Self-Orthogonality Module: A Network Architecture Plug-in for Learning Orthogonal Filters

Ziming Zhang*†
Worcester Polytechnic Institute, MA
zzhang15@wpi.edu

Wenchi Ma* Yuanwei Wu Guanghui Wang ‡
University of Kansas, KS
{wenchima, y262w558, ghwang}@ku.edu

Abstract

In this paper, we investigate the empirical impact of orthogonality regularization (OR) in deep learning, either solo or collaboratively. Recent works on OR showed some promising results on the accuracy. In our ablation study, however, we do not observe such significant improvement from existing OR techniques compared with the conventional training based on weight decay, dropout, and batch normalization. To identify the real gain from OR, inspired by the locality sensitive hashing (LSH) in angle estimation, we propose to introduce an implicit self-regularization *into OR to push the mean and variance of filter angles in a network towards* $90°$ *and* $0°$ *simultaneously to achieve (near) orthogonality among the filters, without using any other explicit regularization. Our regularization can be implemented as an architectural plug-in and integrated with an arbitrary network. We reveal that OR helps* stabilize *the training process and leads to* faster convergence *and* better generalization.

1. Introduction

Nowadays deep learning has achieved the state-of-the-art performance in computer vision and natural language processing [9; 17; 42; 41; 36]. Regularization in deep learning plays an important role in helping avoid bad solutions. Researchers have made great effort on this topic from different perspectives, such as data processing [8; 20; 5; 12], network architectures [46; 16; 26; 27; 36], losses [30], regularizers [32; 48; 6; 28; 43], and optimization [7; 18; 33; 19; 49]. Please refer to [23] for a review.

To better understand the effects of regularization in deep learning, our work in this paper is mainly motivated by the following two basic yet important questions:

Q1. *With the help of regularization, what structural properties among the learned filters in hidden layers*[1] *are good deep models supposed to have?*

To answer this question (partially), we try to explore the angular properties among learned filters. We compute the angles of all filter pairs at each hidden layer in different deep models and plot these *angular distributions* in Fig. 1. To generate each distribution, we first uniformly and randomly draw a sample from the angle pool per hidden layer, and average all the samples to generate a model-level angular sample. We then repeat this procedure for 10^6 times, leading to 10^6 samples based on which we compute a (normalized) histogram as the angular distribution by quantization from $0°$ to $180°$, step by $0.1°$. All the 23 deep models [1] are properly pretrained on different data sets with weight decay [13], dropout [18], and batch normalization (BN) [20].

As shown in Fig. 1, all the angular distributions overlap with each other heavily and behave similarly in Gaussian-like shapes with centers near $90°$ with small variances. Intuitively orthogonal filters are expected to best span the parameter space, especially in the high dimensional spaces where the filter dimensions are larger than the number of filters. Empirically, however, with many noisy factors such as data samples and stochastic training it may not be a good idea to strictly preserve the filter orthogonality in deep learning. In fact, the recent work in [24] has demonstrated that on benchmark data sets, classification accuracy using orthogonal filters (learned by PCA) is inferior to that using learned filters by backpropagation (BP). Similarly another recent work in [35] finds that hard constraints on orthogonality can negatively affect the convergence speed and model performance in training of recurrent neural networks (RNNs), but soft orthogonality can improve the training.

In summary, the comparison on the angular distributions of pretrained deep models in Fig. 1 reveal that deep learning itself may have some internal mechanism to learn nearly orthogonal filters due to its high dimensional parameter spaces, even without any external orthogonal regularization (OR).

*Joint first authors for the paper.

†This work was done when the author was a researcher at Mitsubishi Electric Research Laboratories (MERL).

‡The work was supported in part by USDA NIFA (2019- 67021-28996).

[1]For simplicity, in the rest of the paper we refer to a convolutional or FC layer as a hidden layer.

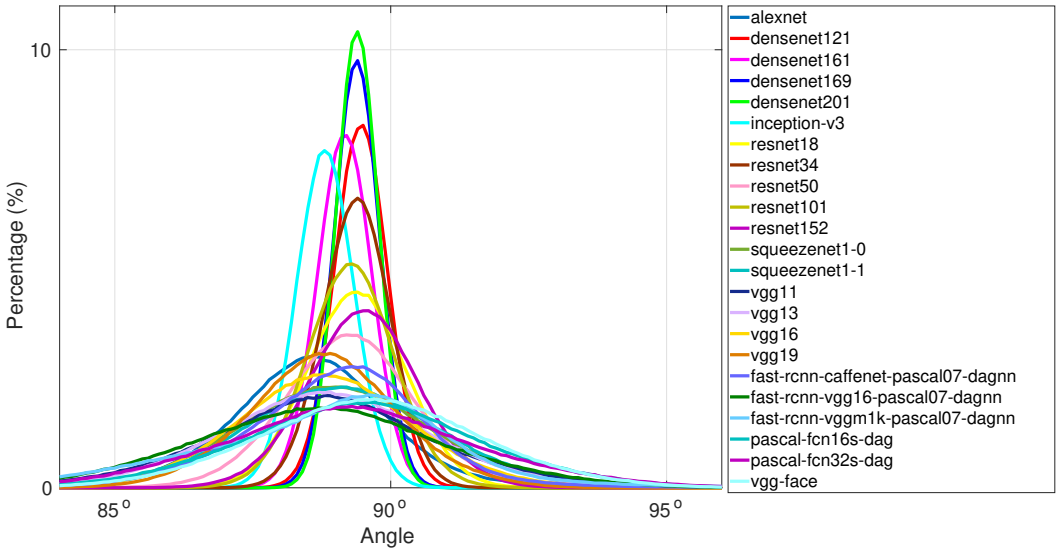

Figure 1: Illustration of the angular distributions of pretrained models with no OR.

Q2. *What are the intrinsic benefits from learning orthogonal filters in deep learning based on OR?*

We notice that recently OR has been attracting more and more attention [32; 47; 35; 19; 6; 24], some of which [32; 35; 19] have released their code. Interestingly, from their code we find that the proposed OR is evaluated together with other regularizers such as weight decay, dropout, and BN. We argue that such experimental settings cannot help identify how much OR contributes to the performance, compared with other regularizers, especially as we observe that the performances with or without OR are very close. Similar argument has been addressed in [34] recently where the author showed that ℓ_2 regularization has no regularizing effect when combined with batch or weight normalization, but has an influence on the scale of weights, and thereby on the effective learning rate.

In summary, it is unclear to us from existing works what is the real gain from OR in deep learning.

Contributions: This paper aims to identify the real gain from OR in training different deep models on different tasks. To do so, we conduct comprehensive experiments on point cloud classification. In contrast to previous works, we separate OR from other regularization techniques to train the same networks respectively. We observe that, however, no significant improvement in accuracy occurs from existing OR techniques, statistically speaking, compared with the conventional training algorithm based on weight decay, dropout, and batch normalization. In fact, we find that, even without any regularization, a workable deep model can achieve the near orthogonality among learned filters, indicating that OR may not be necessarily useful in deep learning to improve accuracy.

What we do observe is that sometimes the training stability using OR is improved, leading to faster convergence in training and better accuracy at test time. We manage to identify this by intentionally designing experiments in extreme learning scenarios such as large learning rate, limited training samples, and small batch sizes. Such observation, however, is not strong overall. We conjecture that this is mainly because existing OR techniques influence the deep learning *externally* and cannot be integrated as a part of network architectures *internally*.

To verify our conjecture, we propose a *self-regularization* technique as a plug-in to the network architectures so that they are able to learn (nearly) orthogonal filters even without any other regularization. We borrow the idea from locality sensitive hashing (LSH) [11] to approximately measure the filter angles at each hidden layer using filter responses from the network. We then push the statistics of such angles (*i.e.* mean and variance) towards 90° and 0°, respectively, as an orthogonality regularizer. We demonstrate that our internal self-regularization significantly improves the training stability, leading to faster convergence and better generalization.

1.1. Related Work

As summarized in [23], there are many regularization techniques in deep learning. For instance, weight decay is essentially an ℓ_2 regularizer over filters, dropout takes random neurons for update, and BN utilizes the statistics from mini-batches to normalize the features. Our work is more

related to representation decorrelation and orthogonality regularizers in the literature.

Representation Decorrelation: Cogswell *et al.* [12] proposed a regularizer, namely DeCov, to learn non-redundant representations by minimizing the cross-covariance of hidden activations. Similarly, Gu *et al.* [14] proposed another regularizer, namely Ensemble-based Decorrelation Method (EDM), by minimizing the covariance between all base learners (*i.e.* hidden activations) during training. Yadav and Agarwal [44] proposed to regularize the training of RNNs by minimizing non-diagonal elements of the correlation matrix computed over the hidden representation, leading to DeCov RNN loss and DeCov Ensemble loss. Zhu *et al.* [50] proposed another decorrelation regularizer based on Pearson correlation coefficient matrix working together with group LASSO to learn sparse neural networks. However, none of the previous work can guarantee that the learned filters are (nearly) orthogonal. Different from these approaches working on hidden activation (representations) by encouraging diverse or non-redundant representations, or like dropout which directly works on neurons by random screening, the proposed method essentially works on regularizing the filter parameters, specifically the weights, to update filters towards orthogonality. It is data adaptive by extracting weights' activation for the regularizer so that the update during training is data dependent.

Orthogonality Regularizers: Harandi and Fernando [15] proposed a generalized BP algorithm to update filters on the Riemannian manifolds as well as introducing a Stiefel layer to learn orthogonal filters. Vorontsov *et al.* [35] verified the effect of learning orthogonal filters on RNN training that is conducted on the Stiefel manifolds. Huang *et al.* [19] proposed an orthogonal weight normalization algorithm based on optimization over multiple dependent Stiefel manifolds (OMDSM). Xie *et al.* [39] proposed a family of orthogonality-promoting regularizer by encouraging the Gram matrix of the functions in the reproducing kernel Hilbert spaces (RKHS) to be close to an identity matrix where the closeness is measured by Bregman matrix divergences. Rodríguez *et al.* [32] proposed a regularizer called OrthoReg to enforce feature orthogonality locally based on cosine similarities of filters. Bansal *et al.* [6] proposed another two orthogonality regularizers based on mutual coherence and restricted isometry property over filters, respectively, and evaluated their gain in training deep models. Xie *et al.* [38] demonstrated that orthonormality among filters helps alleviate the vanishing or exploring gradient issue in training extremely deep networks. Jia *et al.* [21] proposed the algorithms of Orthogonal Deep Neural Networks (OrthDNNs) to connect with recent interest of spectrally regularized deep learning methods.

Self-Regularization: Xu *et al.* [40] proposed a self-regularized neural networks (SRNN) by arguing that the sample-wise soft targets of a neural network may have potentials to drag its own neural network out of its local optimum. Martin & Mahoney [28] proposed interpreting deep neural networks (DNN) from the perspective of random matrix theory (RMT) by analyzing the weight matrices in DNN. They claimed that empirical and theoretical results clearly indicate that the DNN training process itself implicitly implements a form of self-regularization, implicitly sculpting a more regularized energy or penalty landscape.

Differently, we propose introducing self-regularization into the design of OR for better understanding the truly impact of OR in deep learning. In contrast to [28] we discover the self-regularization in convolutional neural networks (CNNs) by considering their angular distributions among learned filters in the context of OR. We propose a novel efficient activation function to compute these angles.

2. Self-Regularization as Internal Orthogonality Regularizer

To better present our experimental results, let us first introduce our self-regularization method. Recall that inspired by the angular distributions of pretrained deep models, we aim to learn deep models with mean and variance of their angular distributions close to 90° and 0° as well. Intuitively we could use $\theta = \arccos \frac{\mathbf{w}_n^T \mathbf{w}_n}{\|\mathbf{w}_m\| \|\mathbf{w}_n\|} \in [0, \pi]$ to directly compute the angle between two filters $\mathbf{w}_m$ and $\mathbf{w}_n$, where $(\cdot)^T$ denotes the matrix transpose operator, and $\|\cdot\|$ denotes the ℓ_2 norm of a vector. Empirically we observe similar behavior of this $\arccos$ based regularization to that of SRIP-v1 and SRIP-v2 (see Sec. 3 for more details). We argue that all the existing orthogonality regularizers are designed *data independently*, and thus lack of the ability of data adaptation in regularization.

In contrast to the literature, we propose introducing implicit *self-regularization* into OR to embed the regularizer into the network architectures directly so that it can be updated *data dependently* during training. To this end, we actively seek for a means that can be used to estimate filter angles based on input data. Only in this way we can naturally incorporate self-regularization with network architectures. Such requirement reminds us of the connection between LSH and angle estimation, leading to the following claim:

Lemma 1 (ϑ-Space)**.** *Without loss of generality, let $\theta \in [0, \pi]$ be the angle between two vectors $\mathbf{w}_m, \mathbf{w}_n \in \mathbb{R}^d$, and $\mathcal{X}$ be the unit ball in the d-dimensional space. We then have*

$$\vartheta \stackrel{def}{=} \mathbb{E}_{\mathbf{x} \sim \mathcal{X}} \left[\operatorname{sgn}(\mathbf{x}^T \mathbf{w}_m) \operatorname{sgn}(\mathbf{x}^T \mathbf{w}_n) \right] = 1 - \frac{2\theta}{\pi}, \quad (1)$$

where $\mathbb{E}$ is the expectation operator, sample $\mathbf{x}$ is uniformly sampled from $\mathcal{X}$, $\operatorname{sgn} : \mathbb{R} \to \{\pm 1\}$ is the sign function returning 1 for positives, otherwise -1.

Proof. In fact $\text{sgn}(\mathbf{x}^T\mathbf{w})$ defines a random hyperplane based hash function for LSH [11]. Therefore,

$$P_{\mathbf{x}\sim\mathcal{X}}\left[\text{sgn}(\mathbf{x}^T\mathbf{w}_m) = \text{sgn}(\mathbf{x}^T\mathbf{w}_n)\right] = 1 - \frac{\theta}{\pi} = \frac{1+\vartheta}{2}, \tag{2}$$

which is equivalent to Eq. 1. We then complete our proof. □

2.1. Formulation

Lemma 1 opens a door for us to estimate filter angles (*i.e.* θ) using filter responses (*i.e.* $\mathbf{x}^T\mathbf{w}$). Due to linear transformation, both mean and variance in the θ-space, *i.e.* $90°$ and $0°$, are converted to 0 in the ϑ-space. Now based on the statistical relation between mean and variance, we can define our self-regularizer, $\mathcal{R}_\vartheta$, as follows:

$$\mathcal{R}_\vartheta \overset{def}{=} \sum_i \lambda_1 \mathbb{E}_{\vartheta\sim\Theta_i}(\vartheta)^2 + \lambda_2 \mathbb{E}_{\vartheta\sim\Theta_i}(\vartheta^2), \tag{3}$$

where $\Theta_i, \forall i$ denotes the filter angle pool in the ϑ-space at the i-th hidden layer, and $\lambda_1, \lambda_2 \geq 0$ are two predefined constants. Here we choose the least square loss for its simplicity, and other proper loss functions can be employed as well. In our experiments we choose $\lambda_1 = 100, \lambda_2 = 1$ by cross-validation using grid search. We observe λ_2 has much larger impact than λ_1 on performance, achieving similar accuracy with $\lambda_1 \in [0, 1000]$ and $\lambda_2 \in [1, 10]$. This makes sense as the mean of an angular distribution in deep learning is close to $90°$ anyway, but the variance should be small.

2.2. Implementation

Computing ϑ in Eq. 1 is challenging because of the expectation over all possible samples in $\mathcal{X}$. To approximate ϑ, we introduce the notion of normalized approximate binary activation as follows:

Definition 1 (Normalized Approximate Binary Activation[2] (NABA)). *Letting* $\mathbf{w} \in \mathbb{R}^d$ *be a vector and* $\mathbf{X} \in \mathbb{R}^{d\times D}$ *be a projection matrix, then we define an NABA vector,* $\mathbf{z} \in \mathbb{R}^D$*, for* $\mathbf{w}$ *as*

$$\mathbf{z} = \frac{\tanh\left(\gamma\mathbf{X}^T\mathbf{w}\right)}{\|\tanh(\gamma\mathbf{X}^T\mathbf{w})\|} \Rightarrow \vartheta(\mathbf{w}_m, \mathbf{w}_n) \approx \mathbf{z}_m^T\mathbf{z}_n, \forall m \neq n \tag{4}$$

where tanh *is an entry-wise function, and* $\gamma \geq 0$ *is a scalar so that* $\lim_{\gamma\to+\infty}\tanh(\gamma x) = \text{sgn}(x), \forall x \in \mathbb{R}$ *as used in [10].*

Based on the consideration of accuracy and running speed, in our experiments we set $\gamma = 10, D = 16$ for Eq. 4. Larger γ brings more difficulty in optimization, as demonstrated in [10]. Larger D does approximate the expectation in Eq. 1 better, but has marginal effect on training loss and test accuracy, as well as leading to higher computational burden.

[2]We tested different sign approximation functions such as softsign, and observed that their performances are very close. Therefore, by referring to [10] in this paper we utilize tanh only.

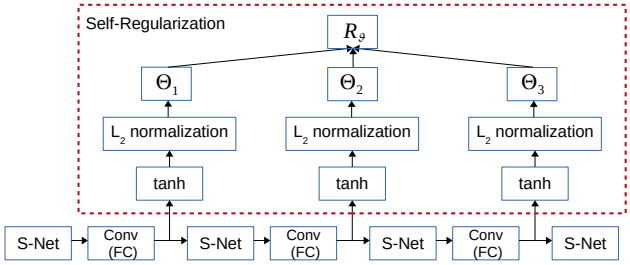

Figure 2: An example of integration of our self-regularization as a plug-in on a three hidden-layer DNN as backbone for training. Here "S-Net" denotes a sub-network consisting of non-hidden layers.

Implementing Eq. 4 using networks is simple, as illustrated in Fig. 2 where $\mathbf{X}$ denotes the input features to each convolutional (conv) or fully-connected (FC) hidden layer and $\mathbf{w}$ denotes the weights of a filter at the hidden layer. We work on the activation of hidden layers and transform feature outputs into a two-dimensional matrix of channel numbers by the neuron dimension, the product of batch size, and feature map's weight and height in the convolution case. Then we take samples along the neuron dimension. Accordingly, z is a vector with the size of sampling number so that the filter angle approximation is workable. Thus, both FC and conv can follow the way of $\mathbf{z}_m^T\mathbf{z}_n$. In this way, our regularization can be easily integrated into an arbitrary network seamlessly as a plug-in so that the network itself can automatically regularize its weights internally and explicitly, leading to self-regularization.

Compared with existing OR algorithms, our self-regularization takes the advantage of the dependency between input features and filters so that the weights are learned more specifically to better fit the data. From this perspective, our self-regularization is similar to a family of batch normalization algorithms. As a consequence, we may not need any external regularization to help training.

3. Experiments

We study the impact of OR on deep learning using the task of point cloud classification.

Learning Scenarios: In order to identify the gain of OR, we intentionally design some extreme learning scenarios where we only change one hyper-parameter from the default setting while keeping the rest unchanged, listed as follows:

- *Default Setting:* Under this setting, we train each model using the full training data set and then test it on the

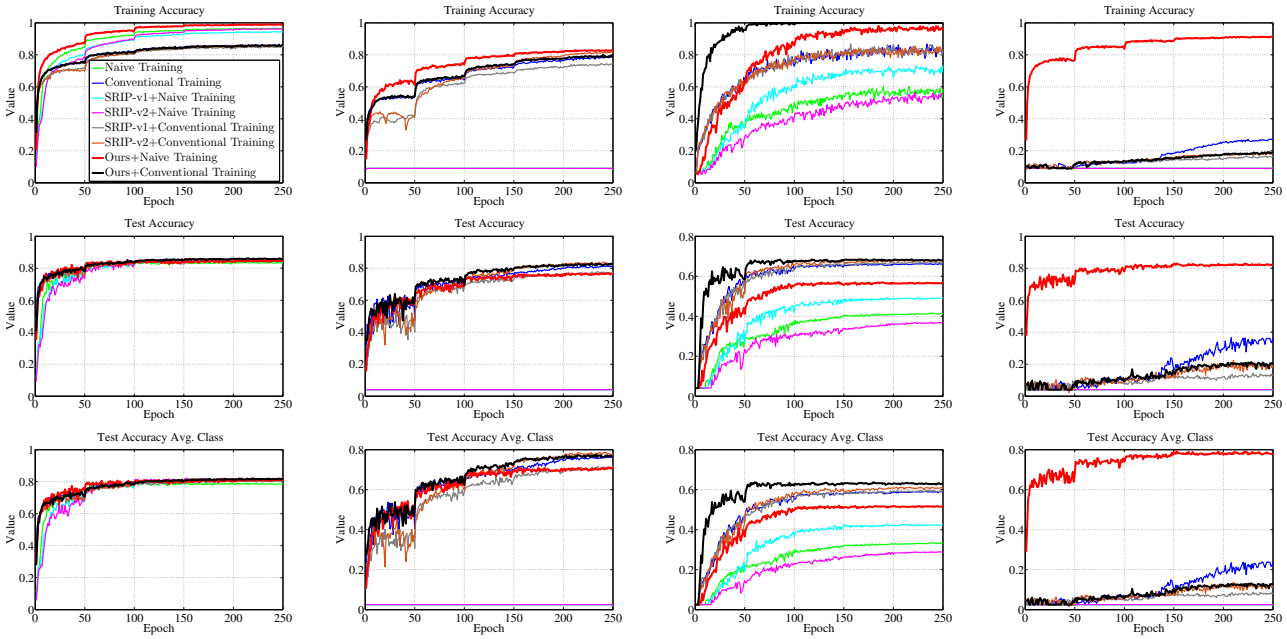

Figure 3: Result comparison on ModelNet40: **(left->right)** default setting, $lr = 0.01$, $ts = 440$, and $bs = 2$.

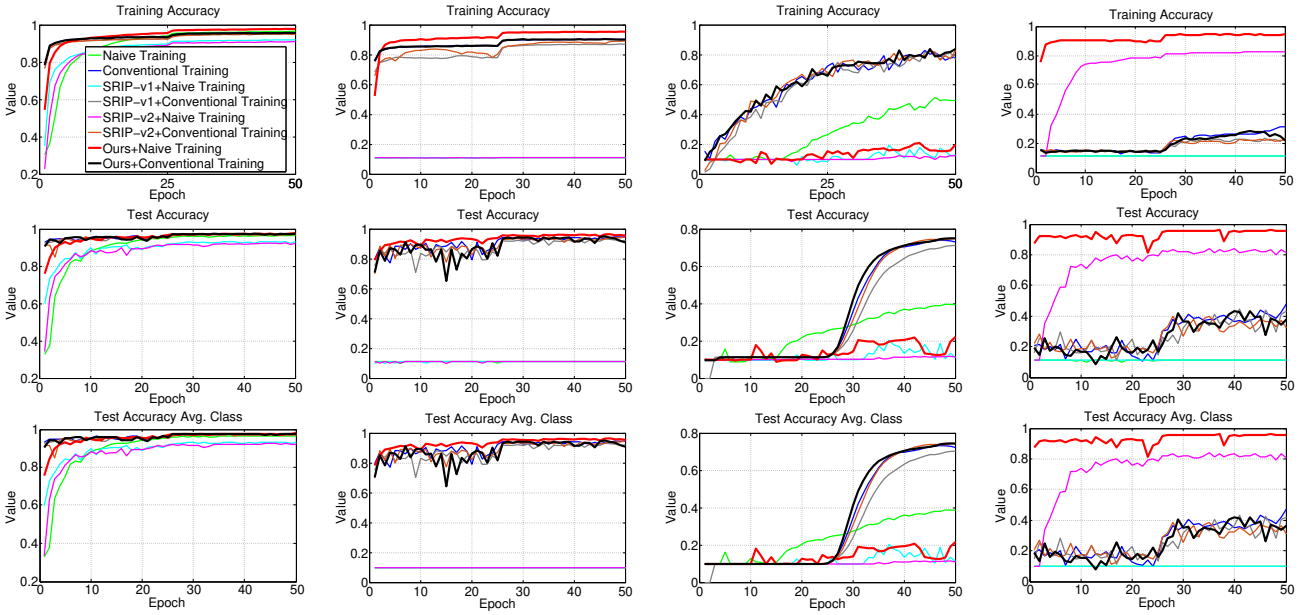

Figure 4: Result comparison on MNIST: **(left->right)** default setting, $lr = 0.01$, $ts = 200$, and $bs = 2$.

full testing data set. We keep all the hyper-parameters unchanged in the public code.

- *Large Initial Learning Rate (lr):* Here we only change the initial learning rate to 0.01.
- *Limited Training Samples (ts):* Here we only use small number of training samples uniformly selected from the entire training set at random.
- *Small Batch Size (bs):* Here we only change the batch size to 2.

Baselines: In order to do comparison fairly, we consider the OR algorithms whose code is publicly available and can run in our task. In this sense, we finally have the following baseline algorithms[3]:

[3]We fail to integrate OrthoReg [32] with our point cloud classification

	default setting		$lr = 0.01$		$ts = 440$		$bs = 2$		ave. of 4 settings	
	avg. cls	overall	avg. cls	overall	avg. cls	overall	avg. cls	overall	avg. cls	overall
NT	0.800	0.840	0.025	0.041	0.336	0.416	0.025	0.041	0.297	0.335
SRIP-v1+NT	0.815	0.849	0.025	0.041	0.428	0.495	0.025	0.041	0.340	0.357
SRIP-v2+NT	**0.822**	**0.858**	0.025	0.041	0.290	0.371	0.025	0.041	0.309	0.328
Ours+NT	0.814	0.850	**0.712**	**0.770**	**0.520**	**0.571**	**0.793**	**0.829**	**0.732**	**0.755**
CT	0.819	0.860	0.767	0.818	0.591	0.666	**0.242**	**0.369**	**0.642**	**0.678**
SRIP-v1+CT	0.812	0.856	0.719	0.780	0.598	0.673	0.088	0.143	0.554	0.613
SRIP-v2+CT	**0.824**	**0.863**	**0.787**	**0.839**	0.612	0.682	0.138	0.223	0.590	0.652
Ours+CT	0.821	0.862	0.775	0.832	**0.638**	**0.685**	0.132	0.214	0.620	0.648

Table 1: Best test accuracy comparison on ModelNet40.

	default setting		$lr = 0.01$		$ts = 200$		$bs = 2$		ave. of 4 settings	
	avg. cls	overall	avg. cls	overall	avg. cls	overall	avg. cls	overall	avg. cls	overall
NT	0.977	0.977	0.967	0.967	**0.398**	**0.389**	0.114	0.100	0.614	0.608
SRIP-v1+NT	0.934	0.934	0.934	0.933	0.204	0.204	0.114	0.100	0.547	0.543
SRIP-v2+NT	0.928	0.927	0.928	0.927	0.119	0.116	0.928	0.927	0.726	0.724
Ours+NT	**0.978**	**0.978**	**0.978**	**0.978**	0.223	0.217	**0.963**	**0.963**	**0.786**	**0.784**
CT	0.967	0.967	0.976	0.976	0.741	0.734	**0.481**	**0.470**	**0.791**	**0.787**
SRIP-v1+CT	0.975	0.975	0.975	0.975	0.713	0.705	0.222	0.206	0.721	0.715
SRIP-v2+CT	0.974	0.973	0.974	0.973	0.751	**0.748**	0.181	0.166	0.720	0.715
Ours+CT	**0.976**	**0.976**	**0.977**	**0.977**	**0.752**	0.746	0.433	0.421	0.785	0.780

Table 2: Best test accuracy comparison on 2D point clouds of MNIST.

- *Naive Training (NT):* In this baseline, we train a network without any regularization.
- *Conventional Training (CT):* In this baseline, we train a network with weight decay (5e-4), dropout (keep-ratio 0.7), and batch normalization (BN).
- *Spectral Restricted Isometry Property [6] (SRIP-v1 & SRIP-v2)*[4]*:* This regularizer is defined as $\mathcal{R} = \lambda \cdot \sigma\left(\mathbf{W}^T\mathbf{W} - \mathbf{I}\right)$, where $\mathbf{W}$ is the network weight matrix, $\mathbf{I}$ is an identity matrix, $\sigma(\cdot)$ is the spectral norm, and λ is a constant. Currently this is the state-of-the-art OR in the literature.

In the sequel we will present our results for different types of networks on different data sets.

3.1. Multilayer Perceptron (MLP): PointNet

Data Sets: We conduct comparison on two benchmark data sets, ModelNet40 [37] and 2D point clouds of MNIST [25]. ModelNet40 has 12,311 CAD models for 40 object categories, split into 9,840 for training, 2,468 for testing. We uniformly sample 1024 points from meshes to obtain 3D point clouds, and normalize them into a unit ball. MNIST is a handwritten digit data set, consisting of 60,000 training images and 10,000 testing images with $28 \times 28 = 784$ gray pixels. We convert each image to 2D point clouds by taking image coordinates of all non-zero pixels.

Networks: We choose PointNet-vanilla [31] (*i.e.* an MLP(64, 64, 64, 128, 1024, 512, 256, 40) without T-nets) as the backbone network and integrate different regularization techniques into it to verify the effects of OR. During training we conduct data augmentation on-the-fly by randomly rotating the points along the up-axis and jittering the coordinates of each point by a Gaussian noise with zero mean and 0.01 std. We use the PyTorch code [3] as our testbed and the same training and evaluation protocols as the original PointNet code. The optimizer is Adam, initial learning rate is 0.001, and momentum is 0.9. The decay rate for batch normalization starts with 0.5, and is gradually increased to 0.99, dropout is set at a ratio of 0.7 until the last fully connected layer. The batch size on ModelNet40 is 32, and on MNIST is 100. We tune each approach to report the best performance.

In order to demonstrate that our findings are common across different MLP, on MNIST we slightly modify the architecture of original PointNet to another MLP(64, 64, 64, 128, 512, 256, 128, 10).

Training Stability: We summarize the training and testing behaviors of each approach on ModelNet40 in Fig. 3. **(a)**. Under the default setting, our regularizer helps the naive training converge much faster than the others which are appreciated, while with the conventional training it seems such effect is neutralized by other regularizations. In testing, the overall behavior of each approach is similar to each other without significant performance gap. **(b)** Under the setting of $lr = 0.01$, naive training and both SRIPs fail to work, while our regularizer with conventional training works reasonably well, it converges faster than others in training and achieves the best performance. **(c)** Under the setting of $ts = 440$,

task, neither using their code nor reimplementing it, because we realize that it is so difficult to modify SGD to achieve reasonable performance or even make it work.

[4]From [2], we find there are two implementations of the approach, under the folders "Wide-Resnet" and "SVHN", respectively. We therefore name them as "v1" and "v2".

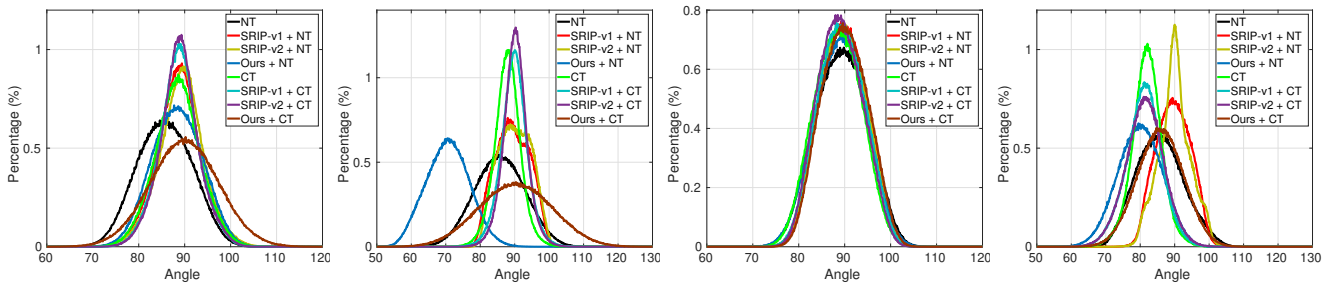

Figure 5: The learned angular distributions on ModelNet40: **(left->right)** default setting, $lr = 0.01$, $ts = 440$, and $bs = 2$.

the performances of different algorithms differ significantly, even though all the competitors work. Our regularizer with conventional training achieves the best performance, converging around 100 epochs which is much faster than the others. Our regularizer outperforms SRIP significantly as well. **(d)** Under the setting of $bs = 2$, only our regularization with naive training works, and surprisingly achieves the very good performance similar to that under the default setting. For conventional training, the regularization dominates the training behavior so strongly that even our regularizer cannot make it work. This is expected as usually conventional regularization techniques cannot work well using very small batch size. In Fig. 4 we summarize the training and testing behaviors of each approach on MNIST. Similar observations can be made here in both training and testing.

In summary, for MLP we do not observe any significant gain on accuracy using SRIP, statistically speaking on average, but some improvement on convergence in training under the default setting. Our self-regularization, however, improves both naive training and conventional training significantly on both convergence and accuracy, indicating better training stability from our method.

Accuracy: We summarize in Table 1 and Table 2 the best accuracy of each approach per setting in Fig. 3 and Fig. 4, respectively. Compared with other regularization techniques, we conclude that our regularizer can work well not only under well-defined setting but also under extreme learning cases. For instance, the naive training with our self-regularization works as well as, or even better than, the conventional training. These observations imply our regularizer has much better generalization ability for optimizing deep networks, potentially leading to broader applications.

Angular Distributions: We illustrate the learned angular distributions in Fig. 5. First of all, it seems that all the well-trained models under the default setting form Gaussian-like distributions with mean around $90°$. The variances of these distributions, however, are larger than those in Fig. 1. We conjecture that the main reason for this is that in PointNet the input dimension is usually smaller than the number of filters per hidden layer so that it is hard to achieve (near) orthogonality among all the filters. Next, our self-regularization

	def. s.	$lr(0.01)$	$ts(400)$	$bs(2)$	ave.
NT	0.898	**0.910**	0.797	0.908	0.878
SRIP-v1+NT	0.898	**0.910**	0.781	0.911	0.875
SRIP-v2+NT	0.897	**0.910**	0.800	**0.912**	0.880
Ours+NT	**0.899**	0.907	**0.810**	0.906	**0.881**
CT	0.894	**0.914**	0.776	0.913	0.874
SRIP-v1+CT	0.894	0.909	0.779	0.915	0.874
SRIP-v2+CT	0.889	0.911	0.776	0.917	0.873
Ours+CT	**0.896**	**0.914**	**0.796**	**0.918**	**0.881**

Table 3: Best test accuracy comparison on ModelNet10.

	lr	acc.
NT	0.001	0.888
SRIP-v1+NT	0.001	0.893
OrthoReg+NT	-	-
Ours+NT	0.001	**0.898**
CT	0.1	0.963
SRIP-v1+CT	0.1	0.962
OrthoReg+CT	0.1	0.962
OMDSM+CT*	0.1	0.963
Ours+CT	0.1	**0.965**

Table 4: Test accuracy comparison on CIFAR-10. Where "-" indicates the model does not converge, "*" indicates OMDSM obtains best performance with dropout=0.3 while keeping all other CT settings unchanged.

with naive training always learns Gaussian-like distributions, while conventional training sometimes has negative effects on our learned distributions such as shifting. This is probably due to BN as it normalizes the statistics in gradients. Finally, it seems that spike-shape distributions tend to produce bad models. This observation has also been made in the angular distributions with random weights.

3.2. Convolutional Neural Networks: VoxNet

Data Set: We use Modelnet10 [29] for our comparison. In the data set there are 3D models as well as voxelized versions, which have been augmented by rotating in 12 rotations. We use the provided voxelizations and follow the train/test splits for evaluation.

Networks: We use VoxNet [29] for comparison, which is a network architecture to efficiently dealing with large amount of point cloud data by integrating a volumetric occu-

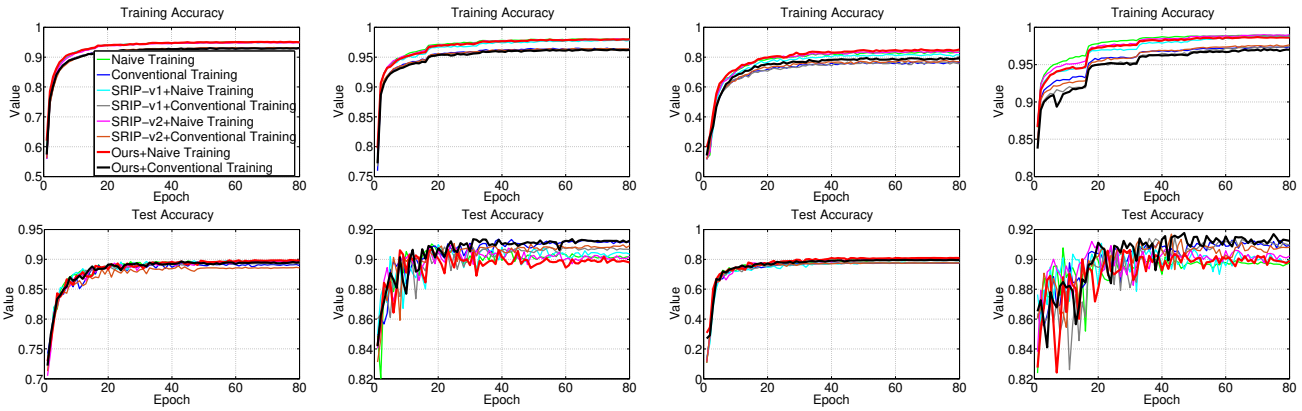

Figure 6: Result comparison on ModelNet10: **(left->right)** default setting, $lr = 0.01$, $ts = 400$, and $bs = 2$.

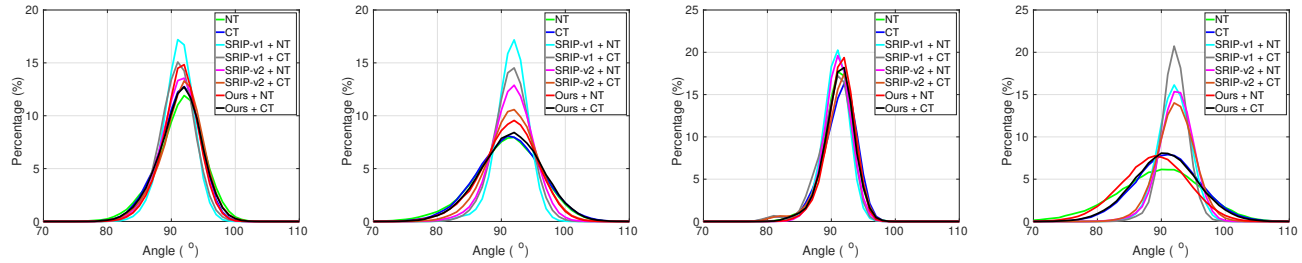

Figure 7: The learned angular distributions on ModelNet10: **(left->right)** default setting, $lr = 0.01$, $ts = 400$, and $bs = 2$.

pancy grid representation into a supervised 3D CNN. We use the PyTorch code [4] as our testbed, and tune each approach to report the best accuracy averaged per class. In the default setting, we use SGD with momentum 0.9, training epoch 80, and batch size 128. The initial learning rate is 0.001 and multiplied by 0.31 every 16 epochs.

Training Stability & Accuracy: We illustrate the training and testing behavior of each algorithm on ModelNet10 in Fig. 6. As we can see, different to the case of PointNet, there is no significant difference in both training and testing for VoxNet. To further verify the performance, we summarize the best test accuracy in Table 3, where our improvement is marginal. Interestingly, we find similar observations in image classification that all the OR algorithms perform equally well, and no obvious advantage over conventional training with careful tuning. For instance, we list our classification results of baselines, OrthoReg [32], and OMSDM [19] on CIFAR-10 [22] based on Wide ResNet [45] with width of 28, depth of 10 in Table 4.

In summary, we observe that OR is more useful for MLP than for CNNs to improve the training stability. We believe one of the key reasons is the filters in CNNs are much better structured due to the input data such as images and 3D volumes, so that learning orthogonal filters becomes unnecessary. In contrast, the input data for MLP is much less structured individually where orthogonal filters can better cover feature space. In both cases, however, our self-regularization outperforms the state-of-the-art OR algorithms.

Angular Distributions: We also illustrate the angular distributions of learned models on MondelNet10 in Fig. 7. Again all the distributions form Gaussian-like shapes with mean close to 90° and relatively small variance. Together with Fig. 5, we conclude that angular distributions may not be a good indicator for selecting good deep models, but their statistics are good for self-regularization.

4. Conclusion

In this paper, we manage to identify that the real gain of orthogonality regularization (OR) in deep learning is to better stabilize the training, leading to faster convergence and better generalization. In terms of accuracy, existing OR algorithms perform no better than the conventional training algorithm with weight decay, dropout, and batch normalization, statistically speaking. Instead, we propose a self-regularization method as an architectural plug-in that can be easily integrated with an arbitrary network to learn orthogonal filters. We utilize LSH to compute the filter angles approximately based on the filter responses, and push the mean and variance of such angles towards 90° and 0°, respectively. Empirical results on point cloud classification demonstrate the superiority of self-regularization.

References

[1] http://www.robots.ox.ac.uk/~albanie/mcn-models.html. 1

[2] https://github.com/nbansal90/Can-we-Gain-More-from-Orthogonality/. 6

[3] https://github.com/meder411/PointNet-PyTorch. 6

[4] https://github.com/Durant35/VoxNet. 8

[5] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 1

[6] N. Bansal, X. Chen, and Z. Wang. Can we gain more from orthogonality regularizations in training deep networks? In *NeurIPS*, pages 4261–4271, 2018. 1, 2, 3, 6

[7] L. Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142. 1

[8] F. Cen and G. Wang. Boosting occluded image classification via subspace decomposition-based estimation of deep features. *IEEE transactions on cybernetics*, 2019. 1

[9] F. Cen and G. Wang. Dictionary representation of deep features for occlusion-robust face recognition. *IEEE Access*, 7:26595–26605, 2019. 1

[10] X. Chang, T. Xiang, and T. M. Hospedales. Deep multi-view learning with stochastic decorrelation loss. *arXiv preprint arXiv:1707.09669*, 2017. 4

[11] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *ACM STOC*, pages 380–388. ACM, 2002. 2, 4

[12] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra. Reducing overfitting in deep networks by decorrelating representations. *arXiv preprint arXiv:1511.06068*, 2015. 1, 3

[13] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*, volume 1. 2016. 1

[14] S. Gu, Y. Hou, L. Zhang, and Y. Zhang. Regularizing deep neural networks with an ensemble-based decorrelation method. In *IJCAI*, pages 2177–2183, 7 2018. 3

[15] M. Harandi and B. Fernando. Generalized backpropagation,Étude de cas: Orthogonality. *arXiv preprint arXiv:1611.05927*, 2016. 3

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 1

[17] L. He, G. Wang, and Z. Hu. Learning depth from single images with deep neural network embedding focal length. *IEEE Transactions on Image Processing*, 27(9):4676–4689, 2018. 1

[18] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 1

[19] L. Huang, X. Liu, B. Lang, A. W. Yu, Y. Wang, and B. Li. Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. In *AAAI*, 2018. 1, 2, 3, 8

[20] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 1

[21] K. Jia, S. Li, Y. Wen, T. Liu, and D. Tao. Orthogonal deep neural networks. *arXiv preprint arXiv:1905.05929*, 2019. 3

[22] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014. 8

[23] J. Kukačka, V. Golkov, and D. Cremers. Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*, 2017. 1, 2

[24] C.-C. J. Kuo, M. Zhang, S. Li, J. Duan, and Y. Chen. Interpretable Convolutional Neural Networks via Feedforward Design. *arXiv preprint arXiv:1810.02786*, 2018. 1, 2

[25] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 6

[26] W. Ma, Y. Wu, F. Cen, and G. Wang. Mdfn: Multi-scale deep feature learning network for object detection. *Pattern Recognition*, page 107149, 2019. 1

[27] W. Ma, Y. Wu, Z. Wang, and G. Wang. Mdcn: Multi-scale, deep inception convolutional neural networks for efficient object detection. In *ICPR*, pages 2510–2515. IEEE, 2018. 1

[28] C. H. Martin and M. W. Mahoney. Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning. *arXiv preprint arXiv:1810.01075*, 2018. 1, 3

[29] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IROS*, pages 922–928. IEEE, 2015. 7

[30] F. Milletari, N. Navab, and S.-A. Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *3DV*, pages 565–571, 2016. 1

[31] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pages 652–660, 2017. 6

[32] P. Rodríguez, J. Gonzalez, G. Cucurull, J. M. Gonfaus, and X. Roca. Regularizing cnns with locally constrained decorrelations. *arXiv preprint arXiv:1611.01967*, 2016. 1, 2, 3, 5, 8

[33] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NeurIPS*, pages 901–909. 2016. 1

[34] T. van Laarhoven. L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350*, 2017. 2

[35] E. Vorontsov, C. Trabelsi, S. Kadoury, and C. Pal. On orthogonality and learning recurrent networks with long term dependencies. In *ICML*, pages 3570–3578. JMLR. org, 2017. 1, 2, 3

[36] Y. Wu, Z. Zhang, and G. Wang. Unsupervised deep feature transfer for low resolution image classification. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019. 1

[37] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, pages 1912–1920, 2015. 6

[38] D. Xie, J. Xiong, and S. Pu. All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation. In *CVPR*, pages 6176–6185, 2017. 3

[39] P. Xie, B. Poczos, and E. P. Xing. Near-orthogonality regularization in kernel methods. In *UAI*, volume 3, page 6, 2017. 3

[40] C. Xu, J. Yang, J. Gao, H. Lai, and S. Yan. Srnn: Self-regularized neural network. *Neurocomputing*, 273:260–270, 2018. 3

[41] W. Xu, S. Keshmiri, and G. R. Wang. Adversarially approx-

imated autoencoder for image generation and manipulation. *IEEE Transactions on Multimedia*, 2019. 1

[42] W. Xu, K. Shawn, and G. Wang. Toward learning a unified many-to-many mapping for diverse image translation. *Pattern Recognition*, 93:570–580, 2019. 1

[43] W. Xu, G. Wang, A. Sullivan, and Z. Zhang. Towards learning affine-invariant representations via data-efficient cnns. *arXiv preprint arXiv:1909.00114*, 2019. 1

[44] M. Yadav and S. Agarwal. Regularization and learning an ensemble of rnns by decorrelating representations. In *The AAAI-17 Workshop on Crowdsourcing, Deep Learning, and Artificial Intelligence Agents WS-17-07*, 2017. 3

[45] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. 8

[46] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013. 1

[47] S. Zhang, H. Jiang, and L. Dai. Hybrid orthogonal projection and estimation (hope): A new framework to learn neural networks. *JMLR*, 17(1):1286–1318, 2016. 2

[48] Z. Zhang and M. Brand. Convergent block coordinate descent for training tikhonov regularized deep neural networks. In *NeurIPS*, pages 1721–1730. 2017. 1

[49] Z. Zhang, Y. Wu, and G. Wang. Bpgrad: Towards global optimality in deep learning via branch and pruning. In *CVPR*, pages 3301–3309, 2018. 1

[50] X. Zhu, W. Zhou, and H. Li. Improving deep neural network sparsity through decorrelation regularization. In *IJCAI*, pages 3264–3270, 7 2018. 3