This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

Improving training of deep neural networks via Singular Value Bounding

Kui Jia[1], Dacheng Tao[2], Shenghua Gao[3], and Xiangmin Xu[1]

[1]School of Electronic and Information Engineering, South China University of Technology, Guangzhou, China
[2]UBTech Sydney AI Institute, SIT, FEIT, The University of Sydney, Australia
[3]School of Information Science and Technology, ShanghaiTech University, Shanghai, China
{kuijia,xmxu}@scut.edu.cn, dacheng.tao@sydney.edu.au, gaoshh@shanghaitech.edu.cn

Abstract

Deep learning methods achieve great success recently on many computer vision problems. In spite of these practical successes, optimization of deep networks remains an active topic in deep learning research. In this work, we focus on investigation of the network solution properties that can potentially lead to good performance. Our research is inspired by theoretical and empirical results that use orthogonal matrices to initialize networks, but we are interested in investigating how orthogonal weight matrices perform when network training converges. To this end, we propose to constrain the solutions of weight matrices in the orthogonal feasible set during the whole process of network training, and achieve this by a simple yet effective method called Singular Value Bounding (SVB). In SVB, all singular values of each weight matrix are simply bounded in a narrow band around the value of 1*. Based on the same motivation, we also propose Bounded Batch Normalization (BBN), which improves Batch Normalization by removing its potential risk of ill-conditioned layer transform. We present both theoretical and empirical results to justify our proposed methods. Experiments on benchmark image classification datasets show the efficacy of our proposed SVB and BBN. In particular, we achieve the state-of-the-art results of* 3.06% *error rate on CIFAR10 and* 16.90% *on CIFAR100, using off-the-shelf network architectures (Wide ResNets). Our preliminary results on ImageNet also show the promise in large-scale learning. We release the implementation code of our methods at* www.aperture-lab.net/research/svb.

1. Introduction

Deep learning methods keep setting the new state-of-the-art for many computer vision problems, with image classification [20] and object detection [16] as the prominent examples. These practical successes are largely achieved by newly proposed deep architectures that have huge model capacities, such as Inception [25] and ResNet [7]. Training of these ultra-deep/ultra-wide networks are enabled by modern techniques such as Batch Normalization (BN) [11] and residual learning [7].

In spite of these practical successes, however, optimization of deep networks remains an active topic in deep learning research. Until recently, deep networks are considered to be difficult to train. Researchers argue for different reasons causing such difficulties, such as the problem of vanishing/exploding gradients [6, 18], internal shift of feature statistics [11], and also the proliferation of saddle points [5, 12]. To address these issues, different schemes of parameter initialization [6, 21], shortcut connections [7, 8], normalization of internal activations [11], and second-order optimization methods [5] are respectively proposed.

In this work, we focus on another important issue to address the difficulty of training deep neural networks. In particular, given the high-dimensional solution space of deep networks, it is unclear on the properties of the (arguably) optimal solutions that can give good performance at inference. Without knowing this, training by a specified objective function easily goes to unexpected results, partially due to the proliferation of local optima/critical points [5, 12]. For example, it is empirically observed in [7] that adding extra layers to a standard convolutional network (ConvNet) does not necessarily give better image classification results. This unclear issue is further compounded by other (aforementioned) optimization difficulties.

Existing deep learning research has some favors on the solutions of network parameters [4, 21], and also on network architectures that can give desirable solutions [25, 3, 7]. In this paper, we are inspired by the analysis of orthogonal initialization in [21], and propose to constrain the solutions of weight matrices in the orthogonal feasible set during the whole process of network training. To this end, we propose a simple yet effective method called Singular Val-

ue Bounding (SVB). In SVB, all singular values of each weight matrix are simply bounded in a narrow band around the value of 1 (Section 3). When using stochastic gradient descent (SGD) or its variants for network training, this amounts to turning SVB on by every a specified number of iterations. We present theoretical analysis, using deep linear networks, to show how such learned networks are better on forward-propagation to achieve training objectives, and backward-propagation of training errors (Section 4). Batch normalization [11] is a very effective method to improve and accelerate network training. We prove that in the framework of our theoretical analysis, trainable parameters in BN may cause ill-conditioned layer transform. We thus propose Bounded Batch Normalization (BBN), a technique that improves BN by removing this risk without sacrificing all its other benefits (Section 5). BBN achieves this by simply bounding the values of BN parameters during training.

We present benchmark image classification experiments using both ConvNets [22] and modern network architectures [8, 27, 25] (Section 6). Our results show that SVB indeed improves over SGD based methods for training various architectures of deep networks, and in many cases with a large margin. Our proposed BBN further improves over BN. In particular, we achieve the state-of-the-art results of 3.06% error rate on CIFAR10 and 16.90% on CIFAR100 [13], using off-the-shelf network architectures (Wide ResNets [27]). Our preliminary results on the large-scale ImageNet dataset are consistent with those on moderate-scale ones.

2. Related works

In this section, we briefly review the closely related deep learning methods that also pay attention to the properties of network solutions.

Saxe *et al.* [21] theoretically study the gradient descent learning dynamics of deep linear networks, and give similar empirical insights for deep nonlinear networks. They further suggest that using orthogonal initialization of weight matrices can achieve learning efficiency similar to that of unsupervised pre-training. Mishkin and Matas [17] present promising results on image classification, using the orthogonal initialization idea in [21]. Our theoretical analysis in Section 4 follows [21], but are different in the following aspects. We focus on studying the conditions when network training converges, while [21] focuses on the conditions right after network initialization. Our analysis centers around our proposed SVB method, and we discuss how SVB can resolve the issues that appear as the network training proceeds. We also extend our theoretical analysis to BN [11], and propose a new BBN method that improves over BN for training modern deep networks.

Arpit *et al.* [4] also study the properties of network parameters that can have good performance, but from a signal recovery point of view. In particular, they study the reverse data-generating properties of auto-encoders where input samples are generated from the true signals of hidden representations. They prove that for sparse true signals, e.g., those out of ReLU activations, strong recovery can be achieved if the weight matrix is highly incoherent. Different from [4], our main concern is on the properties of network parameters that can give good image classification performance by feed-forward computations.

In [18], a soft constraint technique is proposed to deal with the vanishing gradient in training recurrent neural networks (RNNs). The soft constraint regularizes the learning of weight matrices so that those better to achieve norm preservation of error signals across layers are favored. In contrast, our proposed SVB method directly controls the singular values of weight matrices, and norm preservation of error signals is only part of our benefits.

A recent work from Wisdom and Powers *et al.* [26] shows full-capacity unitary recurrence matrices can be used in RNNs, and can be optimized over the differentiable manifold of unitary matrices, which improves over [2]. In contrast, we focus on convolutional networks in this work, where weight matrices are not square. We only enforce column or row vectors of weight matrices of ConvNets to be near orthogonal, while giving them more flexibility to better learn to the training tasks. This relaxation from strict orthogonality enables us to use very simple algorithms compatible with standard SGD based training. Practicably, we observe our SVB algorithm is just as efficient as SGD based training, while achieving the property of near orthogonality.

3. The proposed Singular Value Bounding algorithm

Suppose we have K pairs of training samples $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^K$, where $\boldsymbol{x}_i \in \mathbb{R}^{N_x}$ is a training input and y_i is its corresponding output. $\boldsymbol{y}_i \in \mathbb{R}^{N_y}$ could be a vector with continuous entries for regression problems, or a binary one-hot vector for classification problems. A deep neural network of L layers performs cascaded computations of $\boldsymbol{x}^l = f(\boldsymbol{z}^l) = f(\boldsymbol{W}^l \boldsymbol{x}^{l-1} + \boldsymbol{b}^l) \in \mathbb{R}^{N_l}$ for $l = 1, \ldots, L$, where $\boldsymbol{x}^{l-1} \in \mathbb{R}^{N_{l-1}}$ is the input feature of the l^{th} layer, $f(\cdot)$ is an element-wise activation function, and $\boldsymbol{W}^l \in \mathbb{R}^{N_l \times N_{l-1}}$ and $\boldsymbol{b}^l \in \mathbb{R}^{N_l}$ are respectively the layer-wise weight matrix and bias vector. We have $\boldsymbol{x}^0 = \boldsymbol{x}$. With appropriate training criteria, network optimization aims to find solutions of network parameters $\Theta = \{\boldsymbol{W}^l, \boldsymbol{b}^l\}_{l=1}^L$, so that the trained network is able to produce good estimation of $\boldsymbol{y}$ for any test sample $\boldsymbol{x}$. Training is usually based on SGD or its variants [24]. Given the training loss function $\mathcal{L}\left(\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^K; \Theta\right)$, SGD updates Θ based on a simple rule of $\Theta_{t+1} \leftarrow \Theta_t - \eta \frac{\partial \mathcal{L}}{\partial \Theta_t}$, where η is the learning rate. The gradient $\frac{\partial \mathcal{L}}{\partial \Theta_t}$ is usually computed from a mini-batch of training samples. Network training

proceeds by sampling for each iteration t a mini-batch from $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^K$, until a specified number T of iterations or the training loss plateaus.

Existing deep learning research suggests that in order to get good performance, initializations of Θ matter. In particular, scaled random Gaussian matrices are proposed in [6] as the initializations of weight matrices $\{\boldsymbol{W}^l\}_{l=1}^L$, and random orthogonal ones are advocated in [21, 17]. Given different initializations, these methods train deep networks using SGD or its variants. Theoretical analysis in [21] and empirical results in [17] demonstrate some advantages of orthogonal initializations over Gaussian ones. In this work, we are interested in pushing a step further to know what solutions of Θ matter when network training converges, rather than just at the initialization. For the orthogonal case, our empirical results (cf. Figure 1) show that as the training proceeds, singular value spectra of weight matrices diverge from their initial condition. We are thus motivated to investigate along this line, from the empirical observations of Figure 1 and also the theoretical analysis in [21].

More specifically, we propose a simple yet very effective network training method, which preserves the orthogonality of weight matrices during the procedure of network training. This amounts to solving the following constrained optimization problem

$$\begin{aligned} \min_{\Theta=\{\boldsymbol{W}^l,\boldsymbol{b}^l\}_{l=1}^L} & \mathcal{L}\left(\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^K; \Theta\right) \\ \text{s.t. } & \boldsymbol{W}^l \in \mathcal{O}\ \forall\, l \in \{1, \ldots, L\}, \end{aligned} \tag{1}$$

where $\mathcal{O}$ stands for the set of matrices whose row or column vectors are orthonormal. Compared with standard SGDs, the feasible set of problem (1) for $\{\boldsymbol{W}^l\}_{l=1}^L$ is much reduced. For $\boldsymbol{W}^l$ of any l^{th} layer, problem (1) in fact constrains its solution set as a Riemannian manifold called Stiefel manifold [1]. In this work, we consider near orthonormality of $\{\boldsymbol{W}^l\}_{l=1}^L$, and propose to approximately solve this problem based on projected SGD (or its variants): we simply bound, after every T_{svb} iterations of SGD training, the singular values of each $\boldsymbol{W}^l$, for $l = 1, \ldots, L$, in a narrow band $[1/(1+\epsilon), (1+\epsilon)]$ around the value of 1, where ϵ is a specified small constant. Algorithm 1 presents details of our proposed Singular Value Bounding (SVB) method. In Section 4, we present theoretical analysis on deep linear networks to justify the advantages of our proposed SVB on *forward propagation to achieve training objectives*, and *backward propagation of training errors*.

Empirical computation cost Applying SVB to network training amounts to solving singular value decompositions (SVD) for weight matrices of all the network layers. We note that this cost can be amortized by doing SVB every T_{svb} number of iterations. We usually apply SVB once every epoch of SGD training. When the set of training samples is huge (e.g., the ImageNet dataset), the wall-clock time caused by SVB computation is practically negligible; in fact, we often observe even faster training when using SVB, possibly due to the better conditioning of weight matrices resulting from SVB.

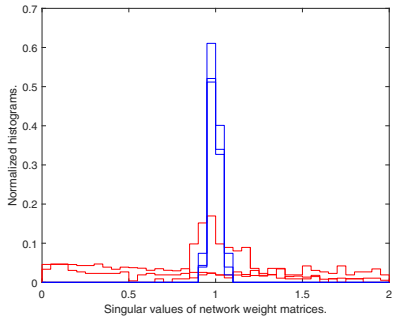

Figure 1. Normalized singular value histograms for weight matrices of a 38-layer ConvNet trained for CIFAR10 image classification (cf. Section 6.1 for details of the network architecture). Red stairs are from SGD based training, and blue ones are from our proposed SVB method. For both methods, three histograms respectively for lower, middle, and higher network layers are counted when network training converges from orthogonal initializations. Given the sharp difference of singular value spectra between the two methods, it is interesting to observe that both methods give reasonably good performance, and our method even outperforms SGD based training.

Algorithm 1: Singular Value Bounding

input : A network of L layers with trainable parameters $\Theta = \{\boldsymbol{W}^l, \boldsymbol{b}^l\}_{l=1}^L$, training loss $\mathcal{L}$, learning rate η, the maximal number T of training iterations, a specified number T_{svb} of iteration steps, a small constant ϵ

Initialize Θ such that $\boldsymbol{W}^{l\top}\boldsymbol{W}^l = \boldsymbol{I}$ or $\boldsymbol{W}^l\boldsymbol{W}^{l\top} = \boldsymbol{I}$ for $l = 1, \ldots, L$
for $t = 0, \ldots, T-1$ **do**
  Update $\Theta_{t+1} \leftarrow \Theta_t - \eta \frac{\partial \mathcal{L}}{\partial \Theta_t}$ using SGD based methods
  while *training proceeds for every* T_{svb} *iterations* **do**
    for $l = 1, \ldots, L$ **do**
      Perform $[\boldsymbol{U}^l, \boldsymbol{S}^l, \boldsymbol{V}^l] = \text{svd}(\boldsymbol{W}^l)$
      Let $\{s_i^l\}_{i=1}^{N_l}$ be the diagonal entries of $\boldsymbol{S}^l$
      for $i = 1, \ldots, N_l$ **do**
        $s_i^l = 1 + \epsilon$ if $s_i^l > 1 + \epsilon$
        $s_i^l = 1/(1+\epsilon)$ if $s_i^l < 1/(1+\epsilon)$
      end
    end
    if *network contains BN layers* **then**
      Use BBN of Algorithm 2 to update BN parameters
    end
  end
end
output: Trained network with parameters Θ_T for inference

4. Propagations of all directions of variations with Singular Value Bounding

In this section, we present theoretical analysis on deep linear networks to discuss the importance of forward-

propagating *all the directions* of training objectives and backward-propagating those of training errors, in order to better train deep neural networks. Our analyses resemble, but are different from, those in [21] (cf. Section 2 for details of the difference). These analyses justify our proposed SVB algorithm, and are supported by the experimental results reported in Section 6.

4.1. The forward propagation

We start our analysis of optimal network solutions with a simple two-layer linear network that computes $\boldsymbol{W}^2\boldsymbol{W}^1\boldsymbol{x}$, where we have used a linear activation $f(\boldsymbol{z}) = \boldsymbol{z}$ and ignored the bias for simplicity. Using squared Euclidean distance as the training criterion gives the following loss function $\mathcal{L} = \frac{1}{2K}\sum_{i=1}^{K} \|\boldsymbol{y}_i - \boldsymbol{W}^2\boldsymbol{W}^1\boldsymbol{x}_i\|_2^2$. To minimize $\mathcal{L}$ with respect to (w.r.t.) $\boldsymbol{W}^1$ and $\boldsymbol{W}^2$, we note that the optimal solutions are characterized by the gradients

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^1} &= \boldsymbol{W}^{2\top}\left(\boldsymbol{C}^{yx} - \boldsymbol{W}^2\boldsymbol{W}^1\boldsymbol{C}^{xx}\right)\\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^2} &= \left(\boldsymbol{C}^{yx} - \boldsymbol{W}^2\boldsymbol{W}^1\boldsymbol{C}^{xx}\right)\boldsymbol{W}^{1\top},\end{aligned} \tag{2}$$

where $\boldsymbol{C}^{yx} = \frac{1}{K}\sum_{i=1}^{K}\boldsymbol{y}_i\boldsymbol{x}_i^\top$ and $\boldsymbol{C}^{xx} = \frac{1}{K}\sum_{i=1}^{K}\boldsymbol{x}_i\boldsymbol{x}_i^\top$. When training deep networks, the input samples $\{\boldsymbol{x}_i\}_{i=1}^K$ are usually pre-processed by whitening, i.e., each $\boldsymbol{x}_i$ has zero mean and $\boldsymbol{C}^{xx} = \boldsymbol{I}$. With input data whitening, $\boldsymbol{C}^{yx}$ is in fact the cross-covariance matrix between input and output training samples, which models how the input variations relate to those of the outputs. Thus $\boldsymbol{C}^{yx}$ contains all the information that determines the learning results of (2) w.r.t. $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$. Applying SVD to $\boldsymbol{C}^{yx}$ gives $\boldsymbol{C}^{yx} = \boldsymbol{U}^y\boldsymbol{S}^{yx}\boldsymbol{V}^{x\top}$, where the orthogonal matrix $\boldsymbol{U}^y \in \mathbb{R}^{N_y\times N_y}$ contains columns of singular vectors in the output space that represent *independent directions of output variations*, the orthogonal matrix $\boldsymbol{V}^x \in \mathbb{R}^{N_x\times N_x}$ contains columns of singular vectors in the input space that represent *independent directions of input variations*, and $\boldsymbol{S}^{yx} \in \mathbb{R}^{N_y\times N_x}$ is a diagonal matrix with ordered singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(N_x,N_y)}$.

As suggested in [21], when we initialize $\boldsymbol{W}^1$ and $\boldsymbol{W}^2$ as

$$\boldsymbol{W}^1 = \boldsymbol{R}\boldsymbol{S}^1\boldsymbol{V}^{x\top},\ \ \boldsymbol{W}^2 = \boldsymbol{U}^y\boldsymbol{S}^2\boldsymbol{R}^\top, \tag{3}$$

where $\boldsymbol{R} \in \mathbb{R}^{N_1\times N_1}$ is an arbitrary orthogonal matrix and $\boldsymbol{S}^1$ and $\boldsymbol{S}^2$ are diagonal matrices with nonnegative entries, and keep $\boldsymbol{R}$ fixed during optimization, the gradients (2) at optimal solutions can be derived as

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^1} &= \boldsymbol{R}\boldsymbol{S}^{2\top}\left(\boldsymbol{S}^{yx} - \boldsymbol{S}^2\boldsymbol{S}^1\right)\boldsymbol{V}^{x\top}\\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^2} &= \boldsymbol{U}^y\left(\boldsymbol{S}^{yx} - \boldsymbol{S}^2\boldsymbol{S}^1\right)\boldsymbol{S}^{1\top}\boldsymbol{R}^\top.\end{aligned} \tag{4}$$

Since $\boldsymbol{R}$ is fixed, the above conditions ensure that $\boldsymbol{W}^1$ and $\boldsymbol{W}^2$ are optimized along their respective independent directions of variations. Denote s_m and t_m, $m = 1,\ldots,\min(N_y,N_1,N_x)$, are the m^{th} diagonal entries of $\boldsymbol{S}^1$ and $\boldsymbol{S}^2$ respectively. By change of optimization variables, (2) can be further simplified as the following equations for each m^{th} direction of variations

$$\frac{\partial \mathcal{L}}{\partial s_m} = (\sigma_m - s_m t_m)\, t_m,\ \ \frac{\partial \mathcal{L}}{\partial t_m} = (\sigma_m - s_m t_m)\, s_m. \tag{5}$$

In fact, the gradients (5) w.r.t. s_m and t_m arise from the following energy function

$$\mathcal{E}(s_m, t_m) = \frac{1}{2}\left(\sigma_m - s_m t_m\right)^2, \tag{6}$$

showing that the product of optimal pairs s_m and t_m approaches σ_m.

We subsequently extend the analysis from (2) to (6) for a deep linear network of L layers. With the same loss function $\mathcal{L}$ of squared Euclidean distance, the optimal weight matrix $\boldsymbol{W}^l$ of the l^{th} layer is characterized by the gradient

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^l} = \left(\prod_{i=l+1}^{L}\boldsymbol{W}^i\right)^\top\left(\boldsymbol{C}^{yx} - \prod_{i=1}^{L}\boldsymbol{W}^i\right)\left(\prod_{i=1}^{l-1}\boldsymbol{W}^i\right)^\top, \tag{7}$$

where $\prod_{i=l}^{l'}\boldsymbol{W}^i = \boldsymbol{W}^{l'}\boldsymbol{W}^{l'-1}\cdots\boldsymbol{W}^l$ with the special case that $\prod_{i=l}^{l'}\boldsymbol{W}^i = \boldsymbol{I}$ when $l > l'$, and we have assumed in (7) that $\boldsymbol{C}^{xx} = \boldsymbol{I}$. Similar to (3), when we initialize weight matrices of the deep network as $\boldsymbol{W}^l = \boldsymbol{R}^{l+1}\boldsymbol{S}^l\boldsymbol{R}^{l\top}$ for any $l \in \{1,\ldots,L\}$, where each $\boldsymbol{R}^l$ is an orthogonal matrix with the special cases that $\boldsymbol{R}^1 = \boldsymbol{V}^x$ and $\boldsymbol{R}^{L+1} = \boldsymbol{U}^y$, and each $\boldsymbol{S}^l$ is an diagonal matrix with nonnegative entries, and keep $\{\boldsymbol{R}^l\}_{l=1}^{L+1}$ fixed during optimization [1], the gradient (7) at optimal solutions can be derived as

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^l} = \boldsymbol{R}^{l+1}\left(\prod_{i=l+1}^{L}\boldsymbol{S}^i\right)^\top\left(\boldsymbol{S}^{yx} - \prod_{i=1}^{L}\boldsymbol{S}^i\right)\left(\prod_{i=1}^{l-1}\boldsymbol{S}^i\right)^\top\boldsymbol{R}^{l\top}. \tag{8}$$

By change of optimization variables, (8) can be further simplified as the following independent gradient for the m^{th} direction of variations with $m \leq M = \min(N_y,\ldots,N_l,\ldots,N_x)$

$$\frac{\partial \mathcal{L}}{\partial s_m^l} = \prod_{i=l+1}^{L}s_m^i\left(\sigma_m - \prod_{i=1}^{L}s_m^i\right)\prod_{i=1}^{l-1}s_m^i, \tag{9}$$

[1] Alternatively, one might relax this constraint and update $\{\boldsymbol{W}^l\}_{l=1}^L$ using standard methods such as SGD, and change the left and right singular vectors of each updated $\boldsymbol{W}^l$ to satisfy $\boldsymbol{W}^l = \boldsymbol{R}^{l+1}\boldsymbol{S}^l\boldsymbol{R}^{l\top}$ (with varying sets of $\{\boldsymbol{R}^l\}_{l=1}^{L+1}$). However, this would cause mixing of different directions in the connecting output/input spaces across layers.

which turns out to be the gradient of the energy function

$$\mathcal{E}(s_m^1, \ldots, s_m^L) = \frac{1}{2}\left(\sigma_m - \prod_{l=1}^{L} s_m^l\right)^2. \tag{10}$$

The positive scalar σ_m in (10) represents *the strength of the m^{th} direction of input-output correlations*. It is usually fixed given provided training data. To characterize the conditions under which the minimum energy of (10) can be achieved, denote $s_m^{l\max} = \max(s_m^1, \ldots, s_m^L)$ and $s_m^{l\min} = \min(s_m^1, \ldots, s_m^L)$. One can easily prove that when $L \to \infty$, it is *necessary* that $s_m^{l\max} > 1$ and $s_m^{l\min} < 1$. Conversely, the *sufficient* conditions for *not achieving* the minimum energy of (10) are either $s_m^{l\max} < 1$ or $s_m^{l\min} > 1$, when $L \to \infty$.

For any fixed and finite σ_m, our proposed SVB algorithm is potentially able to achieve the minimum energy of (10) (although it does not meet the assumptions used to derive (10)), by choosing an appropriate value of ϵ so that values of $\{s_m^l\}_{l=1}^L$ are properly learned to range in a narrow band $[1/(1+\epsilon), 1+\epsilon]$. This applies to any of the M directions of input-output correlations. Existing network training methods have no such constraints, and $\{s_m^l\}$ of all layers/directions are free to be scaled up or down, resulting in very uneven magnitude distribution of $\{\{s_m^l\}_{l=1}^L\}_{m=1}^M$. Consequently, training easily falls in local minima that minimize (10) for certain directions, but not for all of the M ones. And only parts of the input-output correlations are taken into account during learning.

Our derivation from (7) to (8) requires that the output singular vectors of the weight matrix of layer l be the input singular vectors of that of layer $l+1$. However, it does not hold true in the SGD based Algorithm 1, where weight matrices are updated without such constraints. Consider a two-layer basic component $\boldsymbol{W}^{l+1}\boldsymbol{W}^l$ in (7), which propagates signal activations (and hence information of input variations) from layer l to layer $l+1$. After SGD updating, Algorithm 1 computes SVDs of the updated $\boldsymbol{W}^{l+1}$ and $\boldsymbol{W}^l$, resulting in $\boldsymbol{W}^{l+1}\boldsymbol{W}^l = \boldsymbol{U}^{l+1}\boldsymbol{S}^{l+1}\boldsymbol{V}^{l+1\top}\boldsymbol{U}^l\boldsymbol{S}^l\boldsymbol{V}^{l\top}$. While one may initialize $\boldsymbol{W}^{l+1}$ and $\boldsymbol{W}^l$ such that $\boldsymbol{V}^{l+1} = \boldsymbol{U}^l$, after SGD updating, they are generally not equal. Denote $\boldsymbol{M} = \boldsymbol{S}^{l+1}\boldsymbol{V}^{l+1\top}\boldsymbol{U}^l\boldsymbol{S}^l$, we have

$$\boldsymbol{M}_{m,m'} = s_m^{l+1} s_{m'}^l \boldsymbol{v}_m^{l+1\top}\boldsymbol{u}_{m'}^l, \tag{11}$$

where $\boldsymbol{M}_{m,m'}$ is the (m,m') entry of $\boldsymbol{M}$, $\boldsymbol{v}_m^{l+1}$ is the m^{th} column of $\boldsymbol{V}^{l+1}$, $\boldsymbol{u}_{m'}^l$ is the m'^{th} column of $\boldsymbol{U}^l$, and s_m^{l+1} and $s_{m'}^l$ are respectively the m^{th} and m'^{th} singular values of $\boldsymbol{S}^{l+1}$ and $\boldsymbol{S}^l$. By projecting $\boldsymbol{u}_{m'}^l$ onto $\boldsymbol{v}_m^{l+1}$, $\boldsymbol{v}_m^{l+1\top}\boldsymbol{u}_{m'}^l$ represents the *mixing* of the m'^{th} direction of variations in the output space of layer l with the m^{th} one in the input space of layer $l+1$. By bounding s_m^{l+1} and $s_{m'}^l$, our proposed SVB algorithm controls both the *independent* (when $m = m'$ and the assumptions from (7) to (8) hold), and the *mixing* strengths of propagation across layers. Without such constraints, some directions of variations could be over-amplified while others are strongly attenuated, when signals are propagated from lower layers to higher layers.

4.2. The backward propagation

For a deep linear network that performs cascaded computations of $\boldsymbol{x}^l = \boldsymbol{W}^l\boldsymbol{x}^{l-1}$ for $l = 1, \ldots, L$, the gradient of loss function $\mathcal{L}$ w.r.t. the output activation $\boldsymbol{x}^l$ of layer l is written as

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^l} = \left(\frac{\partial \boldsymbol{x}^L}{\partial \boldsymbol{x}^l}\right)^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^L} = \left(\prod_{i=l+1}^{L} \boldsymbol{W}^i\right)^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^L}, \tag{12}$$

where $\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^L}$ contains the error vector for back-propagation. For any $i \in \{l+1, \ldots, L\}$, assume $\boldsymbol{W}^i$ satisfies the condition $\boldsymbol{W}^i = \boldsymbol{R}^{i+1}\boldsymbol{S}^i\boldsymbol{R}^{i\top}$ that we have used to derive from (7) to (10), with analysis similar to the forward propagation case, we have

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^l} = \left(\sum_{m=1}^{M}\left(\prod_{i=l+1}^{L} s_m^i\right)\boldsymbol{r}_m^{L+1}\boldsymbol{r}_m^{l+1\top}\right)^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^L}, \tag{13}$$

where $\boldsymbol{r}_m^{L+1}$ (or $\boldsymbol{r}_m^{l+1}$) denotes the m^{th} column of $\boldsymbol{R}^{L+1}$ (or $\boldsymbol{R}^{l+1}$), and $M = \min(N_L, \ldots, N_l)$. As the network goes deep (i.e., L becomes large), $\prod_{i=l+1}^{L} s_m^i$ would either explode or vanish if $\{s_m^i\}_{i=l+1}^L$ do not satisfy a necessary condition similar to the one for achieving the minimum of (10). Consequently, the m^{th} *component* error vector $\left(\prod_{i=l+1}^{L} s_m^i\right)\boldsymbol{r}_m^{l+1}\boldsymbol{r}_m^{L+1\top}\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^L}$ in (13) would either explode or vanish. When $\epsilon \to 0$ in Algorithm 1, our proposed method guarantees that all the M components of the error vector would propagate to lower layers without attenuation or explosion. In the ideal case of $N_L = \cdots = N_l$, our method also guarantees that $\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^l}\|_2 = \|\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^L}\|_2$, i.e., to preserve the norm of error vector. Without such constraints on singular values of $\{\boldsymbol{W}^i\}_{i=l+1}^L$, it is still possible that the norm of error vector is preserved by amplifying some singular values while shrinking others, as the way advocated in [6]. However, its norm preservation is achieved in a rather anisotropic way.

5. Compatibility with Batch Normalization

In this section, we investigate how our proposed network training algorithm could be compatible with Batch Normalization [11]. BN addresses a network training issue called *internal covariate shift*, which slows down the training since distributions of each layer's inputs keep changing during the training process. BN alleviates this issue by inserting into network trainable normalization layers, which normalize each layer's neuron activations as zero mean and unit variance in a mini-batch and neuron-wise manner.

Formally, for a network layer computing $f(\boldsymbol{z}) = f(\boldsymbol{W}\boldsymbol{x}) \in \mathbb{R}^N$, BN inserts a normalization layer before the activation function, giving the new layer $f(\text{BN}(\boldsymbol{z})) = f(\text{BN}(\boldsymbol{W}\boldsymbol{x}))$, where we have ignored the bias term for simplicity. BN in fact applies the following linear transformation to $\boldsymbol{z}$

$$\text{BN}(\boldsymbol{z}) = \Gamma\Sigma(\boldsymbol{z} - \boldsymbol{\mu}) + \boldsymbol{\beta}, \tag{14}$$

where each entry of $\boldsymbol{\mu} \in \mathbb{R}^N$ is the output mean at each of the N neurons of the layer, the diagonal matrix $\Sigma \in \mathbb{R}^{N\times N}$ contains entries $\{1/\varsigma_i\}_{i=1}^N$ that is the inverse of the neuron-wise output standard deviation ς_i (obtained by adding a small constant to the variance for numerical stability), $\Gamma \in \mathbb{R}^{N\times N}$ is a diagonal matrix containing trainable scalar parameters $\{\gamma_i\}_{i=1}^N$, and $\boldsymbol{\beta} \in \mathbb{R}^N$ is a trainable bias term. Note that during training, μ and ς for each neuron are computed using mini-batch samples, and during inference they are fixed representing the statistics of all the training population, which are usually obtained by running average. Thus the computation (14) for each sample is deterministic after network training.

Inserting $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$ into (14) we get

$$\text{BN}(\boldsymbol{x}) = \widetilde{\boldsymbol{W}}\boldsymbol{x} + \tilde{\boldsymbol{b}} \text{ s.t. } \widetilde{\boldsymbol{W}} = \Gamma\Sigma\boldsymbol{W} \;\; \tilde{\boldsymbol{b}} = \boldsymbol{\beta} - \Gamma\Sigma\boldsymbol{\mu}, \tag{15}$$

which is simply a standard network layer with change of variables. The following lemma suggests that we may bound the entries $\{\gamma_i/\varsigma_i\}_{i=1}^N$ of the product of the diagonal matrices Γ and Σ, to make our proposed SVB algorithm be compatible with BN.

Lemma 1 *For a matrix $\boldsymbol{W} \in \mathbb{R}^{M\times N}$ with singular values of 1, and a diagonal matrix $\boldsymbol{G} \in \mathbb{R}^{M\times M}$ with nonzero entries $\{g_i\}_{i=1}^M$, let $g_{\max} = \max(|g_1|, \ldots, |g_M|)$ and $g_{\min} = \min(|g_1|, \ldots, |g_M|)$, the singular values of $\widetilde{\boldsymbol{W}} = \boldsymbol{G}\boldsymbol{W}$ is bounded in $[g_{min}, g_{\max}]$. When $\boldsymbol{W}$ is fat, i.e., $M \leq N$, and rank$(\boldsymbol{W}) = M$, singular values of $\widetilde{\boldsymbol{W}}$ are exactly $\{|g_i|\}_{i=1}^M$.*

Proof of the lemma is presented in the supplemental material. Lemma 1 suggests that for a deep network with BN layers, the trainable parameters $\{\gamma_i\}_{i=1}^N$, together with sample statistics $\{\varsigma_i\}_{i=1}^N$, could change the conditioning of layer transform, and consequently the behaviors of signal propagation across network layers. In particular, when absolute values $\{|\gamma_i/\varsigma_i|\}_{i=1}^N$ of the diagonal entries of $\Gamma\Sigma$ *for all the network layers simultaneously drift up or down* away from the value of 1, signal propagation would be susceptible to explosion or attenuation when the network goes deep. One direct way to remove this risk is to control the values of $\{\gamma_i/\varsigma_i\}_{i=1}^N$, e.g., to let them be around 1. However, this would also remove an important benefit of BN. More specifically, the introduction of trainable scaling parameters $\{\gamma_i\}_{i=1}^N$ in BN is to make sure that after neuron-wise normalization by $\{\varsigma_i\}_{i=1}^N$ (and $\{\mu_i\}_{i=1}^N$), the change to layer outputs is compensated by $\{\gamma_i\}_{i=1}^N$, so that the BN transform is overall an *identity transform* [11]. One might expect that the value of each ς_i in Σ is similar to that of the corresponding γ_i in Γ. However, this is not the case in practice. In fact, $\{\gamma_i\}_{i=1}^N$ bring additional and significant benefits to training of deep neural networks: the decoupled $\{\gamma_i\}_{i=1}^N$ enable scales of the magnitude of features at different network layers become freely adjustable for better training objectives. Inspired by this scheme of BN, we introduce a decoupled scalar α from $\Gamma\Sigma$, and propose to control the re-scaled version $\{\frac{1}{\alpha}\gamma_i/\varsigma_i\}_{i=1}^N$, instead of $\{\gamma_i/\varsigma_i\}_{i=1}^N$, to make our proposed SVB be compatible with BN. We set $\alpha = \frac{1}{N}\sum_{i=1}^N \gamma_i/\varsigma_i$ during network training. Note that re-scaling the magnitude scales of features at different layers is equivalent to simultaneously scaling up $\{s_m^l\}_{m=1}^M$ of all the M directions in (10) for certain layers, while simultaneously scaling down for other layers, and this does not cause sacrifice of propagation of certain directions of input-output correlations. Algorithms 2 presents our improved BN transform called *Bounded Batch Normalization (BBN)*. We note that in Algorithm 2, we do not take the absolute values. This is because values of $\{\gamma_i\}_{i=1}^N$ are usually initialized as 1, and they are empirically observed to keep positive during the process of network training. Experiments in Section 6 show that image classification results are improved when using BBN instead of BN, demonstrating a consistency between our theoretical analysis and practical results.

Algorithm 2: Bounded Batch Normalization

input : A network with L BN layers, trainable parameters $\{\Gamma_t^l\}_{l=1}^L$, $\{\boldsymbol{\beta}_t^l\}_{l=1}^L$, and statistics $\{\boldsymbol{\mu}_t^l\}_{l=1}^L$, $\{\Sigma_t^l\}_{l=1}^L$ of BN layers at iteration t, a small constant $\tilde{\epsilon}$

Update to get $\{\Gamma_{t+1}^l\}_{l=1}^L$ from $\{\Gamma_t^l\}_{l=1}^L$ (and $\{\boldsymbol{\beta}_{t+1}^l\}_{l=1}^L$ from $\{\boldsymbol{\beta}_t^l\}_{l=1}^L$), using SGD based methods

Update to get $\{\Sigma_{t+1}^l\}_{l=1}^L$ from $\{\Sigma_t^l\}_{l=1}^L$ (and $\{\boldsymbol{\mu}_{t+1}^l\}_{l=1}^L$ from $\{\boldsymbol{\mu}_t^l\}_{l=1}^L$), using running average over statistics of mini-batch samples

for $l = 1, \ldots, L$ **do**

  Let $\{\gamma_i\}_{i=1}^{N_l}$ and $\{1/\varsigma_i\}_{i=1}^{N_l}$ be respectively the diagonal entries of Γ_{t+1}^l and Σ_{t+1}^l

  Let $\alpha = \frac{1}{N_l}\sum_{i=1}^{N_l} \gamma_i/\varsigma_i$

  for $i = 1, \ldots, N_l$ **do**

    $\gamma_i = \alpha\varsigma_i(1+\tilde{\epsilon})$ if $\frac{1}{\alpha}\gamma_i/\varsigma_i > 1+\tilde{\epsilon}$

    $\gamma_i = \alpha\varsigma_i/(1+\tilde{\epsilon})$ if $\frac{1}{\alpha}\gamma_i/\varsigma_i < 1/(1+\tilde{\epsilon})$

  end

end

output: Updated BN parameters and statistics at iteration $t+1$

6. Experiments

We present image classification results to show the efficacy of our proposed SVB and BBN. We use benchmark datasets of CIFAR10, CIFAR100 [13], and ImageNet [20]. CIFAR10 is intensively used for our controlled studies. We investigate how SVB and BBN perform on standard Con-

vNets, and also the modern architectures of (pre-activation versions of) ResNets [8] and Wide ResNets [27].

We use BN layers or our proposed BBN layers in all networks. Training is based on SGD with momentum using softmax loss function. We initialize networks using orthogonal weight matrices (cf. Algorithm 1). Except experiments reported in Table 3, all other experiments are based on a mini-batch size of 128, momentum of 0.9, and weight decay of 0.0001; the learning rate starts from 0.5 and ends at 0.001, and decays every two epochs until the end of 160 epoches of training. When SVB is turned on, we apply it to weight matrices of all layers [2] after every epoch of training.

6.1. Controlled studies using ConvNets

In this section, we use ConvNets to study the behaviors of our proposed SVB algorithm on deep network training. We choose modern convolutional architectures from [22, 7]. The networks start with a conv layer of 16 3×3 filters, and then sequentially stack three types of $2X$ conv layers of 3×3 filters, each of which has the feature map sizes of 32, 16, and 8, and filter numbers 16, 32, and 64, respectively. Spatial sub-sampling of feature maps is achieved by conv layers of stride 2. The networks end with a global average pooling and fully-connected layers. Thus for each network we have $6X + 2$ weight layers in total. We consider $X = 3$ and $X = 6$ in our studies. The used CIFAR10 dataset consists of 10 object categories of $60,000$ color images of size 32×32 ($50,000$ training and $10,000$ testing ones). We use raw images without pre-processing. Data augmentation follows the standard manner in [14]: during training, we zero-pad 4 pixels along each image side, and sample a 32×32 region crop from the padded image or its horizontal flip; during testing, we use the original non-padded image.

Figure 2 shows that for each depth case, our results using SVB are consistently better than those from standard SGD with momentum, verifying that bounding the singular values of weight matrices indeed improves the conditioning of layer transform. Replacing BN with BBN gives similar performance. This shows that in the case of plain ConvNets, the issue of ill-conditioning caused by BN is not severe. In the more complex ResNet type architectures, BBN improves over BN effectively, as presented shortly. Comparative results in Figure 2 also suggest that with the increase of network layers, training becomes more difficult: results of deeper network ($X = 6$) are worse than those of shallower one ($X = 3$). This is consistent with the observation in [7].

6.2. Ablation studies using ResNet

We conduct experiments to investigate whether our proposed SVB and BBN methods are effective for "residual learning" [7]. We use an architecture similar to those presented in Section 4.2 in [7], but change it to the pre-activation version [8]. The network construction is based on the ConvNets presented in Section 6.1, and we use an "identify shortcut" to connect every two conv layers of 3×3 filters, and use a "projection shortcut" when sub-sampling of feature maps is needed. We use a network of $X = 11$ for experiments in this section, which gives 68 weight layers.

Table 1. Ablation studies on CIFAR10, using a pre-activation ResNet with 68 weight layers of 3×3 convolutional filters. We run each setting for 5 times, using standard data augmentation [14]. Results are in the format of best (mean + std).

Training methods	Error rate (%)
SGD with momentum + BN	6.10 (6.22 ± 0.14)
SVB + BN	5.65 (5.79 ± 0.10)
SVB + BBN	**5.37** (5.49 ± 0.11)

We conduct ablation studies by switching SVB on or off, and switching BBN on or off. The parameters ϵ and $\tilde{\epsilon}$ are fixed as 0.5 and 1 respectively. All experiments are run for 5 times, and we report the best, mean, and standard deviation results in Table 1. These results show that using SVB improves deep residual learning, and BBN further improves over standard BN, demonstrating the efficacy of our proposed methods for modern deep architectures.

6.3. Comparisons with the state-of-the-art

We use Wide ResNet [27] to compare with the state-of-the-art results on CIFAR10 and CIFAR100. The CIFAR100 dataset has the same number of 32×32 color images as CIFAR10 does, but it has 100 object categories and each category contains one tenth images of those of CIFAR10. We use raw data without pre-processing, and do data augmentation in the same way as for CIFAR10.

Our architecture of Wide ResNet is the same as that of "WRN-28-10" in [27], but our training hyperparameters (cf. the beginning of Section 6) are different from those in [27]. When BBN is turned on, we replace all BN layers with the BBN ones. The parameters ϵ and $\tilde{\epsilon}$ are fixed as 0.5 and 1 respectively. Without using SVB and BBN, our Wide ResNet gives an error rate of 4.50 on CIFAR10 and 20.78 on CIFAR100. With SVB and BBN, the results are significantly boosted to 3.58 on CIFAR10 and 18.32 on CIFAR100.

To compare with the state-of-the-art method DenseNet [9], we use improved training hyperparameters inspired by [9]: we use the batchsize of 64 for CIFAR10 and 128 for CIFAR100, and train for an extended duration of 300 epochs; all other training parameters are the same as those described in the beginning of Section 6. We set ϵ and $\tilde{\epsilon}$ of our methods as 0.5 and 0.2 respectively. Table 3 reports the comparative results. Wide ResNet in Table 3 uses the same architecture of "WRN-28-10" as in [27], and Wider ResNet increases the widening factor from 10 to 16 (cor-

[2] When applying SVB to a conv layer, we convert its kernel tensor of the size $N_{out} \times N_{in} \times N_h \times N_w$ as a matrix of the size $N_{out} \times N_{in} N_h N_w$, where N_{out} and N_{in} denote the numbers of output and input feature channels, and N_h and N_w denote the kernel height and width, respectively.

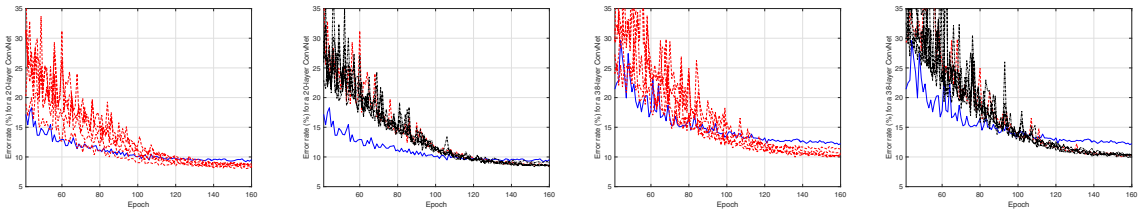

Figure 2. Validation curves on CIFAR10 using two ConvNets of 20 and 38 weight layers respectively. Blue lines are results by SGD with momentum. Red lines are results by SVB at different values of ϵ $(0.01, 0.05, 0.2, 0.5, 1)$ in Algorithm 1. Black lines are results using both SVB (fixing $\epsilon = 0.05$) and BBN at different values of $\tilde{\epsilon}$ $(0.01, 0.05, 0.2, 0.5, 1)$ in Algorithm 2. These parameter settings are simply casual choices. The left two figures are from the 20-layer ConvNet, and the right two ones are from the 38-layer ConvNet.

Table 2. Error rates (%) of different methods on CIFAR10 and CIFAR100 [13]. All methods use standard data augmentation as in [14]. A "-" indicates that result is not explicitly specified in the cited work.

Methods	CIFAR10	CIFAR100	# layers	# params
NIN [15]	8.81	-	-	-
FitNet [19]	8.39	-	19	2.5M
DSN [14]	7.97	-	-	-
Highway [23]	7.54	32.24	19	2.3M
ResNet [7]	6.43	25.16	110	1.7M
Stoc. Depth [10]	4.91	-	1202	10.2M
Pre-Act ResNet [8]	4.92	22.71	1001	10.2M
Wide ResNet [27]	4.17	20.50	28	36.5M
ResNet of ResNet [28]	3.77	19.73	-	13.3M
Our Wide ResNet W/O SVB+BBN	4.50	20.78	28	36.5M
Our Wide ResNet WITH SVB+BBN	**3.58**	**18.32**	28	36.5M

Table 3. Comparisons of error rate (%) with the state-of-the-art method DenseNet [9] on CIFAR10 and CIFAR100 [13]. Our results are obtained by using improved training hyperparameters inspired by [9]. All methods use standard data augmentation as in [14]. *We note that DenseNet practically consumes more GPU memories than Wider ResNet does.*

Methods	CIFAR10	CIFAR100	# layers	# params
DenseNet [9]	3.46	17.18	190	25.6M
Our Wide ResNet W/O SVB+BBN	3.78	19.92	28	36.5M
Our Wide ResNet WITH SVB+BBN	**3.24**	17.47	28	36.5M
Our Wider ResNet W/O SVB+BBN	3.64	19.25	28	94.2M
Our Wider ResNet WITH SVB+BBN	**3.06**	**16.90**	28	94.2M

Table 4. Error rates of single-model and single-crop testing on the ImageNet validation set.

Training methods	Top-1 error (%)	Top-5 error (%)
Our Inception-ResNet	21.61	5.91
Our Inception-ResNet WITH SVB+BN	21.20	5.57

responding to "WRN-28-16" in [27]). Our proposed SVB and BBN improve both architectures, and achieve the new state-of-the-art results of 3.06 on CIFAR10 and 16.90 on CIFAR100 [3]. These results demonstrate the great potential of SVB and BBN on training modern deep architectures.

6.4. Preliminary results on ImageNet

We present preliminary results on ImageNet [20], which has 1.28 million images of 1000 classes for training, and 50 thousand images for validation. The data augmentation scheme follows [25]. We investigate how SVB and BBN may help large-scaling learning, for which we use the pre-activation version of Inception-ResNet [25]. We use the same parameter settings as those for the CIFAR10 experiments in Section 6.3, except the learning rate that starts from 0.045. Table 4 shows that SVB and BBN indeed improves the large-scale learning, with a similar performance gain for top-1 and top-5 errors. The improvement is however lower than what we expected. We are interested for further studies in future research. We note that our architecture is almost identical to [25], but we did not manage to get the results in [25], possibly due to the different choices of gradient descent methods ([25] uses RMSProp while ours are based on SGD with momentum).

[3] In Table 3, Wider ResNet has much more model parameters than DenseNet does. However, we note that DenseNet practically consumes more GPU memories in both training and inference. This is due to the architectural design of DenseNet: in (each stage/block of) DenseNet, the input of an upper layer is formed by concatenating output feature maps of all its lower layers; thus it is easy to have bottleneck layers of tremendous memory consumption when the number of each layer's output feature maps (i.e., the growth rate in [9]) is large. This is indeed the case for model setting of the best result achieved by [9].

Acknowledgements

This work is supported in part by The Thousand Talents Plan of China (for young professionals) and Australian Research Council Projects FT-130101457, DP-140102164, LP-150100671.

References

[1] P. A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ, USA, 2007. 3

[2] M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. *CoRR*, arXiv:1511.06464, 2016. 2

[3] S. Arora, A. Bhaskara, R. Ge, and T. Ma. Provable bounds for learning some deep representations. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 584–592, 2014. 1

[4] D. Arpit, H. Q. Ngo, Y. Zhou, N. Napp, and V. Govindaraju. Towards optimality conditions for non-linear networks. *CoRR*, abs/1605.07145, 2016. 1, 2

[5] Y. N. Dauphin, R. Pascanu, Ç. Gülçehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2933–2941, 2014. 1

[6] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*, 2010. 1, 3, 5

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *arXiv prepring arXiv:1506.01497*, 2015. 1, 7, 8

[8] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision (ECCV)*, 2016. 1, 2, 7, 8

[9] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. 7, 8

[10] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. In *arXiv prepring arXiv:1603.09382*, 2016. 8

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015. 1, 2, 5, 6

[12] K. Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems (NIPS)*, 2016. 1

[13] A. Krizhevsky. Learning multiple layers of features from tiny images. *Tech. Report*, 2009. 2, 6, 8

[14] C. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS*, 2015. 7, 8

[15] M. Lin, Q. Chen, and S. Yan. Network in network. In *In Proceedings of ICLR*, 2013. 8

[16] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollr, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, 2014. 1

[17] D. Mishkin and J. Matas. All you need is a good init. *CoRR*, abs/1511.06422, 2015. 2, 3

[18] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1310–1318, 2013. 1, 2

[19] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. In *In Proceedings of ICLR*, 2015. 8

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 1, 6, 8

[21] 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. 1, 2, 3, 4

[22] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 2, 7

[23] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. *CoRR*, abs/1507.06228, 2015. 8

[24] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1139–1147, May 2013. 2

[25] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *ICLR 2016 Workshop*, 2016. 1, 2, 8

[26] S. Wisdom, T. Powers, J. R. Hershey, J. L. Roux, and L. Atlas. Full-capacity unitary recurrent neural networks. *CoRR*, arXiv:1611.00035, 2016. 2

[27] S. Zagoruyko and N. Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016. 2, 7, 8

[28] K. Zhang, M. Sun, T. X. Han, X. Yuan, L. Guo, and T. Liu. Residual networks of residual networks: Multilevel residual networks. *CoRR*, abs/1608.02908, 2016. 8