Linearly Constrained Weights: Reducing Activation Shift for Faster Training of Neural Networks

Takuro Kutsuna✉[1] [0000−0001−6965−1512]

Toyota Central R&D Labs. Inc., Aichi, 480-1192, Japan
kutsuna@mosk.tytlabs.co.jp

Abstract. In this paper, we first identify *activation shift*, a simple but remarkable phenomenon in a neural network in which the preactivation value of a neuron has non-zero mean that depends on the angle between the weight vector of the neuron and the mean of the activation vector in the previous layer. We then propose *linearly constrained weights (LCW)* to reduce the activation shift in both fully connected and convolutional layers. The impact of reducing the activation shift in a neural network is studied from the perspective of how the variance of variables in the network changes through layer operations in both forward and backward chains. We also discuss its relationship to the vanishing gradient problem. Experimental results show that LCW enables a deep feedforward network with sigmoid activation functions to be trained efficiently by resolving the vanishing gradient problem. Moreover, combined with batch normalization, LCW improves generalization performance of both feedforward and convolutional networks.

Keywords: Artificial neural networks · Feedforward neural networks · Vanishing gradient problem · Analysis of variance.

1 Introduction

Neural networks with a single hidden layer have been shown to be universal approximators [9, 12]. However, an exponential number of neurons may be necessary to approximate complex functions. One solution to this problem is to use more hidden layers. The representation power of a network increases exponentially with the addition of layers [2, 22]. Various techniques have been proposed for training deep nets, that is, neural networks with many hidden layers, such as layer-wise pretraining [8], rectified linear units [13, 17], residual structures [6], and normalization layers [5, 11].

In this paper, we first identify the *activation shift* that arises in the calculation of the preactivation value of a neuron. The preactivation value is calculated as the dot product of the weight vector of a neuron and an activation vector in the previous layer. In a neural network, an activation vector in a layer can be viewed as a random vector whose distribution is determined by the input distribution and the weights in the preceding layers. The preactivation of a neuron then

has a *non-zero mean* depending on the angle between the weight vector of the neuron and the mean of the activation vector in the previous layer. The angles are generally different according to the neuron, indicating that neurons have distinct mean values, even those in the same layer.

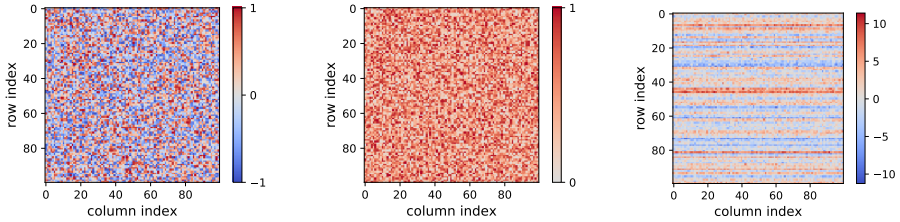
We propose the use of so-called *linearly constrained weights (LCW)* to resolve the activation shift in both fully connected and convolutional layers. An LCW is a weight vector subject to the constraint that the sum of its elements is zero. We investigate the impact of resolving activation shift in a neural network from the perspective of how the variance of variables in a neural network changes according to layer operations in both forward and backward directions. Interestingly, in a fully connected layer in which the activation shift has been resolved by LCW, the variance is amplified by the same rate in both forward and backward chains. In contrast, the variance is more amplified in the forward chain than in the backward chain when activation shift occurs in the layer. This asymmetric characteristic is suggested to be a cause of the vanishing gradient in feedforward networks with sigmoid activation functions. We experimentally demonstrate that we can successfully train a deep feedforward network with sigmoid activation functions by reducing the activation shift using LCW. Moreover, our experiments suggest that LCW improves generalization performance of both feedforward and convolutional networks when combined with batch normalization (BN) [11].

In Section 2, we give a general definition of activation shift in a neural network. In Section 3, we propose LCW as an approach to reduce activation shift and present a technique to efficiently train a network with LCW. In Section 4 we study the impact of removing activation shift in a neural network from the perspective of variance analysis and then discuss its relationship to the vanishing gradient problem. In Section 5, we review related work. We present empirical results in Section 6 and conclude the study in Section 7.

2 Activation Shift

We consider a standard multilayer perceptron (MLP). For simplicity, the number of neurons m is assumed to be the same in all layers. The activation vector in layer l is denoted by $\boldsymbol{a}^l = \left(a_1^l, \ldots, a_m^l\right)^\top \in \mathbb{R}^m$. The input vector to the network is denoted by $\boldsymbol{a}^0$. The weight vector of the i-th neuron in layer l is denoted by $\boldsymbol{w}_i^l \in \mathbb{R}^m$. It is generally assumed that $\|\boldsymbol{w}_i^l\| > 0$. The activation of the i-th neuron in layer l is given by $a_i^l = f\left(z_i^l\right)$ and $z_i^l = \boldsymbol{w}_i^l \cdot \boldsymbol{a}^{l-1} + b_i^l$, where f is a nonlinear activation function, $b_i^l \in \mathbb{R}$ is the bias term, and $z_i^l \in \mathbb{R}$ denotes the preactivation value. Variables z_i^l and a_i^l are regarded as random variables whose distributions are determined by the distribution of the input vector $\boldsymbol{a}^0$, given the weight vectors and the bias terms in the preceding layers.

We introduce activation shift using the simple example shown in Fig. 1. Fig. 1(a) is a heat map representation of a weight matrix $\boldsymbol{W}^l \in \mathbb{R}^{100\times 100}$, whose i-th row vector represents $\boldsymbol{w}_i^l$. In Fig. 1(a), each element of $\boldsymbol{W}^l$ is independently drawn from a uniform random distribution in the range $(-1, 1)$. Fig. 1(b) shows



(a) Weight $\boldsymbol{W}^l$. (b) Activation $\boldsymbol{A}^{l-1}$. (c) Preactivation $\boldsymbol{Z}^l$.

Fig. 1: Activation shift causes a horizontal stripe pattern in preactivation $\boldsymbol{Z}^l = \boldsymbol{W}^l\boldsymbol{A}^{l-1}$, in which each element of $\boldsymbol{W}^l$ and $\boldsymbol{A}^{l-1}$ is randomly generated from the range $(-1, 1)$ and $(0, 1)$, respectively.

an activation matrix $\boldsymbol{A}^{l-1} \in \mathbb{R}^{100\times100}$, whose j-th column vector represents the activation vector corresponding to the j-th sample in a minibatch. Each element of $\boldsymbol{A}^{l-1}$ is randomly sampled from the range $(0, 1)$. We multiply $\boldsymbol{W}^l$ and $\boldsymbol{A}^{l-1}$ to obtain the preactivation matrix $\boldsymbol{Z}^l$, whose i-th row vector represents preactivation values of the i-th neuron in layer l, which is shown in Fig. 1(c). It is assumed that bias terms are all zero. Unexpectedly, a horizontal stripe pattern appears in the heat map of $\boldsymbol{Z}^l$ even though both $\boldsymbol{W}^l$ and $\boldsymbol{A}^{l-1}$ are randomly generated. This pattern is attributed to the activation shift, which is defined as follows:

Definition 1. *$\mathcal{P}_\gamma$ is an m-dimensional probability distribution whose expected value is $\gamma\mathbf{1}_m$, where $\gamma \in \mathbb{R}$ and $\mathbf{1}_m$ is an m-dimensional vector whose elements are all one.*

Proposition 1. *Assume that the activation vector $\boldsymbol{a}^{l-1}$ follows $\mathcal{P}_\gamma$. Given a weight vector $\boldsymbol{w}_i^l \in \mathbb{R}^m$ such that $\|\boldsymbol{w}_i^l\| > 0$, the expected value of $\boldsymbol{w}_i^l \cdot \boldsymbol{a}^{l-1}$ is $|\gamma|\sqrt{m}\|\boldsymbol{w}_i^l\| \cos\theta_i^l$, where θ_i^l is the angle between $\boldsymbol{w}_i^l$ and $\mathbf{1}_m$.*

Proofs of all propositions are provided in Appendix A in the supplementary material.

Definition 2. *From Proposition 1, the expected value of $\boldsymbol{w}_i^l \cdot \boldsymbol{a}^{l-1}$ depends on θ_i^l as long as $\gamma \neq 0$. The distribution of $\boldsymbol{w}_i^l \cdot \boldsymbol{a}^{l-1}$ is then biased depending on θ_i^l; this is called activation shift.*

In Fig. 1, each column vector of $\boldsymbol{A}^{l-1}$ follows $\mathcal{P}_\gamma$ with $\gamma = 0.5$. Therefore, the i-th row of $\boldsymbol{Z}^l$ is biased according to the angle between $\boldsymbol{w}_i^l$ and $\mathbf{1}_m$. We can generalize Proposition 1 for any m-dimensional distribution $\hat{\mathcal{P}}$ instead of $\mathcal{P}_\gamma$ by stating that the distribution of $\boldsymbol{w}^l \cdot \hat{\boldsymbol{a}}^{l-1}$ is biased according to $\hat{\theta}_i^l$ unless $\|\hat{\boldsymbol{\mu}}\| = 0$ as follows:

Proposition 2. *Assume that the activation vector $\hat{\boldsymbol{a}}^{l-1}$ follows an m-dimensional probability distribution $\hat{\mathcal{P}}$ whose expected value is $\hat{\boldsymbol{\mu}} \in \mathbb{R}^m$. Given $\boldsymbol{w}_i^l \in \mathbb{R}^m$ such*

that $\|\boldsymbol{w}_i^l\| > 0$*, it follows that* $E(\boldsymbol{w}_i^l \cdot \hat{\boldsymbol{a}}^{l-1}) = \|\boldsymbol{w}_i^l\| \, \|\hat{\boldsymbol{\mu}}\| \cos\hat{\theta}_i^l$ *if* $\|\hat{\boldsymbol{\mu}}\| > 0$*; otherwise,* $E(\boldsymbol{w}_i^l \cdot \hat{\boldsymbol{a}}^{l-1}) = 0$*, where* $\hat{\theta}_i^l$ *is the angle between* $\boldsymbol{w}_i^l$ *and* $\hat{\boldsymbol{\mu}}$*.*

From Proposition 2, if $\boldsymbol{a}^{l-1}$ follows $\hat{\mathcal{P}}$ with the mean vector $\hat{\boldsymbol{\mu}}$ such that $\|\hat{\boldsymbol{\mu}}\| > 0$, the preactivation z_i^l is biased according to the angle between $\boldsymbol{w}_i^l$ and $\hat{\boldsymbol{\mu}}$.

Note that differences in $E(z_i^l)$ are not resolved by simply introducing bias terms b_i^l, because b_i^l are optimized to decrease the training loss function and not to absorb the differences between $E(z_i^l)$ during the network training. Our experiments suggest that pure MLPs with several hidden layers are not trainable even though they incorporate bias terms. We also tried to initialize b_i^l to absorb the difference in $E(z_i^l)$ at the beginning of the training, though it was unable to train the network, especially when the network has many hidden layers.

3 Linearly Constrained Weights

There are two approaches to reducing activation shift in a neural network. The first one is to somehow make the expected value of the activation of each neuron close to zero, because activation shift does not occur if $\|\hat{\boldsymbol{\mu}}\| = 0$ from Proposition 2. The second one is to somehow regularize the angle between $\boldsymbol{w}_i^l$ and $E\left(\boldsymbol{a}^{l-1}\right)$. In this section, we propose a method to reduce activation shift in a neural network using the latter approach. We introduce $\mathcal{W}_{\mathrm{LC}}$ as follows:

Definition 3. *$\mathcal{W}_{LC}$ is a subspace in $\mathbb{R}^m$ defined by*

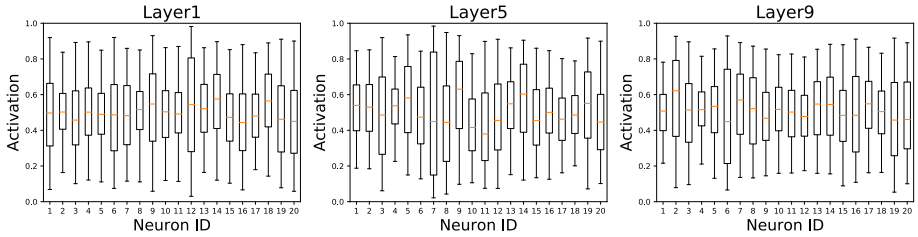
$$\mathcal{W}_{LC} := \{\boldsymbol{w} \in \mathbb{R}^m \mid \boldsymbol{w} \cdot \mathbf{1}_m = 0\}.$$

We call weight vector $\boldsymbol{w}_i^l$ in $\mathcal{W}_{LC}$ the linearly constrained weights (LCWs).

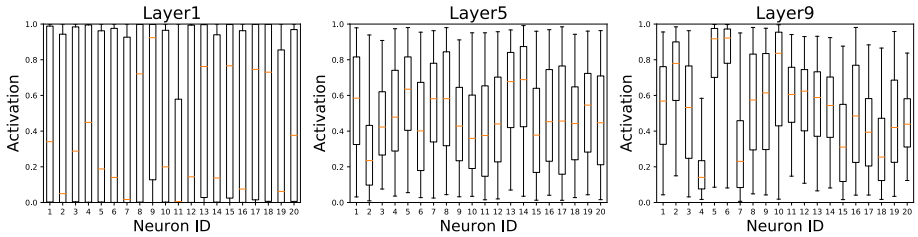
The following holds for $\boldsymbol{w} \in \mathcal{W}_{\mathrm{LC}}$:

Proposition 3. *Assume that the activation vector $\boldsymbol{a}^{l-1}$ follows $\mathcal{P}_\gamma$. Given $\boldsymbol{w}_i^l \in \mathcal{W}_{LC}$ such that $\|\boldsymbol{w}_i^l\| > 0$, the expected value of $\boldsymbol{w}_i^l \cdot \boldsymbol{a}^{l-1}$ is zero.*

Generally, activation vectors in a network do not follow $\mathcal{P}_\gamma$, and consequently, LCW cannot resolve the activation shift perfectly. However, we experimentally observed that the activation vector approximately follows $\mathcal{P}_\gamma$ in each layer. Fig. 2(a) shows boxplot summaries of a_i^l in a 10-layer sigmoid MLP with LCW, in which the weights of the network were initialized using the method that will be explained in Section 4. We used a minibatch of samples in the CIFAR-10 dataset [14] to evaluate the distribution of a_i^l. In the figure, the 1%, 25%, 50%, 75%, and 99% quantiles are displayed as whiskers or boxes. We see that a_i^l distributes around 0.5 in each neuron, which suggests that $\boldsymbol{a}^l \sim \mathcal{P}_\gamma$ approximately holds in every layer. We also observed the distribution of a_i^l after 10 epochs of training, which are shown in Fig. 2(b). We see that $\boldsymbol{a}^l$ are less likely follow $\mathcal{P}_\gamma$, but a_i^l takes various values in each neuron. In contrast, if we do not apply LCW to the network, the variance of a_i^l rapidly shrinks through layers immediately after the initialization as shown in Fig. 3, in which weights are initialized by the



(a) Immediately after the initialization.



(b) After 10 epochs training.

Fig. 2: Boxplot summaries of a_i^l on the first 20 neurons in layers 1,5, and 9 of the 10-layer sigmoid MLP with LCW.

method in [3]. Experimental results in Section 6 suggest that we can train MLPs with several dozens of layers very efficiently by applying the LCW. The effect of resolving the activation shift by applying LCW will be theoretically analyzed in Section 4.

It is possible to force $\boldsymbol{a}^l$ to follow $\mathcal{P}_\gamma$ by applying BN to preactivation z_i^l. The distribution of z_i^l is then normalized to have zero-mean and unit variance, and consequently, $a_i^l = f(z_i^l)$ are more likely to follow the same distribution, indicating that $\boldsymbol{a}^l \sim \mathcal{P}_\gamma$ holds. As will be discussed in Section 5, BN itself also has an effect of reducing activation shift. However, our experimental results suggest that we can train deep networks more smoothly by combining LCW and BN, which will be shown in Section 6.

3.1 Learning LCW via Reparameterization

A straightforward way to train a neural network with LCW is to solve a constrained optimization problem, in which a loss function is minimized under the condition that each weight vector is included in $\mathcal{W}_{\mathrm{LC}}$. Although several methods are available to solve such constrained problems, for example, the gradient projection method [15], it might be less efficient to solve a constrained optimization problem than to solve an unconstrained one. We propose a reparameterization technique that enables us to train a neural network with LCW using a solver for unconstrained optimization. The constraints on the weight vectors are embedded into the structure of the neural network by the following reparameterization.

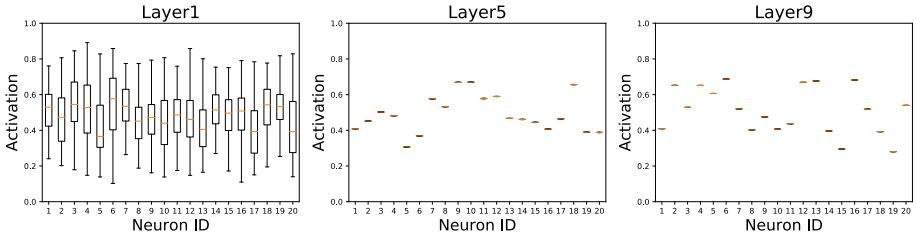


Fig. 3: Boxplot summaries of a_i^l on neurons in layers 1,5, and 9 of the 10-layer sigmoid MLP *without* LCW, in which weights are initialized by the method in [3].

Reparameterization: Let $\boldsymbol{w}_i^l \in \mathbb{R}^m$ be a weight vector in a neural network. To apply LCW to $\boldsymbol{w}_i^l$, we reparameterize $\boldsymbol{w}_i^l$ using vector $\boldsymbol{v}_i^l \in \mathbb{R}^{m-1}$ as $\boldsymbol{w}_i^l = \boldsymbol{B}_m \boldsymbol{v}_i^l$, where $\boldsymbol{B}_m \in \mathbb{R}^{m \times (m-1)}$ is an orthonormal basis of $\mathcal{W}_{\mathrm{LC}}$, written as a matrix of column vectors.

It is obvious that $\boldsymbol{w}_i^l = \boldsymbol{B}_m \boldsymbol{v}_i^l \in \mathcal{W}_{\mathrm{LC}}$. We then solve the optimization problem in which $\boldsymbol{v}_i^l$ is considered as a new variable in place of $\boldsymbol{w}_i^l$. This optimization problem is unconstrained because $\boldsymbol{v}_i^l \in \mathbb{R}^{m-1}$. We can search for $\boldsymbol{w}_i^l \in \mathcal{W}_{\mathrm{LC}}$ by exploring $\boldsymbol{v}_i^l \in \mathbb{R}^{m-1}$. The calculation of an orthonormal basis of $\mathcal{W}_{\mathrm{LC}}$ is described in Appendix B in the supplementary material. Note that the proposed reparameterization can be implemented easily and efficiently using modern frameworks for deep learning based on GPUs.

3.2 LCW for Convolutional Layers

We consider a convolutional layer with C_{out} convolutional kernels. The size of each kernel is $C_{in} \times K_h \times K_w$, where C_{in}, K_h, and K_w are the number of the input channels, height of the kernel, and width of the kernel, respectively. The layer outputs C_{out} channels of feature maps. In a convolutional layer, activation shift occurs at the channel level, that is, the preactivation has different mean value in each output channel depending on the kernel of the channel. We propose a simple extension of LCW for reducing the activation shift in convolutional layers by introducing a subspace $\mathcal{W}_{\mathrm{LC}}^{\mathrm{kernel}}$ in $\mathbb{R}^{C_{in} \times K_h \times K_w}$ defined as follows:

$$\mathcal{W}_{\mathrm{LC}}^{\mathrm{kernel}} := \left\{ \boldsymbol{w} \in \mathbb{R}^{C_{in} \times K_h \times K_w} \;\middle|\; \sum_{i=1}^{C_{in}} \sum_{j=1}^{K_h} \sum_{k=1}^{K_w} w_{i,j,k} = 0 \right\},$$

where $w_{i,j,k}$ indicates the (i,j,k)-th element of w. Subspace $\mathcal{W}_{\mathrm{LC}}^{\mathrm{kernel}}$ is a straightforward extension of $\mathcal{W}_{\mathrm{LC}}$ to the kernel space. To apply LCW to a convolutional layer, we restrict each kernel of the layer in $\mathcal{W}_{\mathrm{LC}}^{\mathrm{kernel}}$. It is possible to apply the reparameterization trick described in the previous subsection to LCW for convolutional layers. We can reparameterize the kernel using an orthonormal basis of $\mathcal{W}_{\mathrm{LC}}^{\mathrm{kernel}}$ in which the kernel in $\mathbb{R}^{C_{in} \times K_h \times K_w}$ is unrolled into a vector of length $C_{in} K_h K_w$.

4 Variance Analysis

In this section, we first investigate the effect of removing activation shift in a neural network based on an analysis of how the variance of variables in the network changes through layer operations both in forward and backward directions. Then, we discuss its relationship to the vanishing gradient problem.

4.1 Variance Analysis of a Fully Connected Layer

The forward calculation of a fully connected layer is $\boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l$, where $\boldsymbol{W}^l = (\boldsymbol{w}_1^l, \ldots, \boldsymbol{w}_m^l)^\top$. We express the j-th column vector of $\boldsymbol{W}^l$ as $\tilde{\boldsymbol{w}}_j^l$. If we denote the gradient of a loss function with respect to parameter v as ∇_v, the backward calculation regarding $\boldsymbol{a}^{l-1}$ is $\nabla_{\boldsymbol{a}^{l-1}} = (\boldsymbol{W}^l)^\top \nabla_{\boldsymbol{z}^l}$. The following proposition holds for the forward computation, in which $\boldsymbol{I}_m$ is the identity matrix of order $m \times m$, V indicates the variance, and Cov denotes the variance-covariance matrix.

Proposition 4. *Assuming that* $\boldsymbol{w}_i^l \in \mathcal{W}_{LC}$, $E(\boldsymbol{a}^{l-1}) = \gamma_{\boldsymbol{a}^{l-1}} \boldsymbol{1}_m$ *with* $\gamma_{\boldsymbol{a}^{l-1}} \in \mathbb{R}$, $Cov(\boldsymbol{a}^{l-1}) = \sigma^2_{\boldsymbol{a}^{l-1}} \boldsymbol{I}_m$ *with* $\sigma_{\boldsymbol{a}^{l-1}} \in \mathbb{R}$, *and* $\boldsymbol{b}^l = \boldsymbol{0}$, *it holds that* $E(z_i^l) = 0$ *and* $V(z_i^l) = \sigma^2_{\boldsymbol{a}^{l-1}} \|\boldsymbol{w}_i^l\|^2$.[1]

We also have the following proposition for the backward computation.

Proposition 5. *Assuming that* $E(\nabla_{\boldsymbol{z}^l}) = \boldsymbol{0}$ *and* $Cov(\nabla_{\boldsymbol{z}^l}) = \sigma^2_{\nabla_{\boldsymbol{z}^l}} \boldsymbol{I}_m$ *with* $\sigma_{\nabla_{\boldsymbol{z}^l}} \in \mathbb{R}$, *it holds that* $E(\nabla_{a_j^{l-1}}) = 0$ *and* $V(\nabla_{a_j^{l-1}}) = \sigma^2_{\nabla_{\boldsymbol{z}^l}} \|\tilde{\boldsymbol{w}}_j^l\|^2$.

For simplicity, we assume that $\forall i, \|\boldsymbol{w}_i^l\|^2 = \eta^l$ and $\forall j, \|\tilde{\boldsymbol{w}}_j^l\|^2 = \xi^l$. Proposition 4 then indicates that, in the forward computation, $V(z_i^l)$, the variance of the output, becomes η^l times larger than that of the input, $V(a_i^{l-1})$. Proposition 5 indicates that, in the backward chain, $V(\nabla_{a_i^{l-1}})$, the variance of the output, becomes ξ^l times larger than that of the input, $V(\nabla_{z_i^l})$. If $\boldsymbol{W}^l$ is a square matrix, then $\eta^l = \xi^l$ (see Appendix A for proof), meaning that the variance is amplified at the same rate in both the forward and backward directions. Another important observation is that, if we replace $\boldsymbol{W}^l$ with $\kappa \boldsymbol{W}^l$, the rate of amplification of the variance becomes κ^2 times larger in both the forward and backward chains. This property does not hold if $\boldsymbol{w}_i^l \notin \mathcal{W}_{\mathrm{LC}}$, because in this case $E(z_i^l) \neq 0$ because of the effect of the activation shift. The variance is then more amplified in the forward chain than in the backward chain by the weight rescaling.

4.2 Variance Analysis of a Nonlinear Activation Layer

The forward and backward chains of the nonlinear activation layer are given by $a_i^l = f(z_i^l)$ and $\nabla_{z_i^l} = f'(z_i^l) \nabla_{a_i^l}$, respectively. The following proposition holds if f is the ReLU [13, 17] function.

[1] A similar result is discussed in [10], but our result is more general because we do not assume the distribution of $\boldsymbol{a}^{l-1}$ to be Gaussian distribution, which is assumed in [10].

Proposition 6. *Assuming that* z_i^l *and* $\nabla_{a_i^l}$ *independently follow* $\mathcal{N}(0, \sigma^2_{z_i^l})$ *and* $\mathcal{N}(0, \sigma^2_{\nabla_{a_i^l}})$*, respectively, where* $\mathcal{N}(\mu, \sigma^2)$ *indicates a normal distribution with mean* μ *and variance* σ^2*, it holds that*

$$V(a_i^l) = \frac{\sigma^2_{z_i^l}}{2}\left(1 - \frac{1}{\pi}\right) \quad \textit{and} \quad V(\nabla_{z_i^l}) = \frac{\sigma^2_{\nabla_{a_i^l}}}{2}.$$

We denote the rate of amplification of variance in the forward and backward directions of a nonlinear activation function by $\phi_{\mathrm{fw}} := V(a_i^l)/V(z_i^l)$ and $\phi_{\mathrm{bw}} := V(\nabla_{z_i^l})/V(\nabla_{a_i^l})$, respectively. Proposition 6 then indicates that the variance is amplified by a factor of $\phi_{\mathrm{fw}} = 0.34$ in the forward chain and by a factor of $\phi_{\mathrm{bw}} = 0.5$ in the backward chain through the ReLU activation layer.

If f is the sigmoid activation, there is no analytical solution for the variance of a_i^l and $\nabla_{z_i^l}$. We therefore numerically examined ϕ_{fw} and ϕ_{bw} for the sigmoid activation under the conditions that z_i^l follows $\mathcal{N}(0, \hat{\sigma}^2)$ for $\hat{\sigma} \in \{0.5, 1, 2\}$ and $\nabla_{a_i^l}$ follows $\mathcal{N}(0, 1)$. As a result, we obtained $(\phi_{\mathrm{fw}}, \phi_{\mathrm{bw}}) = (0.236, 0.237)$, $(0.208, 0.211)$, and $(0.157, 0.170)$ for $\hat{\sigma} = 0.5, 1$, and 2, respectively. It suggests that the difference between ϕ_{fw} and ϕ_{bw} in the sigmoid activation layer decreases as the variance of z_i^l decreases.

4.3 Relationship to the Vanishing Gradient Problem

We consider an MLP in which the number of neurons is the same in all hidden layers. We initialize weights in the network by the method based on minibatch statistics: weights are first generated randomly, then rescaled so that the preactivation in each layer has unit variance on the minibatch of samples. In fully connected layers with standard weights, the variance of variables in the network is more amplified in the forward chain than in the backward chain by the weight rescaling, as discussed in Subsection 4.1. In contrast, in the sigmoid activation layers, the rate of amplification of the variance is almost the same in the forward and backward directions, as mentioned in the previous subsection. Then, the variance of the preactivation gradient decreases exponentially by rescaling the weights to maintain the variance of the preactivation in the forward chain, resulting in the vanishing gradient, that is, the preactivation gradient in earlier layers has almost zero variance, especially when the network have many layers.

In contrast, when the LCW is applied to the network, the variance is amplified at the same rate in both the forward and backward chains through fully connected layers regardless of the weight rescaling. In this case, the preactivation gradient has a similar variance in each layer after the initialization, assuming that the sigmoid activation is used. Concretely, the variance is amplified by approximately 0.21 through the sigmoid activation layers in both the forward and backward chains. Then, fully connected layers are initialized to have the amplification rate of 1/0.21 to keep the preactivation variance in the forward chain. The gradient variance is then also amplified by 1/0.21 in the backward chain of

fully connected layers with LCW, indicating that the gradient variance is also preserved in the backward chain.

From the analysis in the previous subsections, we also see that normal fully connected layer and the ReLU layer have opposite effect on amplifying the variance in each layer, This may be another explanation why ReLU works well in practice without techniques such as BN.

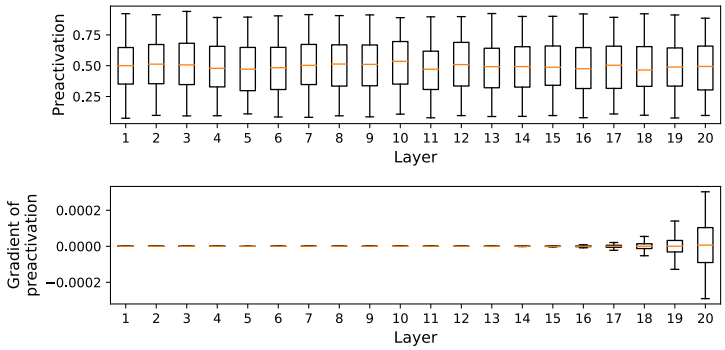
4.4 Example

For example, we use a 20-layered MLP with sigmoid activation functions. The weights of the MLP are initialized according to the method described in the previous subsection. We randomly took 100 samples from the CIFAR-10 dataset and input them into the MLP. The upper part of Fig. 4 (a) shows boxplot summaries of the preactivation in each layer. The lower part shows boxplot summaries of the gradient with respect to the preactivation in each layer, in which the standard cross-entropy loss is used to obtain the gradient. From Fig. 4 (a), we see that the variance of the preactivation is preserved in the forward chain, whereas the variance of the preactivation gradient rapidly shrinks to zero in the backward chain, suggesting the vanishing gradient.

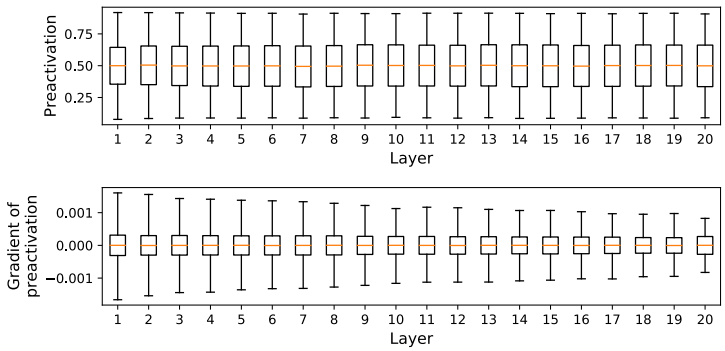
Next, LCW is applied to the MLP, and then, the weighs are initialized by the same procedure. Fig. 4 (b) shows the distribution of the preactivation and its gradient in each layer regarding the same samples from CIFAR-10. In contrast to Fig. 4 (a), the variance of the preactivation gradient does not shrink to zero in the backward chain. Instead we observe that the variance of the gradient slightly increases through the backward chain. This can be explained by the fact that the variance is slightly more amplified in the backward chain than in the forward chain through the sigmoid layer, as discussed in Subsection 4.2. These results suggest that we can resolve the vanishing gradient problem in an MLP with sigmoid activation functions by applying LCW and by initializing weights to preserve the preactivation variance in the forward chain.

5 Related work

Ioffe and Szegedy [11] proposed the BN approach for accelerating the training of deep nets. BN was developed to address the problem of *internal covariate shift*, that is, training deep nets is difficult because the distribution of the input to a layer changes as the weights of the preceding layers change during training. BN is widely adopted in practice and shown to accelerate the training of deep nets, although it has recently been argued that the success of BN does not stem from the reduction of the internal covariate shift [20]. BN computes the mean and standard deviation of z_i^l based on a minibatch, and then, normalizes z_i^l by using these statistics. Gülçehre and Bengio [5] proposed the *standardization layer (SL)* approach, which is similar to BN. The main difference is that SL normalizes a_i^l, whereas BN normalizes z_i^l. Interestingly, both BN and SL can be considered mechanisms for reducing the activation shift. On one hand, SL



(a) MLP with standard weights.



(b) MLP with LCWs.

Fig. 4: Boxplot summaries of the preactivation (top) and its gradient (bottom) in 20-layered sigmoid MLPs with standard weights (a) and LCWs (b).

reduces the activation shift by forcing $\|\hat{\boldsymbol{\mu}}\| = 0$ in Proposition 2. On the other hand, BN reduces the activation shift by removing the mean from z_i^l for each neuron. A drawback of both BN and SL is that the model has to be switched during inference to ensure that its output depends only on the input and not the minibatch. In contrast, the LCW proposed in this paper do not require any change in the model during inference.

Salimans and Kingma [19] proposed *weight normalization (WN)* in which a weight vector $\boldsymbol{w}_i^l \in \mathbb{R}^m$ is reparameterized as $\boldsymbol{w}_i^l = (g_i^l/\|\boldsymbol{v}_i^l\|)\boldsymbol{v}_i^l$, where $g_i^l \in \mathbb{R}$ and $\boldsymbol{v}_i^l \in \mathbb{R}^m$ are new parameters. By definition, WN does not have the property of reducing the activation shift, because the degrees of freedom of $\boldsymbol{w}_i^l$ are unchanged by the reparameterization. They also proposed a minibatch-based initialization by which weight vectors are initialized so that z_i^l has zero mean and unit variance, indicating that the activation shift is resolved immediately after the initialization. Our preliminary results presented in Section 6 suggest that to start learning with initial weights that do not incur activation shift is not

sufficient to train very deep nets. It is important to incorporate a mechanism that reduces the activation shift during training.

Ba et al. [1] proposed *layer normalization (LN)* as a variant of BN. LN normalizes z_i^l over the neurons in a layer on a sample in a minibatch, whereas BN normalizes z_i^l over the minibatch on a neuron. From the viewpoint of reducing the activation shift, LN is not as direct as BN. Although LN does not resolve the activation shift, it should normalize the degree of activation shift in each layer.

Huang et al. [10] proposed *centered weight normalization (CWN)* as an extension of WN, in which parameter $\boldsymbol{v}_i^l$ in WN is reparameterized by $\boldsymbol{v}_i^l = \tilde{\boldsymbol{v}}_i^l - \mathbf{1}_m(\mathbf{1}_m^\top \tilde{\boldsymbol{v}}_i^l)/m$ with $\tilde{\boldsymbol{v}}_i^l \in R^m$. CWN therefore forces a weight vector $\boldsymbol{w}_i^l$ to satisfy both $\|\boldsymbol{w}_i^l\| = 1$ and $\mathbf{1}_m^\top \boldsymbol{w}_i^l = 0$. CWN was derived from the observation that, in practice, weights in a neural network are initially sampled from a distribution with zero-mean. CWN and LCW share the idea of restricting weight vectors so that they have zero mean during training, although they come from different perspectives and have different implementations. The main differences between CWN and LCW are the following: CWN forces weight vectors to have both unit norm and zero mean, whereas LCW only forces the latter from the analysis that the latter constraint is essential to resolve the activation shift; LCW embeds the constraint into the network structure using the orthonormal basis of a subspace of weight vectors; the effect of reducing activation shift by introducing LCW is analyzed from the perspective of variance amplification in both the forward and backward chains.

Miyato et al. [16] proposed *spectral normalization (SN)* that constrains the spectral norm, that is, the largest singular value, of a weight matrix equal to 1. SN was introduced to control the Lipschitz constant of the discriminator in the GAN framework [4] to stabilize the training. The relationship between the spectral norm of weights and the generalization ability of deep nets is discussed in [23]. However, controlling the spectral norm of weight matrices is orthogonal to the reduction of the activation shift.

He et al. [6] proposed *residual network* that consists of a stack of residual blocks with skip connections. If we denote the input to the l-th residual block by $\boldsymbol{x}^l \in \mathbb{R}^m$, the output $\boldsymbol{x}^{l+1}$, which is the input to the next residual block, is given by $\boldsymbol{x}^{l+1} = \boldsymbol{x}^l + \boldsymbol{\mathcal{F}}_l(\boldsymbol{x}^l)$, where $\boldsymbol{\mathcal{F}}_l : \mathbb{R}^m \to \mathbb{R}^m$ is a mapping defined by a stack of nonlinear layers. In contrast to the original residual network that regard the activation as $\boldsymbol{x}^l$, He et al. [7] proposed *preactivation structure* in which the preactivation is regarded as $\boldsymbol{x}^l$. Residual network will indirectly reduce the impact of the activation shift. The reason is explained below: In a residual network, it holds that $\boldsymbol{x}^L = \boldsymbol{x}^0 + \sum_{l=0}^{L-1} \boldsymbol{\mathcal{F}}_l(\boldsymbol{x}^l)$. The activation shift can occur in each of $\boldsymbol{\mathcal{F}}_l(\boldsymbol{x}^l)$, that is, each output element of $\boldsymbol{\mathcal{F}}_l(\boldsymbol{x}^l)$ has different mean. However, the shift pattern is almost random in each $\boldsymbol{\mathcal{F}}_l(\boldsymbol{x}^l)$, and consequently, the mean shift in $\boldsymbol{x}^L$ can be moderate because it is the average over these random shifts. This may be another reason why residual networks are successful in training deep models.

6 Experiments

We conducted experiments using the CIFAR-10 and CIFAR-100 datasets [14], which consist of color natural images each of which is annotated corresponding to 10 and 100 classes of objects, respectively. We preprocessed each dataset by subtracting the channel means and dividing by the channel standard deviations. We adopted standard data augmentation [6]: random cropping and horizontal flipping.

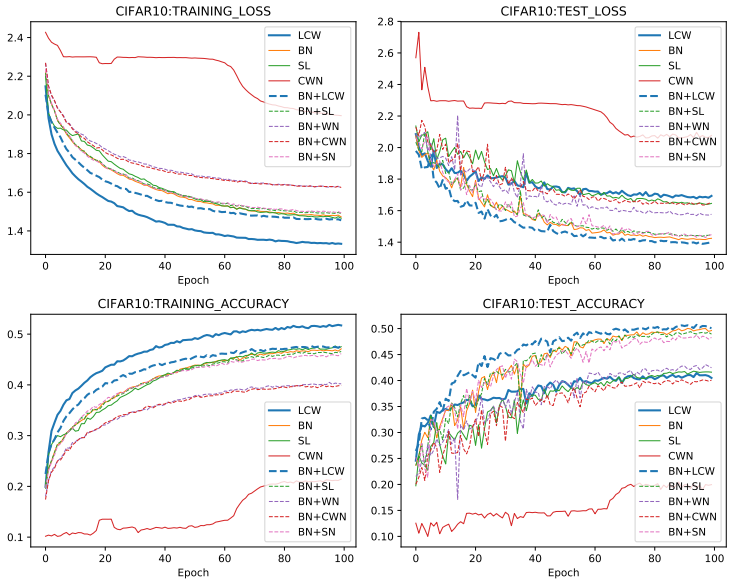
All experiments were performed using Python 3.6 with PyTorch 0.4.1 [18] on a system running Ubuntu 16.04 LTS with GPUs. We implemented LCW using standard modules equipped with PyTorch. As implementation of BN, SL, WN, and SN, we employed corresponding libraries in PyTorch. We implemented CWN by modifying modules for WN.

6.1 Deep MLP with Sigmoid Activation Functions

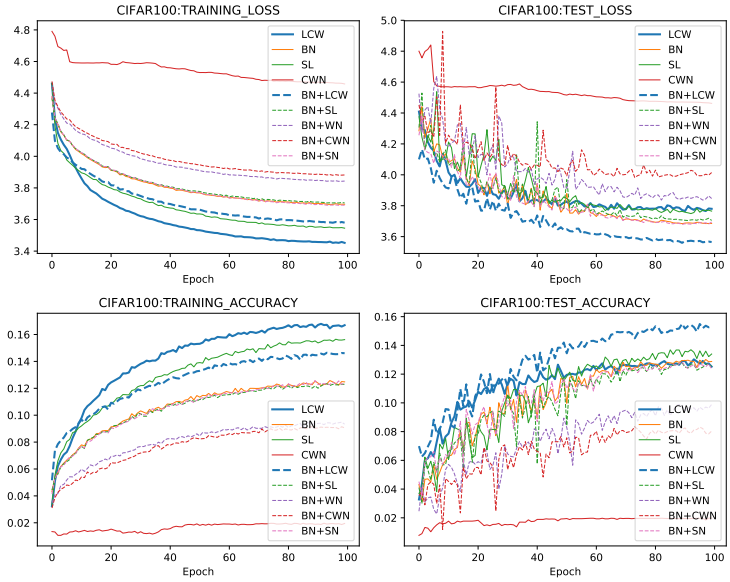
We first conducted experiments using an MLP model with 50 hidden layers, each containing 256 hidden units with sigmoid activation functions, followed by a softmax layer combined with a cross-entropy loss function. We applied each of LCW, BN, SL, WN, CWN, and SN to the model, and compared the performance. We also considered models with each of the above techniques (other than BN) combined with BN. These models are annotated with, for example, "BN+LCW" in the results.

Models with LCW were initialized following the method described in Section 4.3. Models with WN or CWN were initialized according to [19]. Models with BN, SL, or SN were initialized using the method proposed in [3]. Each model was trained using a stochastic gradient descent with a minibatch size of 128, momentum of 0.9, and weight decay of 0.0001 for 100 epochs. The learning rate starts from 0.1 and is multiplied by 0.95 after every epoch until it reaches the lower threshold of 0.001.

Fig. 5 shows the curve of training loss, test loss, training accuracy, and test accuracy of each model on each dataset, in which the horizontal axis shows the training epoch. The results of MLPs with WN or SN are omitted in Fig. 5, because the training of these models did not proceed at all. This result matches our expectation that reducing the activation shift is essential to train deep neural networks, because WN and SN themselves do not have the effect of reducing activation shift as discussed in Section 5. We see that LCW achieves higher rate of convergence and gives better scores with respect to the training loss/accuracy compared with other models. However, with respect to the test loss/accuracy, the scores of LCW are no better than that of other models. This result suggests that LCW has an ability to accelerate the network training but may increase the risk of overfitting. In contrast, combined with BN, LCW achieves better performance in test loss/accuracy, as shown by the results annotated with "BN+LCW" in Fig. 5. We think such improvement was provided because LCW accelerated the training while the generalization ability of BN was maintained.



(a) Results for the CIFAR-10 dataset.



(b) Results for the CIFAR-100 dataset.

Fig. 5: Training loss (upper left), test loss (upper right), training accuracy (lower left), and test accuracy (lower right) of 50-layer MLPs for CIFAR-10 (a) and CIFAR-100 (b).

Table 1: Test accuracy/loss of convolutional models for CIFAR-10 and CIFAR-100 datasets.

	CIFAR-10		CIFAR-100	
Model	Test Accuracy	Test Loss	Test Accuracy	Test Loss
VGG19	0.936	0.354	0.732	1.788
VGG19+LCW	0.938	0.332	0.741	1.569
VGG19+WN	0.931	0.391	0.725	1.914
VGG19+CWN	0.934	0.372	0.727	1.827
VGG19+SN	0.936	0.358	0.733	1.644
ResNet18	0.952	0.204	0.769	0.978
ResNet18+LCW	0.952	0.187	0.770	0.955
ResNet18+WN	0.951	0.206	0.777	0.947
ResNet18+CWN	0.948	0.216	0.781	0.949
ResNet18+SN	0.952	0.206	0.780	1.015

6.2 Deep Convolutional Networks with ReLU Activation Functions

In this subsection, we evaluate LCW using convolutional networks with ReLU activation functions. As base models, we employed the following two models:
VGG19: A 19-layer convolutional network in which 16 convolutional layers are connected in series, followed by three fully connected layers with dropout [21]. We inserted BN layers before each ReLU layer in VGG19, although the original VGG model does not include BN layers.[2]
ResNet18: An 18-layer convolutional network with residual structure [6], which consists of eight residual units each of which contains two convolutional layers in the residual part. We employed the full preactivation structure proposed in [7]. In ResNet18, BN layers are inserted before each ReLU layer.

We applied LCW, WN, CWN, or SN to VGG19 and ResNet18, respectively, and compared the performance including the plain VGG19 and ResNet18 models. Each model was trained using a stochastic gradient descent with a minibatch size of 128, momentum of 0.9, and weight decay of 0.0005. For the CIFAR-10 dataset, we trained each model for 300 epochs with the learning rate that starts from 0.1 and is multiplied by 0.95 after every three epochs until it reaches 0.001. For the CIFAR-100 dataset, we trained each model for 500 epochs with the learning rate multiplied by 0.95 after every five epochs.

Table 1 shows the test accuracy and loss for the CIFAR-10 and CIFAR-100 datasets, in which each value was evaluated as the average over the last ten epochs of training. We see that LCW improves the generalization performance of VGG19 with respect to both the test accuracy and loss. The improvement is more evident for the CIFAR-100 dataset. The curve of training loss and accuracy of VGG19-based models for CIFAR-100 are shown in Fig. 6. We see that LCW enhances the rate of convergence, which we think lead to the better performance. In contrast, the improvement brought by LCW is less evident in ResNet18, in particular, with respect to the test accuracy. We observed little difference in

[2] This is mainly because VGG was proposed earlier than BN.

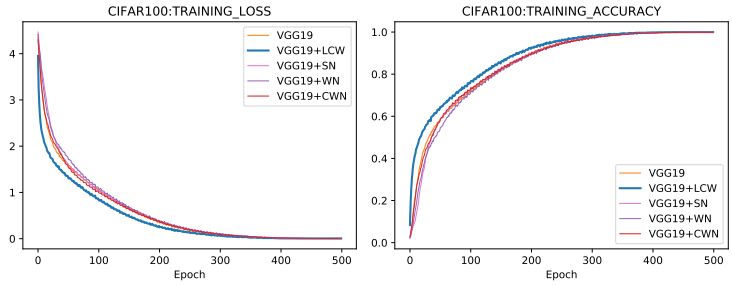


Fig. 6: Training loss (left) and training accuracy (right) of the VGG19-based models for the CIFAR-100 dataset.

the training curve of ResNet18 with and without LCW. A possible reason for this is that the residual structure itself has an ability to mitigate the impact of the activation shift, as discussed in Section 5, and therefore the reduction of activation shift by introducing LCW was less beneficial for ResNet18.

7 Conclusion

In this paper, we identified the activation shift in a neural network: the pre-activation of a neuron has non-zero mean depending on the angle between the weight vector of the neuron and the mean of the activation vector in the previous layer. The LCW approach was then proposed to reduce the activation shift. We analyzed how the variance of variables in a neural network changes through layer operations in both forward and backward chains, and discussed its relationship to the vanishing gradient problem. Experimental results suggest that the proposed method works well in a feedforward network with sigmoid activation functions, resolving the vanishing gradient problem. We also showed that existing methods that successfully accelerate the training of deep neural networks, including BN and residual structures, have an ability to reduce the effect of activation shift, suggesting that alleviating the activation shift is essential for efficient training of deep models. The proposed method achieved better performance when used in a convolutional network with ReLU activation functions combined with BN. Future work includes investigating the applicability of the proposed method for other neural network structures, such as recurrent structures.

References

1. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. In: NIPS 2016 Deep Learning Symposium (2016)
2. Eldan, R., Shamir, O.: The power of depth for feedforward neural networks. In: Annual Conference on Learning Theory. vol. 49, pp. 907–940 (2016)

3. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: International Conference on Artificial Intelligence and Statistics. pp. 249–256 (2010)
4. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 2672–2680. Curran Associates, Inc. (2014)
5. Çağlar Gülçehre, Bengio, Y.: Knowledge matters: Importance of prior information for optimization. Journal of Machine Learning Research **17**(8), 1–32 (2016)
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778 (2016)
7. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: European Conference on Computer Vision. pp. 630–645 (2016)
8. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006)
9. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks **2**(5), 359–366 (1989)
10. Huang, L., Liu, X., Liu, Y., Lang, B., Tao, D.: Centered weight normalization in accelerating training of deep neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition. pp. 2803–2811 (2017)
11. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning. pp. 448–456 (2015)
12. Irie, B., Miyake, S.: Capabilities of three-layered perceptrons. In: IEEE International Conference on Neural Networks. vol. 1, pp. 641–648 (1988)
13. Jarrett, K., Kavukcuoglu, K., Ranzato, M., LeCun, Y.: What is the best multi-stage architecture for object recognition? In: IEEE International Conference on Computer Vision. pp. 2146–2153 (2009)
14. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Tech. rep., University of Toronto (2009)
15. Luenberger, D.G., Ye, Y.: Linear and Nonlinear Programming. Springer (2015)
16. Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y.: Spectral normalization for generative adversarial networks. In: International Conference on Learning Representations (2018)
17. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: International Conference on Machine Learning. pp. 807–814 (2010)
18. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In: NIPS 2017 Workshop Autodiff (2017)
19. Salimans, T., Kingma, D.P.: Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In: Advances in Neural Information Processing Systems. pp. 901–909. Curran Associates, Inc. (2016)
20. Santurkar, S., Tsipras, D., Ilyas, A., Madry, A.: How does batch normalization help optimization? In: Advances in Neural Information Processing Systems, pp. 2488–2498. Curran Associates, Inc. (2018)
21. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556 (2014)
22. Telgarsky, M.: Benefits of depth in neural networks. In: Annual Conference on Learning Theory. vol. 49, pp. 1517–1539 (2016)
23. Yoshida, Y., Miyato, T.: Spectral norm regularization for improving the generalizability of deep learning. CoRR, abs/1705.10941 (2017)