by altering the batch size when estimating the population statistics. One typical work is ghost batch normalization [73], [212], [213], which reduces the generalization error by acquiring the statistics on small virtual ('ghost') batches instead of the real large batch.

9 Applications of Normalization

As previously stated, normalization methods can be wrapped as general modules, which have been extensively integrated into various DNNs to stabilize and accelerate training, probably leading to improved generalization. For example, BN is an essential module in the state-of-the-art network architectures for CV tasks [9], [11], [12], [15], [16], [17], and LN is an essential module in NLP tasks [25], [26], [27]. In this section, we discuss the applications of normalization for particular tasks, in which normalization methods can effectively solve the key issues. To be specific, we mainly review the applications of normalization in domain adaptation, style transfer, training GANs and efficient deep models. However, we note that there also exist works exploring how to apply normalization to meta learning [59], [214], [215], reinforcement learning [216], [217], [218], unsupervised representation learning [219], [220], permutation-equivariant networks [221], [222], graph neural networks [223], ordinary differential equation (ODE) based networks [224], symmetric positive definite (SPD) neural networks [225], and guarding against adversarial attacks [226], [227], [228].

9.1 Domain Adaptation

Machine learning algorithms trained on some given data (source domain) usually perform poorly when tested on data acquired under different settings (target domain). This is explained in domain adaptation as resulting from a shift between the distributions of the source and target domains. Most methods for domain adaptation thus aim to bridge the gap between these distributions. A typical way of achieving this is to align the distributions of the source and target domains based on the mini-batch/population statistics of BNs [229].

Li *et al*. [229] proposed the first work to exploit BNs in domain adaptation, named adaptive batch normalization (AdaBN), where the BN statistics for the source domain are calculated during training, and then those for the target domain are modulated during testing. AdaBN enables domain-invariant features to be learnt without requiring additional loss terms and the extra associated parameters. The hypothesis behind AdaBN is that the domain-invariant information is stored in the weight matrix of each layer, while the domain-specific information is represented by the statistics of the BN layer. However, this hypothesis may not always hold because the target domain is not exploited at the training stage. As a result, it is difficult to ensure that the statistics of the BN layers in the source and target domains correspond to their domain-specific information.

One way to overcome this limitation is to couple the network parameters for both target and source samples in the training stage, which has been the main research focus of several follow-up works inspired by AdaBN. In [230], Carlucci *et al*. proposed automatic domain alignment layers (AutoDIAL), which are embedded in different levels of the deep architecture to align the learned source and target feature distributions to a canonical one. AutoDIAL exploits the source and target features during the training stage, in which an extra parameter is involved in each BN layer as the trade-off between the source and target domains. Chang *et al*. further proposed domain-specific batch normalization (DSBN) [231], where multiple branches of BN are used, each of which is exclusively in charge of a single domain. DSBN learns domain-specific properties using only the estimated population statistics of BN and learns domain-invariant representations with the other parameters in the network. This method effectively separates domain-specific information for unsupervised domain adaptation. Similar ideas have also been exploited in unsupervised adversarial domain adaptation in the context of semantic scene segmentation [232] and adversarial examples for improving image recognition [233]. In [234], Roy *et al*. further generalized DSBN by a domain-specific whitening transform (DWT), where the source and target data distributions are aligned using their covariance matrices. Wang *et al*. [235] proposed transferable normalization (TransNorm), which also calculates the statistics of inputs from the source and target domains separately, while computing the channel transferability simultaneously. The normalized features then go through channel-adaptive mechanisms to re-weight the channels according to their transferability.

Besides population statistics, Seo *et al*. also exploited the affine transform (Eqn.22) of BN to represent the domain-specific information in their proposed domain-specific optimized normalization (DSON) [236]. Moreover, DSON normalizes the activations by a weighted average of multiple normalization statistics (typically BN and IN), and keeps track of the normalization statistics of each normalization type if necessary, for each domain. DSON targets domain generalization, where examples in the target domain cannot be accessed during training. This task is considered to be more challenging than unsupervised domain adaptation.

9.1.1 Learning Universal Representations

The idea of applying BN in domain adaptation can be further extended to the learning of universal representations [237], by constructing neural networks that work simultaneously in many domains. To achieve this, the networks need to learn to share common visual structures where no obvious commonality exists. Universal representations cannot only benefit domain adaptation but also contribute to multi-task learning, which aims to learn multiple tasks simultaneously in the same data domain.

Bilen *et al*. [237] advocate to learn universal image representations using 1) the convolutional kernels to extract domain-agnostic information and 2) the BN layers to transform the internal representations to the relevant target domains. Data *et al*. [238] exploited BN layers to learn discriminate visual classes, while other layers (*e.g*. convolutional layers) are used to learn the universal representation. They also provided a way to interpolate BN layers to solve new tasks. Li *et al*. [239] proposed covariance normalization (CovNorm) for multi-domain learning, which provides efficient solutions to several tasks defined in different domains.

9.2 Style Transfer

Style transfer is an important image editing task that enables the creation of new artistic works [240], [241]. Image style transform algorithms aim to generate a stylized image that has similar content and style to the given images. The key challenge in this task is to extract effective representations that can disentangle the style from the content. The seminal work by Gatys *et al*. [242] showed that the covariance/Gram matrix of the layer activations, extracted by a trained DNN, has a remarkable capacity for capturing visual styles. This provides a feasible solution to matching the styles between images by minimizing Gram matrix based losses, pioneering the way for style transfer.

A key advantage of applying normalization to style transfer is that the normalization operation (NOP) can remove the style information (*e.g*., whitening can ensure the covariance matrix to be an identity matrix), while the normalization representation recovery (NRR), in contrast, introduces it. In other words, the style information is intuitively 'editable' by normalization [71], [243]. In a seminal work, Ulyanov *et al*. proposed instance normalization (IN) [67] to remove instance-specific contrast information (style) from the content image. Since then, IN has been a basic module for image style transfer tasks.

In [68], Dumoulin *et al*. proposed conditional instance normalization (CIN), an efficient solution to integrating multiple styles. Specifically, multiple distinct styles are captured by a single network, by encoding the style information in the affine parameters (Eqn.22) of IN layers, after which each style can be selectively applied to a target image. Huang *et al*. [70] proposed adaptive instance normalization (AdaIN), where the activations of content images are standardized by their statistics, and the affine parameters (β and γ) come from the statistics of style activations. AdaIN transfers the channel-wise mean and variance feature statistics between content and style feature activations. AdaIN can also work well in text effect transfer, which aims at learning the visual effects while maintaining the text content [105]. Rather than manually defining how to compute the affine parameters so as to align the mean and variance between content and style features, dynamic instance normalization (DIN) [244], introduced by Jing *et al*., deals with arbitrary style transfer by encoding a style image into learnable convolution parameters, upon which the content image is stylized.

To address the limitations of AdaIN in only trying to match up the variances of the stylized image and the style image feature, Li *et al*. [240] further proposed whitening and coloring transformations (WCT) to match up the covariance matrix. This shares a similar spirit to the optimization of the Gram matrix based cost in neural style transfer [240], [245]. Some methods [246] also seek to provide a good trade-off between AdaIN (which enjoys higher computational efficiency) and WCT (which synthesizes images visually closer to a given style).

9.2.1 Image Translation

In computer vision, image translation can be viewed as a more general case of image style transfer. Given an image in the source domain, the aim is to learn the conditional distribution of the corresponding images in the target domain. This includes, but is not limited to, the following tasks: super-resolution, colorization, inpainting, and attribute transfer. Similar to style transfer, AdaIN is also an essential tool for image translation used in, for example, multimodal unsupervised image-to-image translation (MUIT) [69]. Note that the affine parameters of AdaIN in [69] are produced by a

learned network, instead of computed from statistics of a pretrained network as in [70]. Apart from IN, Cho *et al*. [247] proposed the group-wise deep whitening-and-coloring transformation (GDWCT) by matching higher-order statistics, such as covariance, for image-to-image translation tasks. Moreover, since the whitening/coloring transformation can be considered a 1×1 convolution, Cho *et al*. [248] further proposed adaptive convolution-based normalization (AdaCoN) to inject the target style into a given image, for unsupervised image-to-image translation. AdaCoN first performs standardization locally on each subregion of an input activation map (similar to the local normalization shown in Section 5.1.1) and then applies an adaptive convolution, where the convolution filter weights are dynamically estimated using the encoded style representation. Besides, Yu *et al*. proposed region normalization (RN) [249] for image inpainting network training. RN divides spatial pixels into different regions according to the input mask and standardizes the activations in each region. Wang [250] introduced attentive normalization (AN) for conditional image generation, which is an extension of instance normalization [67]. AN divides the feature maps into different regions based on their semantics, and then separately normalizes and denormalizes the feature points in the same region.

9.3 Training GANs

GANs [251] can be regarded as a general framework to produce a model distribution that mimics a given target distribution. A GAN consists of a generator, which produces the model distribution, and a discriminator, which distinguishes the model distribution from the target. From this perspective, the ultimate goal when training GANs shares a similar spirit to model training in the domain adaptation task. The main difference lies in that GANs try to reduce the distance between different distributions, while domain adaptation models attempt to close the gap between different domains. Therefore, the techniques that apply BN to domain adaptation, as discussed in Section 9.1, may work for GANs as well. For example, combining samples form different domains in a batch for BN may harm the generalization in domain adaptation, and this also applies for the training of GANs [62], [252].

One persisting challenge in training GANs is the performance control of the discriminator and the learning pace control between the discriminator and generator [23]. The density ratio estimated by the discriminator is often inaccurate and unstable during training, and the generator may fail to learn the structure of the target distribution. One way to remedy this issue is to impose constraints on the discriminator [253]. For instance, Xiang *et al*. [254] leveraged weight normalization to effectively improve the training performance of GANs. Miyato *et al*. [23] proposed spectral normalization (SN), which enforces Lipschitz continuity on the discriminator by normalizing its parameters with the spectral norm estimated by power iteration. Since then, SN has become an important technique in training GANs [23], [29], [30]. In [255], Zhang *et al*. further found that employing SN in the generator improves the stability, allowing for fewer training steps for the discriminator per iteration. Another important constraint in training GANs is the orthogonality [24], [30], [256], [257]. Brock *et al*. [30] found that applying orthogonal regularization to the generator renders it amenable to a simple 'truncation trick', allowing fine control over the trade-off between sample fidelity and variety by reducing the variance of the generator input. Huang *et al*. [24] proposed orthogonalization by Newton's iteration, which can effectively control the orthogonality of the weight matrix, and interpolate between spectral normalization and full orthogonalization by altering the iteration number.

As discussed in Section 9.2 for style transfer, the NRR operation of activation normalization can also be used as the side information for GANs, under the scenario of conditional GANs (cGANs) [258]. cGANs have shown advancements in class conditional image generation [259], image generation from text [260], [261], and image-to-image translation [262].

In [107], Vries *et al*. proposed conditional batch normalization (CBN), which injects a linguistic input (e.g., a question in a VQA task) into the affine parameters of BN. This shares a similar spirit to the conditional instance normalization for style transfer, and has been extensively explored in [30], [255], [263], [264]. In [265], Karras *et al*. proposed a style-based generator architecture for GANs, where the style information is embedded into the affine parameters of AdaIN [70]. Note that the style comes from the latent vector instead of an example image, enabling the model to work without external information. Similarly, Chen *et al*. [266] proposed a more general self-modulation based on CBN, where the affine parameters can also be generated by the generators own input or provided by external information.

9.4 Efficient Deep Models

In real-world applications, it is essential to consider the efficiency of an algorithm in addition to its effectiveness due to the often limited computational resources (such as in smartphones). As such, there is also an active line of research exploiting normalization techniques (*e.g*., BN) to develop efficient DNNs based on network slimming or quantization. In network slimming, the general idea is to exploit the channel-wise scale parameter $\gamma \in \mathbb{R}^d$ of BN, considering that each scale γ_i corresponds to a specific convolutional channel (or a neuron in a fully connected layer) [267]. For example, Liu *et al*. [267] proposed to identify and prune insignificant channels (or neurons) based on the scale parameter in BN layers, which are imposed by L^1 regularization for sparsity. Ye *et al*. [268] also adopted a similar idea, and developed a new algorithmic approach and rescaling trick to improve the robustness and speed of optimization. In [269], Li *et al*. proposed an efficient evaluation component based on adaptive batch normalization [229], which has a strong correlation between different pruned DNN structures and their final settled accuracy.

In [270], Yu *et al*. trained a slimmable network with a new variant of BN, namely switchable batch normalization (SBN), for the networks executable at different widths. SBN privatizes BN for different switches of a slimmable network, and each individual BN has independently accumulated feature statistics. SBN can thus be used as a general solution to obtain a good trade-off between accuracy and latency on the fly. As a complement to BN that normalizes the final summation of the weighted inputs, Luo *et al*. [271] proposed fine-grained batch normalization (FBN) to build light-weight networks, where FBN normalizes the intermediate state of the summation.

Network quantization is another essential technique in building efficient DNNs. This challenging task can also be tackled using normalization algorithms like BN. For instance, Banner *et al*. [272] proposed range batch normalization (RBN) for quantized networks, normalizing activations according to the range of the activation distribution. RBN avoids the sum of squares, square-root and reciprocal operations and is more friendly for low-precise training [273].

Lin *et al*. [274] proposed to quantize BN in model deployment by converting the two floating points affine transformations to a fixed-point operation with shared quantized scale. Ardakani *et al*. [275] employed BN to train binarized/ternarized LSTMs, and achieved state-of-the-art performance in network quantization. Hou *et al*. [276] further studied and compared the quantized LSTMs with WN, LN and BN. They showed that these normalization methods make the gradient invariant to weight scaling, thus alleviating the problem of having a potentially large weight norm increase due to quantization. In [277], Sari *et al*. analyzed how the centering and scaling operations in BN affect the training of binary neural networks.

10 Summary and Discussion

In this paper, we have provided a research landscape for normalization techniques, covering methods, analyses and applications. We believe that our work can provide valuable guidelines for selecting normalization techniques to use in training DNNs. With the help of these guidelines, it will be possible to design new normalization methods tailored to specific tasks (by the choice of NAP) or improve the trade-off between efficiency and performance (by the choice of NOP). We leave the following open problems for discussion.

Theoretical Perspective: While the practical success of DNNs is indisputable, their theoretical analysis is still limited. Despite the recent progress of deep learning in terms of representation [278], optimization [279] and generalization [280], the networks investigated theoretically are usually different from those used in practice [199]. One clear example is that, while normalization techniques are ubiquitously used in the current state-of-the-art architectures, the theoretical analyses for DNNs usually rule out them.

In fact, the methods commonly used for normalizing activations (*e.g*., BN, LN) often conflict with current theoretical analyses. For instance, in the representation of DNNs, one important strategy is to analyze the number of linear regions, where the expressivity of a DNN with rectifier nonlinearity can be quantified by the maximal number of linear regions it can separate its input space into [278], [281]. However, this generally does not hold if BN/LN are introduced, since they create nonlinearity, causing the the theoretical assumptions to no longer be met. It is thus important to further investigate how BN/LN affect a model's representation capacity. As for optimization, most analyses require the input data to be independent, such that the stochastic/mini-batch gradient is an unbiased estimator of the true gradient over the dataset. However, BN typically does not fit this data-independent assumption, and its optimization usually depends on the sampling strategy as well as the mini-batch size [127]. There is thus a need to reformulate the current theoretical framework for optimization when BN is present.

In contrast, normalizing-weights methods do not harm the theoretic analysis of DNNs, and can even attribute to boosting the theoretical results. For example, the Lipschitz constant w.r.t. a linear layer can be controlled/bounded during training by normalizing the weight with (approximate) orthogonality [23], [24], which is an important property for certified defense against adversarial attacks [282], [283], [284], and for theoretically analyzing DNNs generalization [285], [286]. However, normalizing weights is still not as effective as normalizing activations when it comes to improving training performance, leaving room for further development.

Applications Perspective: As mentioned previously, normalization methods can be used to 'edit' the statistical properties of layer activations, which has been exploited in CV tasks to match particular domain knowledges. However, we note that this mechanism is seldom used in NLP tasks. It would thus be interesting to investigate the correlation between the statistical properties of layer activations and the domain knowledge in NLP, and further improve the performance of the corresponding tasks. In addition, There exists an intriguing phenomenon that, while BN/GN work for the CV models, LN is more effective in NLP [96]. Intuitively, BN/GN should work well for NLP tasks, considering that the current state-of-the-art models for CV and NLP tend to be similar (*e.g*., they both use the convolutional operation and attention) and GN is simply a more general version of LN. It is thus important to further investigate whether or not BN/GN can be made to work well for NLP tasks, and, if not, why.

Another interesting observation is that normalization is not very common in deep reinforcement learning (DRL) [63]. Considering that certain DRL frameworks (*e.g*., actor-critic [287], [288]) are very similar to GANs, it should be possible to exploit normalization techniques to improve training in DRL, borrowing ideas from GANs (*e.g*., normalizing the weights in the discriminator [23], [24], [30]).

As the key components in DNNs, normalization techniques are links that connect the theory and application of deep learning. We thus believe that these techniques will continue to have a profound impact on the rapidly growing field of deep learning, and we hope that this paper will aid readers in building a comprehensive landscape for their implementation.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

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

[3] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010.

[4] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *ICML*, 2013.

[5] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[6] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

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

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[11] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *BMVC*, 2016.

[12] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *ECCV*, 2016.

[14] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *CVPR*, 2017.

[15] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017.

[16] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *CVPR*, 2017.

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

[18] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *ECCV*, 2014, pp. 740–755.

[19] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

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

[21] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *NeurIPS*, 2016.

[22] Y. Wu and K. He, "Group normalization," in *ECCV*, 2018.

[23] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *ICLR*, 2018.

[24] L. Huang, L. Liu, F. Zhu, D. Wan, Z. Yuan, B. Li, and L. Shao, "Controllable orthogonalization in training dnns," in *CVPR*, 2020.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017.

[26] A. W. Yu, D. Dohan, M.-T. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le, "Qanet: Combining local convolution with global self-attention for reading comprehension," in *ICLR*, 2018.

[27] J. Xu, X. Sun, Z. Zhang, G. Zhao, and J. Lin, "Understanding and improving layer normalization," in *NeurIPS*, 2019.

[28] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T.-Y. Liu, "On layer normalization in the transformer architecture," in *ICML*, 2020.

[29] K. Kurach, M. Lučić, X. Zhai, M. Michalski, and S. Gelly, "A large-scale study on regularization and normalization in GANs," in *ICML*, 2019.

[30] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," in *ICLR*, 2019.

[31] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Muller, "Efficient backprop," in *Neural Networks: Tricks of the Trade*, 1998.

[32] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[33] A. Kessy, A. Lewin, and K. Strimmer, "Optimal whitening and decorrelation," *The American Statistician*, vol. 72, no. 4, pp. 309–314, 2018.

[34] L. Huang, D. Yang, B. Lang, and J. Deng, "Decorrelated batch normalization," in *CVPR*, 2018.

[35] L. Huang, L. Zhao, Y. Zhou, F. Zhu, L. Liu, and L. Shao, "An investigation into the stochasticity of batch whitening," in *CVPR*, 2020.

[36] Y. LeCun, I. Kanter, and S. A. Solla, "Second order properties of error surfaces," in *NeurIPS*, 1990.

[37] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018.

[38] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," in *ICML*, 2015.

[39] J. Ba, R. Grosse, and J. Martens, "Distributed second-order optimization using kronecker-factored approximations," in *ICLR*, 2017.

[40] K. Sun and F. Nielsen, "Relative Fisher information and natural gradient for learning large modular models," in *ICML*, 2017.

[41] A. Bernacchia, M. Lengyel, and G. Hennequin, "Exact natural gradient in deep linear networks and its application to the nonlinear case," in *NeurIPS*, 2018.

[42] J. Martens, "New perspectives on the natural gradient method," *arXiv preprint arXiv:1412.1193*, 2014.

[43] L. Huang, J. Qin, L. Liu, F. Zhu, and L. Shao, "Layer-wise conditioning analysis in exploring the learning dynamics of dnns," in *ECCV*, 2020.

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *ICCV*, 2015.

[45] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," in *ICLR*, 2014.

[46] D. Mishkin and J. Matas, "All you need is a good init," in *ICLR*, 2016.

[47] P. A. Sokol and I. M. Park, "Information geometry of orthogonal initializations and training," in *ICLR*, 2020.

[48] G. Montavon and K.-R. Müller, *Deep Boltzmann Machines and the Centering Trick*, 2012, vol. 7700.

[49] S. Wiesler, A. Richard, R. Schlüter, and H. Ney, "Mean-normalized stochastic gradient for large-scale deep learning," in *ICASSP*, 2014.

[50] G. Desjardins, K. Simonyan, R. Pascanu, and k. kavukcuoglu, "Natural neural networks," in *NeurIPS*, 2015.

[51] L. Huang, X. Liu, Y. Liu, B. Lang, and D. Tao, "Centered weight normalization in accelerating training of deep neural networks," in *ICCV*, 2017.

[52] L. Huang, X. Liu, B. Lang, A. W. Yu, Y. Wang, and B. Li, "Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks," in *AAAI*, 2018.

[53] A. W. Yu, L. Huang, Q. Lin, R. Salakhutdinov, and J. Carbonell, "Block-normalized gradient method: An empirical study for training deep neural network," *arXiv preprint arXiv:1707.04822*, 2017.

[54] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," *arXiv preprint arXiv:1708.03888*, 2017.

[55] N. N. Schraudolph, "Accelerated gradient descent by factor-centering decomposition," Tech. Rep., 1998.

[56] T. Raiko, H. Valpola, and Y. LeCun, "Deep learning made easier by linear transformations in perceptrons," in *AISTATS*, 2012.

[57] P. Luo, "Learning deep architectures via generalized whitened neural networks," in *ICML*, 2017.

[58] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[59] J. Bronskill, J. Gordon, J. Requeima, S. Nowozin, and R. E. Turner, "Tasknorm: Rethinking batch normalization for meta-learning," in *ICML*, 2020.

[60] T. Cooijmans, N. Ballas, C. Laurent, and A. C. Courville, "Recurrent batch normalization," in *ICLR*, 2017.

[61] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, "Batch normalized recurrent neural networks," in *ICASSP*, 2016.

[62] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen, "Improved techniques for training gans," in *NeurIPS*, 2016.

[63] A. Bhatt, M. Argus, A. Amiranashvili, and T. Brox, "Crossnorm: Normalization for off-policy td reinforcement learning," *arXiv preprint arXiv:1902.05605*, 2019.

[64] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NeurIPS Autodiff Workshop*, 2017.

[65] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016.

[66] Ç. Gülçehre and Y. Bengio, "Knowledge matters: Importance of prior information for optimization," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 226–257, 2016.

[67] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016.

[68] V. Dumoulin, J. Shlens, and M. Kudlur, "A learned representation for artistic style," in *ICLR*, 2017.

[69] X. Huang, M. Liu, S. J. Belongie, and J. Kautz, "Multimodal unsupervised image-to-image translation," in *ECCV*, 2018.

[70] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *ICCV*, 2017.

[71] B. Li, F. Wu, K. Q. Weinberger, and S. Belongie, "Positional normalization," in *NeurIPS*, 2019.

[72] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation," in *ICLR*, 2018.

[73] C. Summers and M. J. Dinneen, "Four things everyone should know to improve batch normalization," in *ICLR*, 2020.

[74] A. Ortiz, C. Robinson, M. Hassan, D. Morris, O. Fuentes, C. Kiekintveld, and N. Jojic, "Local context normalization: Revisiting local normalization," *arXiv preprint arXiv:1912.05845*, 2019.

[75] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012, pp. 1097–1105.

[76] M. Ren, R. Liao, R. Urtasun, F. H. Sinz, and R. S. Zemel, "Normalizing the normalizers: Comparing and extending network normalization schemes," in *ICLR*, 2017.

[77] Siwei Lyu and E. P. Simoncelli, "Nonlinear image representation using divisive normalization," in *CVPR*, 2008.

[78] C. Ionescu, O. Vantzos, and C. Sminchisescu, "Training deep networks with structured layers by matrix backpropagation," in *ICCV*, 2015.

[79] A. Siarohin, E. Sangineto, and N. Sebe, "Whitening and coloring transform for gans," in *ICLR*, 2019.

[80] C. Ye, M. Evanusa, H. He, A. Mitrokhin, T. Goldstein, J. A. Yorke, C. Fermuller, and Y. Aloimonos, "Network deconvolution," in *ICLR*, 2020.

[81] L. Huang, Y. Zhou, F. Zhu, L. Liu, and L. Shao, "Iterative normalization: Beyond standardization towards efficient whitening," in *CVPR*, 2019.

[82] N. J. Higham, *Functions of matrices: theory and computation*. SIAM, 2008.

[83] M. Cogswell, F. Ahmed, R. B. Girshick, L. Zitnick, and D. Batra, "Reducing overfitting in deep networks by decorrelating representations," in *ICLR*, 2016.

[84] W. Xiong, B. Du, L. Zhang, R. Hu, and D. Tao, "Regularizing deep convolutional neural networks with a structured decorrelation constraint," in *ICDM*, 2016.

[85] E. Littwin and L. Wolf, "Regularizing by the variance of the activations' sample-variances," in *NeurIPS*, 2018.

[86] T. Joo, D. Kang, and B. Kim, "Regularizing activations in neural networks via distribution matching with the wasserstein metric," in *ICLR*, 2020.

[87] W. Zhou, B. Y. Lin, and X. Ren, "Isobn: Fine-tuning bert with isotropic batch normalization," *arXiv preprint arXiv:2005.02178*, 2020.

[88] W. Shao, S. Tang, X. Pan, P. Tan, X. Wang, and P. Luo, "Channel equilibrium networks for learning deep representation," in *ICML*, 2020.

[89] Z. Chen, Y. Bei, and C. Rudin, "Concept whitening for interpretable image recognition," *arXiv preprint arXiv:2002.01650*, 2020.

[90] Q. Liao, K. Kawaguchi, and T. Poggio, "Streaming normalization: Towards simpler and more biologically-plausible normalizations for online and recurrent learning," *arXiv preprint arXiv:1610.06160*, 2016.

[91] S. Wu, G. Li, L. Deng, L. Liu, Y. Xie, and L. Shi, "L1-norm batch normalization for efficient training of deep neural networks," *arXiv preprint arXiv:1802.09769*, 2018.

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

[93] X. Yuan, Z. Feng, M. Norton, and X. Li, "Generalized batch normalization: Towards accelerating deep neural networks," in *AAAI*, 2019.

[94] S. Arora, Z. Li, and K. Lyu, "Theoretical analysis of auto rate-tuning by batch normalization," in *ICLR*, 2019.

[95] J. Yan, R. Wan, X. Zhang, W. Zhang, Y. Wei, and J. Sun, "Towards stabilizing batch statistics in backward propagation of batch normalization," in *ICLR*, 2020.

[96] S. Shen, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Powernorm: Rethinking batch normalization in transformers," in *ICML*, 2020.

[97] B. Zhang and R. Sennrich, "Root mean square layer normalization," in *NeurIPS*, 2019.

[98] Z. Wang, Q. She, P. Zhang, and J. Zhang, "Correct normalization matters: Understanding the effect of normalization on deep neural network models for click-through rate prediction," *arXiv preprint arXiv:2006.12753*, 2020.

[99] V. Chiley, I. Sharapov, A. Kosson, U. Koster, R. Reece, S. Samaniego de la Fuente, V. Subbiah, and M. James, "Online normalization for training neural networks," in *NeurIPS*, 2019.

[100] S. Singh and S. Krishnan, "Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks," in *CVPR*, 2020.

[101] T. Kim, I. Song, and Y. Bengio, "Dynamic layer normalization for adaptive neural acoustic modeling in speech recognition," in *INTERSPEECH*, 2017, pp. 2411–2415.

[102] J. Kim, M. Kim, H. Kang, and K. H. Lee, "U-gat-it: Unsupervised generative attentional networks with adaptive layer-instance normalization for image-to-image translation," in *ICLR*, 2020.

[103] S. Jia, D. Chen, and H. Chen, "Instance-level meta normalization," in *CVPR*, 2019.

[104] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *CVPR*, 2018.

[105] S. Liang, Z. Huang, M. Liang, and H. Yang, "Instance enhancement batch normalization: an adaptive regulator of batch noise," in *AAAI*, 2020.

[106] X. Li, W. Sun, and T. Wu, "Attentive normalization," *ArXiv*, vol. arXiv preprint arXiv:1908.01259, 2019.

[107] H. de Vries, F. Strub, J. Mary, H. Larochelle, O. Pietquin, and A. C. Courville, "Modulating early visual processing by language," in *NeurIPS*, 2017, pp. 6594–6604.

[108] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *ICCV*, 2009.

[109] T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu, "Semantic image synthesis with spatially-adaptive normalization," in *CVPR*, 2019.

[110] M. M. Kalayeh and M. Shah, "Training faster by separating modes of variation in batch-normalized models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[111] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, vol. 39, no. 1, pp. 1–38, 1977.

[112] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[113] P. Luo, J. Ren, Z. Peng, R. Zhang, and J. Li, "Differentiable learning-to-normalize via switchable normalization," in *ICLR*, 2019.

[114] W. Shao, T. Meng, J. Li, R. Zhang, Y. Li, X. Wang, and P. Luo, "Ssn: Learning sparse switchable normalization via sparsestmax," in *CVPR*, 2019.

[115] X. Pan, X. Zhan, J. Shi, X. Tang, and P. Luo, "Switchable whitening for deep representation learning," in *ICCV*, 2019.

[116] R. Zhang, Z. Peng, L. Wu, Z. Li, and P. Luo, "Exemplar normalization for learning deep representation," in *CVPR*, 2020.

[117] P. Luo, P. Zhanglin, S. Wenqi, Z. Ruimao, R. Jiamin, and W. Lingyun, "Differentiable dynamic normalization for learning deep representation," in *ICML*, 2019, pp. 4203–4211.

[118] H. Nam and H.-E. Kim, "Batch-instance normalization for adaptively style-invariant neural networks," in *NeurIPS*, 2018.

[119] J. Bronskill, J. Gordon, J. Requeima, S. Nowozin, and R. E. Turner, "Tasknorm: Rethinking batch normalization for meta-learning," in *ICML*, 2020.

[120] H. Liu, A. Brock, K. Simonyan, and Q. V. Le, "Evolving normalization-activation layers," *arXiv preprint arXiv:2004.02967*, 2020.

[121] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *ICLR*, 2017.

[122] S. Ioffe, "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models," in *NeurIPS*, 2017.

[123] S. Singh and A. Shrivastava, "Evalnorm: Estimating batch normalization statistics for evaluation," in *ICCV*, 2019.

[124] A. Kaku, S. Mohan, A. Parnandi, H. Schambra, and C. Fernandez-Granda, "Be like water: Robustness to extraneous variables via adaptive feature normalization," *arXiv preprint arXiv:2002.04019*, 2020.

[125] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," in *ICLR*, 2017.

[126] Y. Ma and D. Klabjan, "Convergence analysis of batch normalization for deep neural nets," *arXiv preprint arXiv:1705.08011*, 2017.

[127] X. Lian and J. Liu, "Revisit batch normalization: New understanding and refinement via composition optimization," in *AISTATS*, 2019.

[128] H. Yong, J. Huang, D. Meng, X. Hua, and L. Zhang, "Momentum batch normalization for deep learning with small batch size," in *ECCV*, 2020.

[129] Y. Guo, Q. Wu, C. Deng, J. Chen, and M. Tan, "Double forward propagation for memorized batch normalization," in *AAAI*, 2018.

[130] Z. Yao, Y. Cao, S. Zheng, G. Huang, and S. Lin, "Cross-iteration batch normalization," *arXiv preprint arXiv:2002.05712*, 2020.

[131] G. Wang, J. Peng, P. Luo, X. Wang, and L. Lin, "Kalman normalization: Normalizing internal representations across network layers," in *NeurIPS*, 2018.

[132] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *CVPR*, 2017.

[133] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *CVPR*, 2018.

[134] C. Peng, T. Xiao, Z. Li, Y. Jiang, X. Zhang, K. Jia, G. Yu, and J. Sun, "Megdet: A large mini-batch object detector," in *CVPR*, 2018.

[135] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, "Averaging weights leads to wider optima and better generalization," *arXiv preprint arXiv:1803.05407*, 2018.

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

[137] A. Shekhovtsov and B. Flach, "Normalization of neural networks using analytic variance propagation," in *Computer Vision Winter Workshop*, 2018.

[138] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *NeurIPS*, 2017.

[139] Y. Lu, S. Gould, and T. Ajanthan, "Bidirectional self-normalizing neural networks," *arXiv preprint arXiv:2006.12169*, 2020.

[140] M. Özay and T. Okatani, "Training cnns with normalized kernels," in *AAAI*, 2018.

[141] S. Qiao, H. Wang, C. Liu, W. Shen, and A. Yuille, "Weight standardization," *arXiv preprint arXiv:1903.10520*, 2019.

[142] X. Li, S. Chen, and J. Yang, "Understanding the disharmony between weight normalization family and weight decay." in *AAAI*, 2020.

[143] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," in *ICML*, 2016.

[144] S. Wisdom, T. Powers, J. Hershey, J. Le Roux, and L. Atlas, "Full-capacity unitary recurrent neural networks," in *NeurIPS*, 2016.

[145] V. Dorobantu, P. A. Stromhaug, and J. Renteria, "Dizzyrnn: Reparameterizing recurrent neural networks for norm-preserving backpropagation," *arXiv preprint arXiv:1612.04035*, 2016.

[146] E. Vorontsov, C. Trabelsi, S. Kadoury, and C. Pal, "On orthogonality and learning recurrent networks with long term dependencies," in *ICML*, 2017.

[147] S. Hyland and G. Rtsch, "Learning unitary operators with help from u(n)," in *AAAI*, 2017.

[148] L. Jing, Ç. Gülçehre, J. Peurifoy, Y. Shen, M. Tegmark, M. Soljacic, and Y. Bengio, "Gated orthogonal recurrent units: On learning to forget," *arXiv preprint arXiv:1706.02761*, 2017.

[149] K. Helfrich, D. Willmott, and Q. Ye, "Orthogonal recurrent neural networks with scaled Cayley transform," in *ICML*, 2018.

[150] M. Ozay, "Fine-grained optimization of deep neural networks," in *NeurIPS*, 2019.

[151] K. Jia, S. Li, Y. Wen, T. Liu, and D. Tao, "Orthogonal deep neural networks," *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[152] J. Wang, Y. Chen, R. Chakraborty, and S. X. Yu, "Orthogonal convolutional neural networks," in *CVPR*, 2020.

[153] H. Qi, C. You, X. Wang, Y. Ma, and J. Malik, "Deep isometric learning for visual recognition," in *ICML*, 2020.

[154] N. Bansal, X. Chen, and Z. Wang, "Can we gain more from orthogonality regularizations in training deep cnns?" in *NeurIPS*, 2018.

[155] J. Amjad, Z. Lyu, and M. R. Rodrigues, "Deep learning for inverse problems: Bounds and regularizers," *arXiv preprint arXiv:1901.11352*, 2019.

[156] L. Zhang, M. Edraki, and G.-J. Qi, "Cappronet: Deep feature learning via orthogonal projections onto capsule subspaces," in *NeurIPS*, 2018.

[157] J. Lezama, Q. Qiu, P. Mus, and G. Sapiro, "Ol: Orthogonal low-rank embedding - a plug and play geometric loss for deep learning," in *CVPR*, 2018.

[158] C. Moustapha, B. Piotr, E. Grave, Y. Dauphin, and N. Usunie, "Parseval networks: Improving robustness to adversarial examples," in *ICML*, 2017.

[159] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Neural photo editing with introspective adversarial networks," in *ICLR*, 2017.

[160] M. Cho and J. Lee, "Riemannian approach to batch normalization," in *NeurIPS*, 2017.

[161] L. Huang, X. Liu, B. Lang, and B. Li, "Projection based weight normalization for deep neural networks," *arXiv preprint arXiv:1710.02338*, 2017.

[162] J. Li, L. Fuxin, and S. Todorovic, "Efficient riemannian optimization on the stiefel manifold via the cayley transform," in *ICLR*, 2020.

[163] M. Ozay and T. Okatani, "Optimization on submanifolds of convolution kernels in cnns," *arXiv preprint arXiv:1610.07008*, 2016.

[164] M. Harandi and B. Fernando, "Generalized backpropagation, etude de cas: Orthogonality," *arXiv preprint arXiv:1611.05927*, 2016.

[165] T. Kaneko, S. G. O. Fiori, and T. Tanaka, "Empirical arithmetic averaging over the compact stiefel manifold," *IEEE Trans. Signal Processing*, vol. 61, no. 4, pp. 883–894, 2013.

[166] Z. Wen and W. Yin, "A feasible method for optimization with orthogonality constraints." *Math. Program.*, vol. 142, no. 1-2, pp. 397–434, 2013.

[167] K. Jia, "Improving training of deep neural networks via singular value bounding," in *CVPR*, 2017.

[168] B. Neyshabur, R. Tomioka, and N. Srebro, "Norm-based capacity control in neural networks," in *COLT*, 2015, pp. 1376–1401.

[169] X. Wu, R. Ward, and L. Bottou, "Wngrad: Learn the learning rate in gradient descent," *arXiv preprint arXiv:1803.02865*, 2018.

[170] I. Gitman and B. Ginsburg, "Comparison of batch normalization and weight normalization algorithms for the large-scale image classification," *arXiv preprint arXiv:1709.08145*, 2017.

[171] S. Qiao, H. Wang, C. Liu, W. Shen, and A. Yuille, "Rethinking normalization and elimination singularity in neural networks," *arXiv preprint arXiv:1911.09738*, 2019.

[172] C. Luo, J. Zhan, X. Xue, L. Wang, R. Ren, and Q. Yang, "Cosine normalization: Using cosine similarity instead of dot product in neural networks," in *ICANN*, 2018, pp. 382–391.

[173] J. Martens, "Deep learning via hessian-free optimization," in *ICML*. Omnipress, 2010, pp. 735–742.

[174] O. Vinyals and D. Povey, "Krylov subspace descent for deep learning," in *AISTATS*, 2012, pp. 1261–1268.

[175] J. Martens and I. Sutskever, "Training deep and recurrent networks with hessian-free optimization," in *Neural Networks: Tricks of the Trade (2nd ed.)*, ser. Lecture Notes in Computer Science, vol. 7700. Springer, 2012, pp. 479–535.

[176] R. B. Grosse and R. Salakhutdinov, "Scaling up natural gradient by sparsely factorizing the inverse fisher matrix," in *ICML*, 2015, pp. 2304–2313.

[177] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[178] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," 2012.

[179] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large batch optimization for deep learning: Training bert in 76 minutes," in *ICLR*, 2020.

[180] S. Zheng, H. Lin, S. Zha, and M. Li, "Accelerated large batch optimization of bert pretraining in 54 minutes," *arXiv preprint arXiv:2006.13484*, 2020.

[181] Z. Huo, B. Gu, and H. Huang, "Large batch training does not need warmup," *arXiv preprint arXiv:2002.01576*, 2020.

[182] H. Yong, J. Huang, X. Hua, and L. Zhang, "Gradient centralization: A new optimization technique for deep neural networks," in *ECCV*, 2020.

[183] B. Neyshabur, R. Tomioka, R. Salakhutdinov, and N. Srebro, "Data-dependent path normalization in neural networks," in *ICLR*, 2016.

[184] J. Sun, X. Cao, H. Liang, W. Huang, Z. Chen, and Z. Li, "New interpretations of normalization methods in deep learning," in *AAAI*, 2020.

[185] R. Wan, Z. Zhu, X. Zhang, and J. Sun, "Spherical motion dynamics of deep neural networks with batch normalization and weight decay," *arXiv preprint arXiv:2006.08419*, 2020.

[186] S. Roburin, Y. de Mont-Marin, A. Bursuc, R. Marlet, P. Pérez, and M. Aubry, "Spherical perspective on learning with batch norm," *arXiv preprint arXiv:2006.13382*, 2020.

[187] Y. Cai, Q. Li, and Z. Shen, "A quantitative analysis of the effect of batch normalization on gradient descent," in *ICML*, 2019, pp. 882–890.

[188] E. Chai, M. Pilanci, and B. Murmann, "Separating the effects of batch normalization on cnn training speed and stability using classical adaptive filter theory," *arXiv preprint arXiv:2002.10674*, 2020.

[189] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *NeurIPS*, 1992.

[190] T. Van Laarhoven, "L2 regularization versus batch and weight normalization," *arXiv preprint arXiv:1706.05350*, 2017.

[191] G. Zhang, C. Wang, B. Xu, and R. B. Grosse, "Three mechanisms of weight decay regularization," in *ICLR*, 2019.

[192] Z. Li and S. Arora, "An exponential learning rate schedule for batch normalized networks," in *ICLR*, 2020.

[193] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" in *NeurIPS*, 2018.

[194] B. Ghorbani, S. Krishnan, and Y. Xiao, "An investigation into neural net optimization via hessian eigenvalue density," in *ICML*, 2019.

[195] J. Bjorck, C. Gomes, and B. Selman, "Understanding batch normalization," in *NeurIPS*, 2018.

[196] R. Karakida, S. Akaho, and S.-i. Amari, "The normalization method for alleviating pathological sharpness in wide neural networks," in *NeurIPS*, 2019, pp. 6403–6413.

[197] H. Daneshmand, J. Kohler, F. Bach, T. Hofmann, and A. Lucchi, "Theoretical understanding of batch-normalization: A markov chain perspective," *arXiv preprint arXiv:2003.01652*, 2020.

[198] G. Chen, P. Chen, Y. Shi, C. Hsieh, B. Liao, and S. Zhang, "Rethinking the usage of batch normalization and dropout in the training of deep neural networks," *arXiv preprint arXiv:1905.05928*, 2019.

[199] G. Yang, J. Pennington, V. Rao, J. Sohl-Dickstein, and S. S. Schoenholz, "A mean field theory of batch normalization," in *ICLR*, 2019.

[200] M. Wei, J. Stokes, and D. J. Schwab, "Mean-field analysis of batch normalization," *arXiv preprint arXiv:1903.02606*, 2019.

[201] A. Labatie, "Characterizing well-behaved vs. pathological deep neural networks," in *ICML*, 2019, pp. 3611–3621.

[202] J. Lee, J. Sohl-dickstein, J. Pennington, R. Novak, S. Schoenholz, and Y. Bahri, "Deep neural networks as gaussian processes," in *ICLR*, 2018.

[203] J. Kohler, H. Daneshmand, A. Lucchi, T. Hofmann, M. Zhou, and K. Neymeyr, "Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization," in *AISTATS*, 2019.

[204] Y. Dukler, Q. Gu, and G. Montúfar, "Optimization theory for relu neural networks trained with normalization layers," in *ICML*, 2020.

[205] A. Shekhovtsov and B. Flach, "Stochastic normalizations as bayesian learning," in *ACCV*, 2018.

[206] P. Luo, X. Wang, W. Shao, and Z. Peng, "Towards understanding regularization in batch normalization," in *ICLR*, 2019.

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

[208] X. Li, S. Chen, X. Hu, and J. Yang, "Understanding the disharmony between dropout and batch normalization by variance shift," in *CVPR*, 2019.

[209] M. Teye, H. Azizpour, and K. Smith, "Bayesian uncertainty estimation for batch normalized deep networks," in *ICML*, 2018.

[210] A. Atanov, A. Ashukha, D. Molchanov, K. Neklyudov, and D. Vetrov, "Uncertainty estimation via stochastic batch normalization," in *ICLR Workshop*, 2018.

[211] Z. Nado, S. Padhy, D. Sculley, A. D'Amour, B. Lakshminarayanan, and J. Snoek, "Evaluating prediction-time batch normalization for robustness under covariate shift," *arXiv preprint arXiv:2006.10963*, 2020.

[212] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," in *NeurIPS*, 2017.

[213] N. Dimitriou and O. Arandjelovic, "A new look at ghost normalization," *arXiv preprint arXiv:2007.08554*, 2020.

[214] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv preprint arXiv:1803.02999*, 2018.

[215] J. Gordon, J. Bronskill, M. Bauer, S. Nowozin, and R. Turner, "Meta-learning probabilistic inference for prediction," in *ICLR*, 2019.

[216] H. P. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver, "Learning values across many orders of magnitude," in *NeurIPS*, 2016.

[217] A. Bhatt, M. Argus, A. Amiranashvili, and T. Brox, "Crossnorm: Normalization for off-policy td reinforcement learning," *arXiv preprint arXiv:1902.05605*, 2019.

[218] C. Wang, Y. Wu, Q. Vuong, and K. Ross, "Striving for simplicity and performance in off-policy drl: Output normalization and non-uniform sampling," in *ICML*, 2020.

[219] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *CVPR*, 2020.

[220] M. Taha Kocyigit, L. Sevilla-Lara, T. M. Hospedales, and H. Bilen, "Unsupervised batch normalization," in *CVPR Workshops*, 2020.

[221] K. Moo Yi, E. Trulls, Y. Ono, V. Lepetit, M. Salzmann, and P. Fua, "Learning to find good correspondences," in *CVPR*, 2018.

[222] W. Sun, W. Jiang, E. Trulls, A. Tagliasacchi, and K. M. Yi, "Attentive context normalization for robust permutation-equivariant learning," in *CVPR*, 2020.

[223] T. Cai, S. Luo, K. Xu, D. He, T.-y. Liu, and L. Wang, "Graphnorm: A principled approach to accelerating graph neural network training," *arXiv preprint arXiv:2009.03294*, 2020.

[224] J. Gusak, L. Markeeva, T. Daulbaev, A. Katrutsa, A. Cichocki, and I. Oseledets, "Towards understanding normalization in neural odes," *arXiv preprint arXiv:2004.09222*, 2020.

[225] D. Brooks, O. Schwander, F. Barbaresco, J.-Y. Schneider, and M. Cord, "Riemannian batch normalization for spd neural networks," in *NeurIPS*, 2019, pp. 15 463–15 474.

[226] A. Galloway, A. Golubeva, T. Tanay, M. Moussa, and G. W. Taylor, "Batch normalization is a cause of adversarial vulnerability," *arXiv preprint arXiv:1905.02161*, 2019.

[227] M. Awais, F. Shamshad, and S.-H. Bae, "Towards an adversarially robust normalization approach," *arXiv preprint arXiv:2006.11007*, 2020.

[228] C. Xie and A. Yuille, "Intriguing properties of adversarial training at scale," in *ICLR*, 2020.

[229] Y. Li, N. Wang, J. Shi, J. Liu, and X. Hou, "Revisiting batch normalization for practical domain adaptation," *arXiv preprint arXiv:1603.04779*, 2016.

[230] F. M. Carlucci, L. Porzi, B. Caputo, E. Ricci, and S. R. Bulo, "Autodial: Automatic domain alignment layers," in *ICCV*, 2017.

[231] W. Chang, T. You, S. Seo, S. Kwak, and B. Han, "Domain-specific batch normalization for unsupervised domain adaptation," in *CVPR*, 2019.

[232] R. Romijnders, P. Meletis, and G. Dubbelman, "A domain agnostic normalization layer for unsupervised adversarial domain adaptation," in *WACV*, 2019.

[233] C. Xie, M. Tan, B. Gong, J. Wang, A. Yuille, and Q. V. Le, "Adversarial examples improve image recognition," in *ICLR*, 2020.

[234] S. Roy, A. Siarohin, E. Sangineto, S. R. Bulo, N. Sebe, and E. Ricci, "Unsupervised domain adaptation using feature-whitening and consensus loss," in *CVPR*, 2019.

[235] X. Wang, Y. Jin, M. Long, J. Wang, and M. I. Jordan, "Transferable normalization: Towards improving transferability of deep neural networks," in *NeurIPS*, 2019.

[236] S. Seo, Y. Suh, D. Kim, J. Han, and B. Han, "Learning to optimize domain specific normalization for domain generalization," in *ECCV*, 2020.

[237] H. Bilen and A. Vedaldi, "Universal representations: The missing link between faces, text, planktons, and cat breeds," *arXiv preprint arXiv:1701.07275*, 2017.

[238] G. Wesley Putra Data, K. Ngu, D. William Murray, and V. Adrian Prisacariu, "Interpolating convolutional neural networks using batch normalization," in *ECCV*, 2018.

[239] Y. Li and N. Vasconcelos, "Efficient multi-domain learning by covariance normalization," in *CVPR*, 2019.

[240] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, "Universal style transfer via feature transforms," in *NeurIPS*, 2017.

[241] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song, "Neural style transfer: A review," *IEEE transactions on visualization and computer graphics*, 2019.

[242] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *CVPR*, 2016.

[243] B. Li, F. Wu, S.-N. Lim, S. Belongie, and K. Q. Weinberger, "On feature normalization and data augmentation," *arXiv preprint arXiv:2002.11102*, 2020.

[244] Y. Jing, X. Liu, Y. Ding, X. Wang, E. Ding, M. Song, and S. Wen, "Dynamic instance normalization for arbitrary style transfer," in *AAAI*, 2020.

[245] T.-Y. Chiu, "Understanding generalized whitening and coloring transform for universal style transfer," in *ICCV*, 2019.

[246] L. Sheng, Z. Lin, J. Shao, and X. Wang, "Avatar-net: Multi-scale zero-shot style transfer by feature decoration," in *CVPR*, 2018.

[247] W. Cho, S. Choi, D. K. Park, I. Shin, and J. Choo, "Image-to-image translation via group-wise deep whitening-and-coloring transformation," in *CVPR*, 2019.

[248] W. Cho, K. Kim, E. Kim, H. J. Kim, and J. Choo, "Unpaired image translation via adaptive convolution-based normalization," *arXiv preprint arXiv:1911.13271*, 2019.

[249] T. Yu, Z. Guo, X. Jin, S. Wu, Z. Chen, W. Li, Z. Zhang, and S. Liu, "Region normalization for image inpainting," in *AAAI*, 2020.

[250] Y. Wang, Y.-C. Chen, X. Zhang, J. Sun, and J. Jia, "Attentive normalization for conditional image generation," in *CVPR*, 2020.

[251] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NeurIPS*, 2014.

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

[253] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[254] S. Xiang and H. Li, "On the effects of batch and weight normalization in generative adversarial networks," *arXiv preprint arXiv:1704.03971*, 2017.

[255] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *ICML*, 2019.

[256] B. Liu, Y. Zhu, Z. Fu, G. de Melo, and A. Elgammal, "Oogan: Disentangling gan with one-hot sampling and orthogonal regularization." in *AAAI*, 2020.

[257] J. Müller, R. Klein, and M. Weinmann, "Orthogonal wasserstein gans," *arXiv preprint arXiv:1911.13060*, 2019.

[258] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[259] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier GANs," in *ICML*, 2017.

[260] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," in *ICML*, 2016.

[261] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. Metaxas, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," in *ICCV*, 2017.

[262] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *ICCV*, 2017.

[263] T. Miyato and M. Koyama, "cGANs with projection discriminator," in *ICLR*, 2018.

[264] V. Michalski, V. S. Voleti, S. E. Kahou, A. Ortiz, P. Vincent, C. Pal, and D. Precup, "An empirical study of batch normalization and group normalization in conditional computation," *arXiv preprint arXiv:1908.00061*, 2019.

[265] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *CVPR*, 2019.

[266] T. Chen, M. Lucic, N. Houlsby, and S. Gelly, "On self modulation for generative adversarial networks," in *ICLR*, 2019.

[267] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming." in *ICCV*, 2017, pp. 2755–2763.

[268] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," in *ICLR*, 2018.
[269] B. Li, B. Wu, J. Su, G. Wang, and L. Lin, "Eagleeye: Fast sub-net evaluation for efficient neural network pruning," in *ECCV*, 2020.
[270] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *ICLR*, 2019.
[271] C. Luo, J. Zhan, L. Wang, and W. Gao, "Finet: Using fine-grained batch normalization to train light-weight neural networks," *arXiv preprint arXiv:2005.06828*, 2020.
[272] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," in *NeurIPS*, 2018.
[273] B. Graham, "Low-precision batch-normalized activations," *arXiv preprint arXiv:1702.08231*, 2017.
[274] D. Lin, P. Sun, G. Xie, S. Zhou, and Z. Zhang, "Optimal quantization for batch normalization in neural network deployments and beyond," *arXiv preprint arXiv:2008.13128*, 2020.
[275] A. Ardakani, Z. Ji, S. C. Smithson, B. H. Meyer, and W. J. Gross, "Learning recurrent binary/ternary weights," in *ICLR*, 2019.
[276] L. Hou, J. Zhu, J. Kwok, F. Gao, T. Qin, and T.-Y. Liu, "Normalization helps training of quantized lstm," in *NeurIPS*, 2019.
[277] E. Sari, M. Belbahri, and V. P. Nia, "How does batch normalization help binary training," *arXiv preprint arXiv:1909.09139*, 2019.
[278] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *NeurIPS*, 2014.
[279] W. Sun, W. Jiang, E. Trulls, A. Tagliasacchi, and K. M. Yi, "Attentive context normalization for robust permutation-equivariant learning," in *CVPR*, 2020.
[280] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *ICLR*, 2017.
[281] H. Xiong, L. Huang, M. Yu, L. Liu, F. Zhu, and L. Shao, "On the number of linear regions of convolutional neural networks," in *ICML*, 2020.
[282] Y. Tsuzuku, I. Sato, and M. Sugiyama, "Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks," in *NeurIPS*, 2018.
[283] C. Anil, J. Lucas, and R. Grosse, "Sorting out lipschitz function approximation," in *ICLR*, 2019.
[284] H. Qian and M. N. Wegman, "L2-nonexpansive neural networks," in *ICLR*, 2019.
[285] P. L. Bartlett, D. J. Foster, and M. J. Telgarsky, "Spectrally-normalized margin bounds for neural networks," in *NeurIPS*, 2017.
[286] B. Neyshabur, S. Bhojanapalli, and N. Srebro, "A PAC-bayesian approach to spectrally-normalized margin bounds for neural networks," in *ICLR*, 2018.
[287] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *ICLR*, 2016.
[288] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *ICML*, 2016.