The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Orthogonal Weight Normalization: Solution to Optimization over Multiple Dependent Stiefel Manifolds in Deep Neural Networks

Lei Huang,[†] Xianglong Liu,[†] Bo Lang,[†] Adams Wei Yu,[‡] Yongliang Wang,[♮] Bo Li[♯]
[†]State Key Laboratory of Software Development Environment, Beihang University, P.R.China
[‡]Machine Learning Department, Carnegie Mellon University
[♮]JD.COM, Beijing, P.R.China
[♯]Electrical Engineering and Computer Sciences, University of California, Berkeley

Abstract

Orthogonal matrix has shown advantages in training Recurrent Neural Networks (RNNs), but such matrix is limited to be square for the hidden-to-hidden transformation in RNNs. In this paper, we generalize such square orthogonal matrix to orthogonal rectangular matrix and formulating this problem in feed-forward Neural Networks (FNNs) as Optimization over Multiple Dependent Stiefel Manifolds (OMDSM). We show that the orthogonal rectangular matrix can stabilize the distribution of network activations and regularize FNNs. We propose a novel orthogonal weight normalization method to solve OMDSM. Particularly, it constructs orthogonal transformation over proxy parameters to ensure the weight matrix is orthogonal. To guarantee stability, we minimize the distortions between proxy parameters and canonical weights over all tractable orthogonal transformations. In addition, we design orthogonal linear module (OLM) to learn orthogonal filter banks in practice, which can be used as an alternative to standard linear module. Extensive experiments demonstrate that by simply substituting OLM for standard linear module without revising any experimental protocols, our method improves the performance of the state-of-the-art networks, including Inception and residual networks on CIFAR and ImageNet datasets.

Introduction

Standard deep neural networks (DNNs) can be viewed as a composition of multiple simple nonlinear functions, each of which usually consists of one linear transformation with learnable weights or parameters followed by an element-wise nonlinearity. Such hierarchy and deep architectures equip DNNs with large capacity to represent complicated relationships between inputs and outputs. However, they also introduce potential risk of overfitting. Many methods have been proposed to address this issue, *e.g.* weight decay (Krogh and Hertz 1992) and Dropout (Srivastava et al. 2014) are commonly applied by perturbing objectives or adding random noise directly. These techniques can improve generalization of networks, but hurt optimization efficiency, which means one needs to train more epochs to achieve better performance. This naturally rises one question: is there any technique that can regularize DNNs to guarantee generalization while still guarantee efficient convergence?

To achieve this goal, we focus on the orthogonality constraint, which is imposed in linear transformation between layers of DNNs. This technique performs optimization over low embedded submanifolds, where weights are orthogonal, and thus regularizes networks. Besides, the orthogonality implies energy preservation, which is extensively explored for filter banks in signal processing and guarantees that energy of activations will not be amplified (Zhou, Do, and Kovacevic 2006) . Therefore, it can stabilize the distribution of activations over layers within DNNs (Desjardins et al. 2015; Rodríguez et al. 2017) and make optimization more efficient.

Orthogonal matrix has been actively explored in Recurrent Neural Networks (RNNs) (Arjovsky, Shah, and Bengio 2016; Wisdom et al. 2016; Vorontsov et al. 2017). It helps to avoiding gradient vanishing and explosion problem in RNNs due to its energy preservation property (Dorobantu, Stromhaug, and Renteria 2016). However, the orthogonal matrix here is limited to be square for the hidden-to-hidden transformation in RNNs. More general setting of learning *orthogonal rectangular matrix* is barely studied in DNNs (Harandi and Fernando 2016), especially in deep Convolutional Neural Networks (CNNs) (Ozay and Okatani 2016). We formulate such a problem as Optimization over Multiple Dependent Stiefel Manifolds (OMDSM), due to the fact that the weight matrix with orthogonality constraint in each layer is an embedded Stiefel Manifold (Absil, Mahony, and Sepulchre 2008) and the weight matrix in certain layer is affected by those in preceding layers in DNNs.

To solve OMDSM problem, one straightforward idea is to use Riemannian optimization method that is extensively used for single manifold or multiple independent manifolds problem, either in optimization communities (Absil and Malick 2012; Wen and Yin 2013; Cunningham and Ghahramani 2015) or in applications to the hidden-to-hidden transformation of RNNs (Wisdom et al. 2016). However, Riemannian optimization methods suffer instability in convergence or inferior performance in deep feed-forward neural networks based on our comprehensive experiments. Therefore, a stable method is highly required for OMDSM problem.

Inspired by the orthogonality-for-vectors problem (Garthwaite et al. 2012) and the fact that eigenvalue decomposition is differentiable (Ionescu, Vantzos, and Sminchisescu 2015), we propose a novel proxy parameters based solution referred to as *orthogonal weight normalization*. Specifically, we de-

vise explicitly a transformation that maps the proxy parameters to canonical weights such that the canonical weights are orthogonal. Updating is performed on the proxy parameters when gradient signal is ensured to back-propagate through the transformation. To guarantee stability, we minimize the distortions between proxy parameters and canonical weights over all tractable orthogonal transformations.

Based on *orthogonal weight normalization*, we design orthogonal linear module for practical purpose. This module is a linear transformation with orthogonality, and can be used as an alternative of standard linear modules for DNNs. At the same time, this module is capable of stabilizing the distribution of activation in each layer, and therefore facilitates optimization process. Our method can also cooperate well with other practical techniques in deep learning community, e.g., batch normalization (Ioffe and Szegedy 2015), Adam optimization (Kingma and Ba 2014) and Dropout (Srivastava et al. 2014), and moreover improve their original performance.

Comprehensive experiments are conducted over Multilayer Perceptrons (MLPs) and CNNs. By simply substituting the orthogonal linear modules for standard ones without revising any experimental protocols, our method improves the performance of various state-of-the-art CNN architectures, including BN-Inception (Ioffe and Szegedy 2015) and residual networks (He et al. 2016a) over CIFAR (Krizhevsky 2009) and ImageNet (Russakovsky et al. 2015) datasets.

In summarization, our main contributions are as follows.

- To the best of our knowledge, this is the first work to formulate the problem of learning orthogonal filters in DNNs as optimization over multiple dependent Stiefel manifolds problem (OMDSM). We further analyze two remarkable properties of orthogonal filters for DNNs: stabilizing the distributions of activation and regularizing the networks.
- We conduct comprehensive experiments to show that several extensively used Riemannian optimization methods for single Stiefel manifold suffer severe instability in solving OMDSM, We thus propose a novel *orthogonal weight normalization* method to solve OMDSM and show that the solution is stable and efficient in convergence.
- We devise an orthogonal linear module to perform as an alternative to standard linear module for practical purpose.
- We apply the proposed method to various architectures including BN-Inception and residual networks, and achieve significant performance improvement over large scale datasets, including ImageNet.

Optimization over Multiple Dependent Stiefel Manifolds

Let $X \subseteq \mathcal{R}^d$ be the feature space, with d the number of features. Suppose the training set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^M$ is comprised of feature vector $x_i \in X$ generated according to some unknown distribution $x_i \sim \mathcal{D}$, with y_i the corresponding labels. A standard feed-forward neural network with L-layers can be viewed as a function $f(\mathbf{x};\theta)$ parameterized by θ, which is expected to fit the given training data and generalize well for unseen data points. Here $f(\mathbf{x};\theta)$ is a composition of multiple simple nonlinear functions. Each of them usually consists of a linear transformation $\mathbf{s}^l = \mathbf{W}^l\mathbf{h}^{l-1} + \mathbf{b}^l$ with learnable weights $\mathbf{W}^l \in \mathbb{R}^{n_l \times d_l}$ and biases $\mathbf{b}^l \in \mathbb{R}^{n_l}$, followed by an element-wise nonlinearity: $\mathbf{h}^l = \varphi(\mathbf{s}^l)$. Here $l \in \{1, 2, ..., L\}$ indexes the layers. Under this notation, the learnable parameters are $\theta = \{\mathbf{W}^l, \mathbf{b}^l | l = 1, 2, \ldots, L\}$. Training neural networks is to minimize the discrepancy between the desired output $\mathbf{y}$ and the predicted output $f(\mathbf{x};\theta)$. This discrepancy is usually described by a loss function $\mathcal{L}(\mathbf{y}, f(\mathbf{x};\theta))$, and thus the objective is to optimize θ by minimizing the loss function: $\theta^* = \arg\min_\theta \mathbb{E}_{(\mathbf{x},\mathbf{y})\in D}[\mathcal{L}(\mathbf{y}, f(\mathbf{x};\theta))]$.

Formulation

This paper targets to train deep neural networks (DNNs) with orthogonal rectangular weight matrix $\mathbf{W}^l \in \mathbb{R}^{n_l \times d_l}$ in each layer. Particularly, we expect to learn orthogonal filters of each layer (the rows of $\mathbf{W}$). We thus formulate it as a constrained optimization problem:

$$\begin{aligned} \theta^* &= \arg\min_\theta \mathbb{E}_{(\mathbf{x},\mathbf{y})\in D}\left[\mathcal{L}\left(\mathbf{y}, f\left(\mathbf{x};\theta\right)\right)\right] \\ &s.t. \quad \mathbf{W}^l \in \mathcal{O}_l^{n_l \times d_l}, l = 1, 2, ..., L \end{aligned} \tag{1}$$

where the matrix family $\mathcal{O}_l^{n_l \times d_l} = \{\mathbf{W}^l \in \mathbb{R}^{n_l \times d_l} : \mathbf{W}^l(\mathbf{W}^l)^T = \mathbf{I}\}$ is real Stiefel manifold (Absil, Mahony, and Sepulchre 2008; Cunningham and Ghahramani 2015), which is an embedded sub-manifold of $\mathbb{R}^{n_l \times d_l}$. Note that here we assume $n_l \leq d_l$ and will discuss how to handle the case $n_l > d_l$ in subsequent sections. The formulated problem has following characteristics: (1) the optimization space is over multiple embedded submanifolds; (2) the embedded submanifolds $\{\mathcal{O}_1^{n_1 \times d_1}, ..., \mathcal{O}_L^{n_L \times d_L}\}$ is dependent due to the fact that the optimization of weight matrix $\mathbf{W}^l$ is affected by those in preceding layers $\{\mathbf{W}^i, i < l\}$; (3) moreover, the dependencies amplify as the network becomes deeper. We thus call such a problem as Optimization over Multiple Dependent Stiefel Manifolds (OMDSM). To our best knowledge, we are the first to learn orthogonal filters for deep feed-forward neural networks and formulate such a problem as OMDSM. Indeed, the previous works (Wisdom et al. 2016; Vorontsov et al. 2017) that learning orthogonal hidden-to-hidden transformation in RNNs is over single manifold due to weight sharing of hidden-to-hidden transformation.

Properties of Orthogonal Weight Matrix

Before solving OMDSM, we first introduce two remarkable properties of orthogonal weight matrix for DNNs.

Stabilize the Distribution of Activations Orthogonal weight matrix can stabilize the distributions of activations in DNNs as illustrated in the following theorem.

Theorem 1. *Let* $\mathbf{s} = \mathbf{W}\mathbf{x}$*, where* $\mathbf{W}\mathbf{W}^T = \mathbf{I}$ *and* $\mathbf{W} \in \mathbb{R}^{n\times d}$*. (1) Assume the mean of* $\mathbf{x}$ *is* $\mathbb{E}_\mathbf{x}[\mathbf{x}] = \mathbf{0}$*, and covariance matrix of* $\mathbf{x}$ *is* $cov(\mathbf{x}) = \sigma^2\mathbf{I}$*. Then* $\mathbb{E}_\mathbf{s}[\mathbf{s}] = \mathbf{0}$*,* $cov(\mathbf{s}) = \sigma^2\mathbf{I}$*. (2) If* $n = d$*, we have* $\|\mathbf{s}\| = \|\mathbf{x}\|$*. (3) Given the back-propagated gradient* $\frac{\partial\mathcal{L}}{\partial\mathbf{s}}$*, we have* $\|\frac{\partial\mathcal{L}}{\partial\mathbf{x}}\| = \|\frac{\partial\mathcal{L}}{\partial\mathbf{s}}\|$*.*

The proof of Theorem 1 is omitted to supplementary materials in the full paper (https://arxiv.org/abs/1709.06079). The first point of Theorem 1 shows that in each layer of DNNs

the weight matrix with orthonormality can maintain the activation $\mathbf{s}$ to be normalized and even de-correlated if the input is whitened. The normalized and de-correlated activation is well known for improving the conditioning of the *Fisher information matrix* and accelerating the training of deep neural networks (LeCun et al. 1998; Desjardins et al. 2015; Yu et al. 2017). Besides, orthogonal filters can well keep the norm of the activation and back-propagated gradient information in DNNs as shown by the second and third point of Theorem 1.

Regularize Neural Networks Orthogonal weight matrix can also ensure each filter to be *orthonomal*: i.e. $\mathbf{w}_i^T \mathbf{w}_j = 0, i \neq j$ and $\|\mathbf{w}_i\|_2 = 1$, where $\mathbf{w}_i \in \mathbb{R}^d$ indicates the weight vector of the i-th neuron and $\|\mathbf{w}_i\|_2$ denotes the Euclidean norm of $\mathbf{w}_i$. This provides $n(n+1)/2$ constraints. Therefore, orthogonal weight matrix regularizes the neural networks as the embedded Stiefel manifold $\mathcal{O}^{n\times d}$ with degree of freedom $nd - n(n+1)/2$ (Absil, Mahony, and Sepulchre 2008). Note that this regularization may harm the representation capacity if neural networks is not enough deep. We can relax the constraint of orthonormal to orthogonal, which means we don't need $\|\mathbf{w}_i\|_2 = 1$. A practical method is to introduce a learnable scalar parameter g to fine tune the norm of $\mathbf{w}$ (Salimans and Kingma 2016). This trick can recover the representation capacity of orthogonal weight layer to some extent, that is practical in shallow neural networks but for deep CNNs, it is unnecessary based on our observation. We also discuss how to trade off the regularization and optimization efficiency of orthogonal weight matrix in subsequent sections.

Orthogonal Weight Normalization

To solve OMDSM problem, one straightforward idea is to use Riemannian optimization methods that are used for the hidden-to-hidden transform in RNNs (Wisdom et al. 2016; Vorontsov et al. 2017). However, we find that the Riemannian optimization methods to solve OMDSM suffered instability in convergence or inferior performance as shown in the experiment section.

Here we propose a novel algorithm to solve OMDSM problem via re-parameterization (Salimans and Kingma 2016). For each layer l, we represent the weight matrix $\mathbf{W}^l$ in terms of the proxy parameter matrix $\mathbf{V}^l \in \mathbb{R}^{n_l \times d_l}$ as $\mathbf{W}^l = \phi(\mathbf{V}^l)$, and parameter update is performed with respect to $\mathbf{V}^l$. By devising a transformation $\phi : \mathbb{R}^{n_l \times d_l} \to \mathbb{R}^{n_l \times d_l}$ such that $\phi(\mathbf{V}^l) * \phi(\mathbf{V}^l)^T = \mathbf{I}$, we can ensure the weight matrix $\mathbf{W}^l$ is orthogonal. Besides, we require the gradient information back-propagates through the transformation ϕ. An illustrative example is shown in Figure 1. Without loss of generality, we drop the layer indexes of $\mathbf{W}^l$ and $\mathbf{V}^l$ for clarity.

Devising Transformation Inspired by the classic problem of orthogonality-for-vectors (Garthwaite et al. 2012), we represent $\phi(\mathbf{V})$ as linear transformation $\phi(\mathbf{V}) = \mathbf{PV}$. In general, vectors in this problem are usually assumed to be zero-centered. We therefore first center $\mathbf{V}$ by: $\mathbf{V}_C = \mathbf{V} - \mathbf{c}\mathbf{1}_d^T$ where $\mathbf{c} = \frac{1}{d}\mathbf{V}\mathbf{1}_d$ and $\mathbf{1}_d$ is d-dimension vector with all ones. The transformation is performed over $\mathbf{V}_C$.

There can be infinite $\mathbf{P}$ satisfying $\mathbf{W} = \mathbf{PV}_C$ and

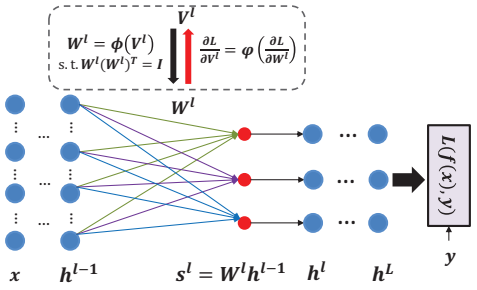


Figure 1: An illustrative example of orthogonal weight normalization in certain layer of neural networks (for brevity, we leave out the bias nodes).

$\mathbf{WW}^T = \mathbf{I}$. For example, if $\hat{\mathbf{P}}$ is the solution, $\mathbf{Q}\hat{\mathbf{P}}$ is also the solution where $\mathbf{Q}$ is an arbitrary orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n\times n}$, since we have $\mathbf{WW}^T = \mathbf{Q}\hat{\mathbf{P}}\mathbf{V}_C\mathbf{V}_C^T\hat{\mathbf{P}}^T\mathbf{Q}^T = \mathbf{QQ}^T = \mathbf{I}$. The question is which $\mathbf{P}$ should be chosen?

In order to achieve a stable solution, we expect the singular values of Jacobians $\partial\mathbf{W}/\partial\mathbf{V}$ close to 1 (Saxe, McClelland, and Ganguli 2013). However, this constraint is difficult to be formulated. We thus look for a relaxation and tractable constraint as minimizing the distortion between $\mathbf{W}$ and $\mathbf{V}_C$ in a least square way:

$$\begin{aligned} &\min_{\mathbf{P}} tr\left((\mathbf{W} - \mathbf{V}_C)(\mathbf{W} - \mathbf{V}_C)^T\right) \\ &s.t. \quad \mathbf{W} = \mathbf{PV}_C \ and \ \mathbf{WW}^T = \mathbf{I}, \end{aligned} \tag{2}$$

where $tr(\cdot)$ indicates the trace of matrix. We omit the derivation of solving this optimization to supplementary materials due to the space limitation. The solution is $\mathbf{P}^* = \mathbf{D}\Lambda^{-1/2}\mathbf{D}^T$, where $\Lambda = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$ and $\mathbf{D}$ represent the eigenvalues and eigenvectors of the covariance matrix $\Sigma = (\mathbf{V} - \mathbf{c}\mathbf{1}_d^T)(\mathbf{V} - \mathbf{c}\mathbf{1}_d^T)^T$. Based on this solution, we use the transformation as follows:

$$\mathbf{W} = \phi(\mathbf{V}) = \mathbf{D}\Lambda^{-1/2}\mathbf{D}^T(\mathbf{V} - \mathbf{c}\mathbf{1}_d^T). \tag{3}$$

We also consider another transformation $\mathbf{P}_{var} = \Lambda^{-1/2}\mathbf{D}^T$ without minimizing such distortions, and observe that $\mathbf{P}_{var}$ suffers the instability problem and fails convergence in subsequent experiments. Therefore, we hypothesize that minimizing distortions formulated by Eqn. 2 is essential to ensure the stability of solving OMDSM.

Back-Propagation We target to update proxy parameters $\mathbf{V}$, and therefore it is necessary to back-propagate the gradient information through the transformation $\phi(\mathbf{V})$. To achieve this, we use the result from matrix differential calculus (Ionescu, Vantzos, and Sminchisescu 2015), which combines the derivatives of eigenvalues and eigenvectors based on chain rule: given $\frac{\partial\mathcal{L}}{\partial\mathbf{D}} \in \mathbb{R}^{n\times n}$ and $\frac{\partial\mathcal{L}}{\partial\Lambda} \in \mathbb{R}^{n\times n}$, where $\mathcal{L}$ is the loss function, the back-propagate derivatives are $\frac{\partial\mathcal{L}}{\partial\Sigma} = \mathbf{D}((\mathbf{K}^T \odot (\mathbf{D}^T\frac{\partial\mathcal{L}}{\partial\mathbf{D}})) + (\frac{\partial\mathcal{L}}{\partial\Lambda})_{diag})\mathbf{D}^T$, where $\mathbf{K} \in \mathbb{R}^{n\times n}$ is 0-diagonal and structured as $\mathbf{K}_{ij} = \frac{1}{\sigma_i - \sigma_j}[i \neq j]$, and $(\frac{\partial\mathcal{L}}{\partial\Lambda})_{diag}$ sets all off-diagonal elements of $\frac{\partial\mathcal{L}}{\partial\Lambda}$ to zero. The $\odot$ operator represents element-wise matrix multiplication. Based on the chain rule, the back-propagated formulations

for calculating $\frac{\partial \mathcal{L}}{\partial \mathbf{V}}$ are shown as below.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \Lambda} &= -\frac{1}{2}\mathbf{D}^T \frac{\partial \mathcal{L}}{\partial \mathbf{W}} \mathbf{W}^T \mathbf{D} \Lambda^{-1} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{D}} &= \mathbf{D}\Lambda^{\frac{1}{2}}\mathbf{D}^T \mathbf{W} \frac{\partial \mathcal{L}}{\partial \mathbf{W}}^T \mathbf{D}\Lambda^{-\frac{1}{2}} + \frac{\partial \mathcal{L}}{\partial \mathbf{W}} \mathbf{W}^T \mathbf{D} \\
\frac{\partial \mathcal{L}}{\partial \Sigma} &= \mathbf{D}((\mathbf{K}^T \odot (\mathbf{D}^T \frac{\partial \mathcal{L}}{\partial \mathbf{D}})) + (\frac{\partial \mathcal{L}}{\partial \Lambda})_{diag})\mathbf{D}^T \\
\frac{\partial \mathcal{L}}{\partial \mathbf{c}} &= -\mathbf{1}_d^T \frac{\partial \mathcal{L}}{\partial \mathbf{W}}^T \mathbf{D}\Lambda^{-\frac{1}{2}}\mathbf{D}^T - 2 \cdot \mathbf{1}_d^T (\mathbf{V} - \mathbf{c}\mathbf{1}_d^T)^T (\frac{\partial \mathcal{L}}{\partial \Sigma})_s \\
\frac{\partial \mathcal{L}}{\partial \mathbf{V}} &= \mathbf{D}\Lambda^{-\frac{1}{2}}\mathbf{D}^T \frac{\partial \mathcal{L}}{\partial \mathbf{W}} + 2(\frac{\partial \mathcal{L}}{\partial \Sigma})_s (\mathbf{V} - \mathbf{c}\mathbf{1}_d^T) + \frac{1}{d}\frac{\partial \mathcal{L}}{\partial \mathbf{c}}^T \mathbf{1}_d^T
\end{aligned}$$

where $(\frac{\partial \mathcal{L}}{\partial \Sigma})_s$ means symmetrizing $\frac{\partial \mathcal{L}}{\partial \Sigma}$ by $(\frac{\partial \mathcal{L}}{\partial \Sigma})_s = \frac{1}{2}(\frac{\partial \mathcal{L}}{\partial \Sigma}^T + \frac{\partial \mathcal{L}}{\partial \Sigma})$. Given $\frac{\partial \mathcal{L}}{\partial \mathbf{V}}$, we can apply regular gradient decent or other tractable optimization methods to update $\mathbf{V}$.

Algorithm 1 Forward pass of OLM.

1: **Input**: mini-batch input $\mathbf{H} \in \mathbb{R}^{d\times m}$ and parameters: $\mathbf{b} \in \mathbb{R}^{n\times 1}$, $\mathbf{V} \in \mathbb{R}^{n\times d}$.
2: **Output**: $\mathbf{S} \in \mathbb{R}^{n\times m}$ and $\mathbf{W} \in \mathbb{R}^{n\times d}$.
3: Calculate: $\Sigma = (\mathbf{V} - \frac{1}{d}\mathbf{V}\mathbf{1}_d\mathbf{1}_d^T)(\mathbf{V} - \frac{1}{d}\mathbf{V}\mathbf{1}_d\mathbf{1}_d^T)^T$.
4: Eigenvalue decomposition: $\Sigma = \mathbf{D}\Lambda\mathbf{D}^T$.
5: Calculate $\mathbf{W}$ based on Eqn. 3.
6: Calculate $\mathbf{S}$ as standard linear module does.

Algorithm 2 Backward pass of OLM.

1: **Input**: activation derivative $\frac{\partial \mathcal{L}}{\partial \mathbf{S}} \in \mathbb{R}^{n\times m}$ and variables from respective forward pass.
2: **Output**: $\{\frac{\partial \mathcal{L}}{\partial \mathbf{H}} \in \mathbb{R}^{d\times m}\}$, $\mathbf{V} \in \mathbb{R}^{n\times d}$ and $\mathbf{b} \in \mathbb{R}^{n\times 1}$.
3: Calculate: $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$, $\frac{\partial \mathcal{L}}{\partial \mathbf{b}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{H}}$ as standard linear module does.
4: Calculate $\frac{\partial \mathcal{L}}{\partial \mathbf{V}}$ base on Eqn. 4
5: Update $\mathbf{V}$ and $\mathbf{b}$.

Orthogonal Linear Module

Based on our orthogonal weight normalization method for solving OMDSM, we build up the Orthogonal Linear Module (OLM) from practical perspective. Algorithm 1 and 2 summarize the forward and backward pass of OLM, respectively. This module can be an alternative of standard linear module. Based on this, we can train DNNs with orthogonality constraints by simply substituting it for standard linear module without any extra efforts. After training, we calculate the weight matrix $\mathbf{W}$ based on Eqn. 3. Then $\mathbf{W}$ will be saved and used for inference as the standard module does.

Convolutional Layer With regards to the convolutional layer parameterized by weights $\mathbf{W}^C \in \mathbb{R}^{n\times d\times F_h\times F_w}$ where F_h and F_w are the height and width of the filter, it takes feature maps $X \in \mathbb{R}^{d\times h\times r}$ as input, where h and r are the height and width of the feature maps, respectively. We denote Δ the set of spatial locations and Ω the set of spatial offsets. For each output feature map k and its spatial location $\delta \in \Delta$, the convolutional layer computes the activation $\{s_{k,\delta}\}$ as: $s_{k,\delta} = \sum_{i=1}^{d}\sum_{\tau\in\Omega} w_{k,i,\tau} h_{i,\delta+\tau} =< \mathbf{w}_k, \mathbf{h}_\delta >$. Here $\mathbf{w}_k$ eventually can be viewed as unrolled filter produced by $\mathbf{W}^C$. We thus reshape $\mathbf{W}^C$ as $\mathbf{W} \in \mathbb{R}^{n\times p}$ where $p = d \cdot F_h \cdot F_w$, and the orthogonalization is executed over the unrolled weight matrix $\mathbf{W} \in \mathbb{R}^{n\times(d\cdot F_h\cdot F_w)}$.

Group Based Orthogonalization In previous sections, we assume $n <= d$, and obtain the solution of OMDSM such that the rows of $\mathbf{W}$ is orthogonal. To handle the case with $n > d$, we propose the *group based orthogonalization* method. That is, we divide the weights $\{w_i\}_{i=1}^n$ into groups with size $N_G <= d$ and the orthogonalization is performed over each group, such that the weights in each group is orthogonal.

One appealing property of *group based orthogonalization* is that we can use group size N_G to control to what extent we regularize the networks. Assume N_G can be divided by n, the free dimension of embedded manifold is $nd - n(N_G + 1)/2$ by using *orthogonal group* method. If we use $N_G = 1$, this method reduces to Weight Normalization (Salimans and Kingma 2016) without learnable scalar parameters.

Besides, *group based orthogonalization* is a practical strategy in real application, especially reducing the computational burden. Actually, the cost of eigen decomposition with high dimension in GPU is expensive. When using group with small size (e.g., 64), the eigen decomposition is not the bottleneck of computation, compared to convolution operation. This make our orthogonal linear module possible to be applied in very deep and high dimensional CNNs.

Computational Complexity We show our method is scalable from complexity analysis here and provide empirical results later for large CNNs. Given a convolutional layer with filters $\mathbf{W} \in \mathbb{R}^{n\times d\times F_h\times F_w}$, and m mini-batch data $\{\mathbf{x}_i \in \mathbb{R}^{d\times h\times w}\}_{i=1}^m$. The computational complexity of our method with group size N_G is $O(nN_GdF_hF_w + nN_G^2 + nmdhwF_hF_w)$ per iteration, and if we control a small group size $N_G \ll mhw$, it will be close to the standard convolutional layer as $O(nmdhwF_hF_w)$.

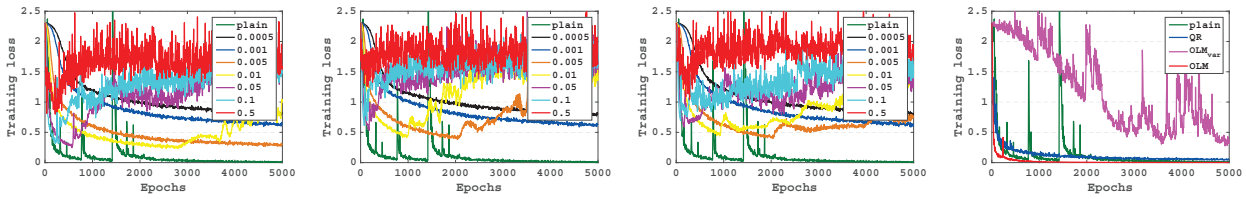
Experiments

In this section, we first conduct comprehensive experiments to explore different methods to solve the OMDSM problem, and show the advantages of our proposed orthogonal weight normalization solution in terms of the stability and efficiency in optimization. We then evaluate the effectiveness of the proposed method that learns orthogonal weight matrix in DNNs, by simply replacing our OLM with standard ones on MLPs and CNNs. Codes to reproduce our results are available from: https://github.com/huangleiBuaa/OthogonalWN.

Comparing Methods for Solving OMDSM

In this section, we use 3 widely used Riemannian optimization methods for solving OMDSM and compared two other baselines. For completeness, we provide a brief review for Riemannian optimization shown in supplementary material and for more details please refer to (Absil, Mahony, and Sepulchre 2008) and references therein.

We design comprehensive experiments on MNIST dataset to compare methods for solving OMDSM. The compared



(a) EI+QR (b) CI+QR (c) CayT (d) Our OLM

Figure 2: Results of Riemannian optimization methods to solve OMDSM on MNIST dataset under the 4-layer MLP. We show the training loss curves for different learning rate of 'EI+QR', 'CI+QR' and 'CayT' compared to the baseline 'plain' in (a), (b) and (c) respectively. We compare our methods to baselines and report the best performance among all learning rates based on the training loss for each method in (d).

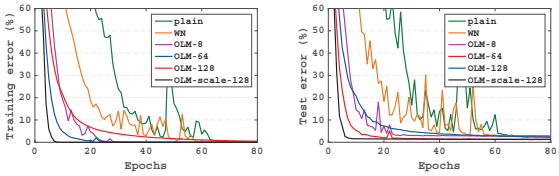
methods including: (1) 'EI+QR': using Riemannian gradient with Euclidean inner product and QR-retraction (Harandi and Fernando 2016); (2) 'CI+QR': using Riemannian gradient with canonical inner product and QR-retraction; (3) 'CayT': using the Cayley transformation (Wisdom et al. 2016; Vorontsov et al. 2017); (4) 'QR': a conventional method that runs the ordinary gradient descent based on gradient $\frac{\partial F}{\partial \mathbf{W}}$ and projects the solution back to the manifold $\mathbb{M}$ by QR decomposition; (5) 'OLM_{var}': using orthogonal transformation: $\mathbf{P}_{var} = \Lambda^{-1/2}\mathbf{D}^T$; (6) 'OLM': our proposed orthogonal transformation by minimizing distortions: $\mathbf{P}^* = \mathbf{D}\Lambda^{-1/2}\mathbf{D}^T$. The baseline is the standard network without any orthogonal constraints referred to as 'plain'.

We use MLP architecture with 4 hidden layers. The number of neurons in each hidden layer is 100. We train the model with stochastic gradient descent and mini-batch size of 1024. We tried a broadly learning rate in ranges of $\{0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5\}$.

We firstly explored the performance of Riemannian optimization methods for solving OMDSM problem. Figure 2 (a), (b) and (c) show the training loss curves for different learning rate of 'EI+QR', 'CI+QR' and 'CayT' respectively, compared to the baseline 'plain'. From Figure 2, we can find that under larger learning rate (e.g., larger than 0.05) these Riemannian optimization methods suffer severe instability and divergence, even though they show good performance in the initial iterations. They can also obtain stable optimization behaviours under small learning rate but are significantly slower in convergence than the baseline 'plain' and suffer worse performance.

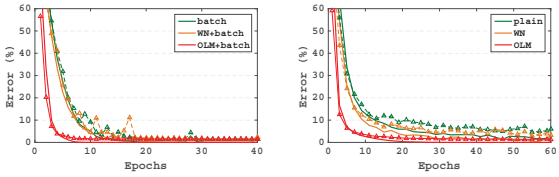
We then compared our proposed method with the baseline 'plain' and the conventional method 'QR', and report the best performance among all learning rates based on the training loss for each method in Figure 2 (d). We can find that the conventional method 'QR' performs stably. However, it also suffers inferior performance of final training loss compared to 'plain'. The proposed '*OLM*' works stably and converges the fastest. Besides, we find that 'OLM_{var}' suffered instability, which means that minimizing distortions formulated by Eqn. 2 is essential to ensure the stability of solving OMDSM.

We also explore 6-layer and 8-layer MLPs and further with mini-batch size of 512 and 256. We observe the similar phenomena shown in supplementary materials due to space limit. Especially with the number of layer increasing, '*OLM*' shows



(a) training error (b) test error

Figure 3: Performance comparisons in MLP architecture on PIE dataset. We compare the effect of different group size of OLM.



(a) batch normalization (b) Adam optimization

Figure 4: Performance comparisons in MLP architecture on PIE dataset by combining (a) batch normalization; (b) Adam optimization. We evaluate the training error (solid lines) and test error (dash lines marked with triangle).

more advantages compared to other methods. These comprehensive experiments strongly support our empirical conclusions that: (1) Riemannian optimization methods probably do not work for the OMDSM problem, and if work, they must be under fine designed algorithms or tuned hyper-parameters; (2) deep feed-forward neural networks (e.g., MLP in this experiment) equipped with orthogonal weight matrix is easier for optimization by our 'OLM' solution.

MLP Architecture

Now we investigate the performance of OLM in MLP architecture. On PIE face recognition dataset with 11,554 images from 68 classes, we sample 1,340 images as the test set and others as training set. Here, we employ standard networks (referred as *plain*) and networks with Weight Normalization (*WN*) (Salimans and Kingma 2016) as baselines for comparisons. *WN* is one of the most related study that normalizes the

weights as unit norm via re-parameterization as OLM does, but it does not introduce the orthogonality for the weights matrix. For all methods, we train a 6-layers MLP with the number of neurons in each hidden layer as 128,128,128,128,128, and Relu as nonlinearity. The mini-batch size is set to 256. We evaluate the training error and test error as a function with respect to epochs.

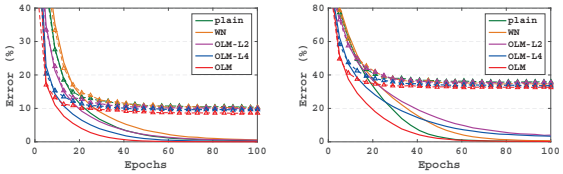
Using Different Group Sizes We explore the effects of group size N_G on the performance when applying OLM. In this setup, we employ stochastic gradient descent (SGD) optimization and the learning rates are selected based on the validation set (10% samples of the training set) from $\{0.05, 0.1, 0.2, 0.5, 1\}$. Figure 3 shows the performance of OLM using different N_G ('OLM-N_G'), compared with *plain* and *WN* methods. We can find that OLM achieves significantly better performance in all cases, which means introducing orthogonality to weight matrix can largely improve the network performance. Another observation is that though increasing group size would help improve orthogonalization, too large group size will reduce the performance. This is mainly because a large $N_G = 128$ provides overmuch regularization. Fortunately, when we add extra learnable scale (indicated by 'OLM-scale-128') to recover the model capacity as described in previous section, it can help to achieve the best performance.

Combining with Batch Normalization Batch normalization (Ioffe and Szegedy 2015) has been shown to be helpful for training the deep architectures (Ioffe and Szegedy 2015; He et al. 2016b). Here, we show that OLM enjoys good compatibility to incorporate well with batch normalization, which still outperforms others in this case. Figure 4 (a) shows the results of training/test error with respect to epochs. We can see that *WN* with batch normalization ('WN+batch') has no advantages compared with the standard network with batch normalization ('batch'), while 'OLM+batch' consistently achieves the best performance.

Applying Adam Optimization We also try different optimization technique such as Adam (Kingma and Ba 2014) optimization. The hyper-parameters are selected from learning rates in $\{0.001, 0.002, 0.005, 0.01\}$. We show error rates based on Adam optimization in Figure 4 (b). From the figure, we can see OLM also obtains the best performance.

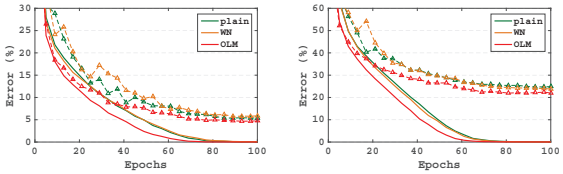
CNN Architectures

In this section, We evaluate our method on a VGG-style CNN (Simonyan and Zisserman 2015), BN-Inception (Szegedy et al. 2015; Ioffe and Szegedy 2015), and Wide Residual Networks (Zagoruyko and Komodakis 2016) for image classification, respectively on CIFAR-10 and CIFAR-100 (Krizhevsky 2009). For each dataset, We use the official training set of 50k images and the standard test set of 10k images. The data preprocessing and data argumentation follow the commonly used mean&std normalization and flip translation as described in (He et al. 2016a). For OLM method, we replace all convolution layers with our OLM modules by default on CNNs, if we do not specify it. Among all experiments, the group size N_G of OLM is set as 64.



(a) CIFAR-10 (b) CIFAR-100

Figure 5: Experimental results on VGG-style architectures over CIFAR datasets. We evaluate the training error (solid lines) and test error (dash lines marked with triangle) with respect to epochs, and all results are averaged over 5 runs.



(a) CIFAR-10 (b) CIFAR-100

Figure 6: Experimental results on BN-Inception over CIFAR datasets. We evaluate the training error (solid lines) and test error (dash lines marked with triangle) with respect to epochs, and all results are averaged over 5 runs.

VGG-style network We adopt the 3×3 convolutional layer as the following specification: $\rightarrow$ $conv(64)$ $\rightarrow$ $conv(128)$ $\rightarrow$ $maxPooling(2,2,2,2)$ $\rightarrow$ $conv(256)$ $\rightarrow$ $conv(256)$ $\rightarrow$ $maxPooling(2,2,2,2)$ $\rightarrow$ $conv(512)$ $\rightarrow$ $conv(512)$ $\rightarrow$ $AvePooling(8,8,1,1)$ $\rightarrow$ $fc(512 \times ClassNum)$. SGD is used as our optimization method with mini-batch size of 256. The best initial learning rate is chosen from $\{0.01, 0.05, 0.1\}$ over the validation set of 5k examples from the training set, and exponentially decayed to 1% in the last (100th) epoch. We set the momentum to 0.9 and weight decay to 5×10^{-4}. Table 1 reports the test error, from which we can find OLM achieves the best performance consistently on both datasets. Figure 5 (a) and (b) show the training and test errors with respect to epochs on CIFAR-10 and CIFAR-100, respectively. On CIFAR-100, to achieve the final test error of *plain* as 36.02 $\%$, OLM takes only 17 epochs. Similarly, on CIFAR-10, OLM only takes 21 epochs to achieve the final test error of *plain* as 10.39 $\%$. While on both datasets, 'plain' takes about 100 epochs. Results demonstrate that OLM converges significantly faster in terms of training epochs and achieves better error rate compared to baselines.

We also study the effect of OLM on different layers. We optionally replace the first 2 and 4 convolution layers with OLM modules (referred as OLM-L2 and OLM-L4 respectively). From Figure 5 and Table 1, we can find that with the numbers of used OLM increasing, the VGG-style network achieves better performance both in optimization efficiency and generalization.

Table 1: Test error (%) on VGG-style over CIFAR datasets. We report the 'mean $\pm std$' computed over 5 independent runs.

	CIFAR-10	CIFAR-100
plain	10.39 ± 0.14	36.02 ± 0.40
WN	10.29 ± 0.39	34.66 ± 0.75
OLM-L2	10.06 ± 0.23	35.42 ± 0.32
OLM-L4	9.61 ± 0.23	33.66 ± 0.11
OLM	**8.61** ± 0.18	**32.58** ± 0.10

Table 2: Test error (%) on BN-Inception over CIFAR datasets. We report the 'mean $\pm std$' computed over 5 independent runs.

	CIFAR-10	CIFAR-100
plain	5.38 ± 0.18	24.87 ± 0.15
WN	5.87 ± 0.35	23.85 ± 0.28
OLM	**4.74** $\pm$ **0.16**	**22.02** ± 0.13

Table 3: Test errors (%) of different methods on CIFAR-10 and CIFAR-100. For OLM, we report the 'mean $\pm std$' computed over 5 independent runs. 'WRN-28-10*' indicates the new results given by authors on their Github.

	CIFAR-10	CIFAR-100
pre-Resnet-1001	4.62	22.71
WRN-28-10	4.17	20.04
WRN-28-10*	3.89	18.85
WRN-28-10-OLM (ours)	**3.73** ± 0.12	18.76 ± 0.40
WRN-28-10-OLM-L1 (ours)	3.82 ± 0.19	**18.61** ± 0.14

Table 4: Top-5 test error (%, single model and single-crop) on ImageNet dataset.

	AlexNet	BN-Inception	ResNet	Pre-ResNet
plain	20.91	12.5	9.84	9.79
OLM	**20.43**	**9.83**	**9.68**	**9.45**

BN-Inception For BN-inception network, batch normalization (Ioffe and Szegedy 2015) is inserted after each linear layer based on original Inception architecture (Szegedy et al. 2015). Again, we train the network using SGD, with the momentum 0.9, weight decay 5×10^{-4} and the batch size 64. The initial learning rate is set to 0.1 and decays exponentially every two epochs until the end of 100 epoches with 0.001. Table 2 reports the test error after training and Figure 5 (c) and (d) show the training/test error with respect to epochs on CIFAR-10 and CIFAR-100, respectively. We can find that OLM converges faster in terms of training epochs and achieve better optimum, compared to baselines, which indicate consistent conclusions for VGG-style network above.

Wide Residual Netwok Wide Residual Network (WRN) has been reported to achieve state-of-the-art results on CIFARs (Zagoruyko and Komodakis 2016). We adopt WRN architecture with depth 28 and width 10 and the same experimental setting as in (Zagoruyko and Komodakis 2016). Instead of ZCA whitening, we preprocess the data using per-pixel mean subtract and standard variance divided as described in (He et al. 2016a). We implement two setups of OLM: (1) replace all the convolutional layers by WRN (*WRN-OLM*); (2) only replace the first convolutional layer in WRN (*WRN-OLM-L1*). Table 3 reports the test errors. We can see that OLM can further improve the state-of-the-art results achieved by WRN. For example, on CIFAR-100, our method *WRN-OLM* achieves the best 18.61 test error, compared to 20.04 of WRN reported in (Zagoruyko and Komodakis 2016). Another interesting observation is that *WRN-OLM-L1* obtains the best performance on CIFAR-10 with test error as 3.73%, compare to 4.17% of WRN, which means that we can improve residual networks by only constraining the first convolution layer orthogonal and the extra computation cost is negligible.

Computation Cost We also evaluate computational cost per iteration in our current Torch-based implementation, where the convolution relies on the fastest *cudnn* package. In the small VGG-style architecture with batch size of 256, OLM costs 0.46s, while *plain* and *WN* cost 0.26s and 0.38s, respectively. On large WRN network, OLM costs 3.12s compared to 1.1s of *plain*. Note that, our current implementation of OLM can be further optimized.

Large Scale Classification on ImageNet Challenge

To further validate the effectiveness of OLM on large-scale dataset, we employ ImageNet 2012 consisting of more than 1.2M images from 1,000 classes (Russakovsky et al. 2015). We use the given 1.28M labeled images for training and the validation set with 50k images for testing. We evaluate the classification performance based on top-5 error. We apply the well-known AlexNet (Krizhevsky, Sutskever, and Hinton 2012) with batch normalization inserted after the convolution layers, BN-Inception, ResNet (He et al. 2016a) with 34 layers and its advanced version Pre-ResNet (He et al. 2016b) to perform the classification task. In AlexNet and BN-Inception, we replace all the convolution layers with OLM modules for our method, and in ResNet and Pre-ResNet, we only replace the first convolution layer with OLM module, which is shown effective with negligible computation cost based on previous experiment.

We run our experiments on one GPU. To guarantee a fair comparison between our method with the baseline, we keep all the experiments settings the same as the publicly available Torch implementation from: https://github.com/facebook/fb.resnet.torch. We apply stochastic gradient descent with momentum of 0.9, weight decay of 0.0001, and set the initial learning rate to 0.1. The exception is that we use mini-batch size of 64 and 50 training epochs considering the GPU memory limitations and training time costs. Regarding learning rate annealing, we use exponential decay to 0.001, which has slightly better performance than the method of lowering by a factor of 10 after epoch 20 and epoch 40 for each method. The final test errors are shown in Table 4. We can find that our proposed OLM

method obtains better performance compared to the baselines over AlexNet, BN-Inception, ResNet and Pre-ResNet architectures.

Conclusions and Further Work

We formulate learning orthogonal linear transformation in DNNs as Optimization over Multiple Dependent Stiefel Manifolds (OMDSM) and propose the Orthogonal Weight Normalization method to solve it, which is stable and can be applied to large and deep networks. Base on this solution, we design Orthogonal Linear Module (OLM) which can be applied as an alternative to standard linear module. We show that neural networks equipped with OLM can improve optimization efficiency and generalization ability. In addition, new deep architectures that address domain-specific representation can also benefit from the proposed method by simply replacing standard linear module with OLM.

Various shallow dimensional reduction methods have been unified under the optimization framework with orthogonality constraints (Cunningham and Ghahramani 2015). Our method has potentials to improve the performance of corresponding unsupervised (Qi Wang 2017) and semi-supervised methods (Rasmus et al. 2015) in DNNs. Besides, our method has great potential to be used in improving the robust of the networks to adversarial examples (Cisse et al. 2017).

Acknowledgments

This work was partially supported by NSFC-61370125, NSFC-61402026, NSFC-61502022, SKLSDE-2017ZX-03, the Innovation Foundation of BUAA for PhD Graduates, China Scholarship Council and NVIDIA PhD Fellowship.

References

Absil, P.-A., and Malick, J. 2012. Projection-like retractions on matrix manifolds. *SIAM Journal on Optimization* 22(1):135–158.

Absil, P.-A.; Mahony, R.; and Sepulchre, R. 2008. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press.

Arjovsky, M.; Shah, A.; and Bengio, Y. 2016. Unitary evolution recurrent neural networks. In *ICML*.

Cisse, M.; Bojanowski, P.; Grave, E.; Dauphin, Y.; and Usunier, N. 2017. Parseval networks: Improving robustness to adversarial examples. In *ICML*.

Cunningham, J. P., and Ghahramani, Z. 2015. Linear dimensionality reduction: Survey, insights, and generalizations. *J. Mach. Learn. Res.* 16(1):2859–2900.

Desjardins, G.; Simonyan, K.; Pascanu, R.; and Kavukcuoglu, K. 2015. Natural neural networks. In *NIPS*.

Dorobantu, V.; Stromhaug, P. A.; and Renteria, J. 2016. Dizzyrnn: Reparameterizing recurrent neural networks for norm-preserving backpropagation. *CoRR* abs/1612.04035.

Garthwaite, P. H.; Critchley, F.; Anaya-Izquierdo, K.; and Mubwandarikwa, E. 2012. Orthogonalization of vectors with minimal adjustment. *Biometrika* 99(4):787 – 798.

Harandi, M., and Fernando, B. 2016. Generalized backpropagation, etude de cas: Orthogonality. *CoRR* abs/1611.05927.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016a. Deep residual learning for image recognition. In *CVPR*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016b. Identity mappings in deep residual networks. In *ECCV*.

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

Ionescu, C.; Vantzos, O.; and Sminchisescu, C. 2015. Training deep networks with structured layers by matrix backpropagation. In *ICCV*.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report.

Krogh, A., and Hertz, J. A. 1992. A simple weight decay can improve generalization. In *NIPS*.

LeCun, Y.; Bottou, L.; Orr, G. B.; and Müller, K.-R. 1998. Effiicient backprop. In *Neural Networks: Tricks of the Trade*.

Ozay, M., and Okatani, T. 2016. Optimization on submanifolds of convolution kernels in cnns. *CoRR* abs/1610.07008.

Qi Wang, Zequn Qin, F. N. Y. Y. 2017. Convolutional 2d lda for nonlinear dimensionality reduction. In *IJCAI*.

Rasmus, A.; Valpola, H.; Honkala, M.; Berglund, M.; and Raiko, T. 2015. Semi-supervised learning with ladder networks. In *NIPS*.

Rodríguez, P.; Gonzàlez, J.; Cucurull, G.; Gonfaus, J. M.; and Roca, F. X. 2017. Regularizing cnns with locally constrained decorrelations. In *ICLR*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV* 115(3):211–252.

Salimans, T., and Kingma, D. P. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS*.

Saxe, A. M.; McClelland, J. L.; and Ganguli, S. 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR* abs/1312.6120.

Simonyan, K., and Zisserman, A. 2015. Very deep convolutional networks for large-scale image recognition. In *ICLR*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15(1):1929–1958.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *CVPR*.

Vorontsov, E.; Trabelsi, C.; Kadoury, S.; and Pal, C. 2017. On orthogonality and learning recurrent networks with long term dependencies. In *ICML*.

Wen, Z., and Yin, W. 2013. A feasible method for optimization with orthogonality constraints. *Math. Program.* 142(1-2):397–434.

Wisdom, S.; Powers, T.; Hershey, J.; Le Roux, J.; and Atlas, L. 2016. Full-capacity unitary recurrent neural networks. In *NIPS*.

Yu, A. W.; Huang, L.; Lin, Q.; Salakhutdinov, R.; and Carbonell, J. G. 2017. Block-normalized gradient method: An empirical study for training deep neural network. *CoRR* abs/1707.04822.

Zagoruyko, S., and Komodakis, N. 2016. Wide residual networks. In *BMVC*.

Zhou, J.; Do, M. N.; and Kovacevic, J. 2006. Special paraunitary matrices, cayley transform, and multidimensional orthogonal filter banks. *IEEE Trans. Image Processing* 15(2):511–519.