LEARNING TO SHARE: SIMULTANEOUS PARAMETER TYING AND SPARSIFICATION IN DEEP LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep neural networks (DNNs) usually contain millions, maybe billions, of parameters/weights, making both storage and computation very expensive. This has motivated a large body of work to reduce the complexity of the neural network by using sparsity-inducing regularizers. Another well-known approach for controlling the complexity of DNNs is parameter sharing/tying, where certain sets of weights are forced to share a common value. Some forms of weight sharing are hard-wired to express certain invariances, with a notable example being the shift-invariance of convolutional layers. However, there may be other groups of weights that may be tied together during the learning process, thus further reducing the complexity of the network. In this paper, we adopt a recently proposed sparsity-inducing regularizer, named GrOWL (*group ordered weighted* ℓ_1), which encourages sparsity and, simultaneously, learns which groups of parameters should share a common value. GrOWL has been proven effective in linear regression, being able to identify and cope with strongly correlated covariates. Unlike standard sparsity-inducing regularizers (*e.g.*, ℓ_1 a.k.a. Lasso), GrOWL not only eliminates unimportant neurons by setting all the corresponding weights to zero, but also explicitly identifies strongly correlated neurons by tying the corresponding weights to a common value. This ability of GrOWL motivates the following two-stage procedure: (i) use GrOWL regularization in the training process to simultaneously identify significant neurons and groups of parameter that should be tied together; (ii) retrain the network, enforcing the structure that was unveiled in the previous phase, *i.e.*, keeping only the significant neurons and enforcing the learned tying structure. We evaluate the proposed approach on several benchmark datasets, showing that it can dramatically compress the network with slight or even no loss on generalization performance.

1 INTRODUCTION

Deep neural networks (DNNs) have recently revolutionized machine learning by dramatically advancing the state-of-the-art performance in many applications, ranging from speech and image recognition to playing video games (Goodfellow et al., 2016). A typical DNN consists of a sequence of concatenated layers of functions, potentially involving millions or billions of parameters; by using extremely large training datasets, these architectures are able to learn extremely complex non-linear mappings, features, and dependencies.

A large amount of research has been devoted to the use of regularization in DNN learning (Goodfellow et al., 2016), aiming at reducing the generalization error. It has been shown that the parametrization of many DNNs is very redundant, in the sense that a large fraction of the parameters can be predicted from the remaining ones, with no accuracy loss (Denil et al., 2013). Several regularization methods have been proposed to tackle the potential over-fitting caused by this redundancy. Arguably, the earliest and simplest choice is the classical ℓ_2 norm, known as *weight decay* in the early neural networks literature (Rumelhart et al., 1988), and as *ridge regression* in statistics. In the past two decades, sparsity-inducing regularization based on the ℓ_1 norm, also known as Lasso (Tibshirani, 1996), and variants thereof, became standard tools in statistics and machine learning, including in deep learning (Goodfellow et al., 2016). In recent work, Scardapane et al. (2017) use group-Lasso (a variant of Lasso that assumes that the parameters are organized in groups and encourages sparsity at the group level (Yuan & Lin, 2006)) in deep learning. One of the effects of Lasso or group-Lasso

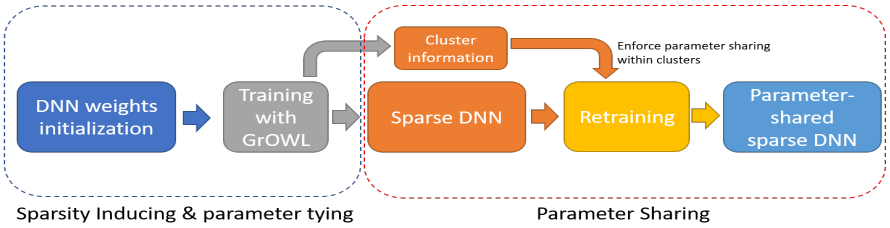

Figure 1: A DNN is first trained with GrOWL regularizer to simultaneously identify the sparse but significant connectivities and the cluster information of the selected connectivities. We then retrain the neuron network only in terms of the selected connectivities while enforcing parameter sharing within each cluster.

regularization in learning a DNN is that many of the parameters may become exactly zero, thus reducing the amount of memory needed to store the model, and lowering the computational cost of applying it.

It has been pointed out by several authors that a major drawback of Lasso (or group-Lasso) regularization is that in the presence of groups of highly correlated covariates/features, it tends to select only one or an arbitrary convex combination of features from each group (Bondell & Reich, 2008; Bühlmann et al., 2013; Figueiredo & Nowak, 2016; Oswal et al., 2016; Zou & Hastie, 2005). Moreover, the learning process tends to be unstable, in the sense that subsets of parameters that end up being selected may change dramatically with minor changes in the data or algorithmic procedure. In DNNs, it is almost unavoidable to encounter correlated features, not only due to the high dimensionality of the input to each layer, but also because neurons tend to co-adapt, yielding strongly correlated features that are passed as input to the subsequent layer (Srivastava et al., 2014).

In this work, we propose using, as a regularizer for learning DNNs, the group version of the *ordered weighted* ℓ_1 (OWL) norm Figueiredo & Nowak (2016), termed group-OWL (GrOWL), which was recently proposed by Oswal et al. (2016). In a linear regression context, GrOWL regularization has been shown to avoid the above mentioned deficiency of group-Lasso regularization. In addition to being a sparsity-inducing regularizer, GrOWL is able to explicitly identify groups of correlated features and set the corresponding parameters/weights to be very close or exactly equal to each other, thus taking advantage of correlated features, rather than being negatively affected by them. In deep learning parlance, this corresponds to adaptive *parameter sharing/tying*, where instead of having to define *a priori* which sets of parameters are forced to share a common value, these sets are learned during the training process. We exploit this ability of GrOWL regularization to encourage parameter sparsity and group-clustering in a two-stage procedure depicted in Fig. 1: we first use GrOWL to identify the significant parameters/weights of the network and, simultaneously, the correlated cluster information of the selected features; then, we retrain the network only in terms of the selected features, while enforcing the weights within the same cluster to share a common value.

The experiments reported in this paper confirm that using GrOWL regularization in learning DNNs does encourage sparsity, by eliminating small weights, and yields parameter sharing, by tying groups of weights to have the same absolute values. We test the proposed approach on two benchmark datasets, MNIST and CIFAR-10, comparing its performance with weight decay and group-Lasso regularization, and exploring the accuracy-memory trade-off. Our results indicate that GrOWL is able to reduce the number of free parameters in the network without degrading the accuracy, as compared to other regularization approaches.

2 Related Work

In order to relieve the burden on both required memory and data for training and storing DNNs, a substantial amount of work has focused on reducing the number of free parameters that need to be estimated, namely by enforcing weight sharing. The classical instance of parameter sharing is found in the convolutional layers of DNNs (Goodfellow et al., 2016). In fact, the use of weight-sharing as a simplifying technique for neural networks can be traced back to more than three decades ago (LeCun, 1987; Rumelhart & McClelland, 1986).

Recently, with the goal of reducing the storage and communication costs of DNNs, there has been a surge of interest in compressing the description of these models. Various methods have been proposed that approximate or quantize the learned weights after the training process. Denton et al. (2014) have shown that, in some cases, it is possible to replace the original weight matrix with a low-rank approximation. Alternatively, Aghasi et al. (2016) propose retraining the network layer by layer, keeping the layer inputs and outputs close to the originally trained model, while seeking a sparse transform matrix, whereas Gong et al. (2014) propose the use of vector quantization to compress the parameters of DNNs.

Network pruning is another relevant line of work. Early work by LeCun et al. (1989) and Hassibi et al. (1993) uses the information provided by Hessian of the loss function to remove less important weights of the network, which requires expensive computation of second order derivatives. Recently, Han et al. (2016) reduce the number of parameters by an order of magnitude using an iterative algorithm that alternates between learning the parameters and removing those below a certain threshold. Li et al. (2016) propose to prune filters, which seeks sparsity with respect to neurons, rather than connectivities; that approach relieves the burden on requiring sparse libraries or special hardware to deploy the network. All these methods either require multiple training/retraining iterations or a careful choice of pruning thresholds.

There is a large body of work on using sparsity-inducing regularization for training DNNs. For example, Collins & Kohli (2014) exploit ℓ_1 and ℓ_0 regularization to encourage weight sparsity; however, the sparsity level achieved is typically modest, making this approach not competitive in terms of DNN compression. Group-Lasso has also been used in training DNNs, which allows seeking sparsity in terms of neurons (Scardapane et al., 2017; Alvarez & Salzmann, 2016; Zhou et al., 2016; Murray & Chiang, 2015) or other structures, *e.g.*, filters, channels, filter shapes, and layer depth (Wen et al., 2016). However, as mentioned above, both Lasso and group-Lasso can fail to select important features if they are strongly correlated with each other. In Section 4, we will further illustrate this fact on both synthetic and real data.

A recent stream of work has focused on using further parameter sharing in convolutional DNNs. By tying weights in an appropriate way, Dieleman et al. (2016) obtain a convolutional DNN with rotation invariance. On the task of analyzing positions in the game *Go*, Clark & Storkey (2015) showed improved performance by constraining features to be invariant to reflections along the x-axis, y-axis, and diagonal-axis. Finally, Chen et al. (2015) used a hash function to randomly group the weights such that those in a hash bucket share the same value. In contrast, with GrOWL regularization, we aim to learn weight sharing from the data itself, rather than specifying it *a priori*.

Dropout-type methods have been proposed to fight over-fitting and are very popular, arguably due to their simplicity of implementation. It has been shown that dropout effectively reduces over-fitting and prevents different neurons from co-adapting. Decorrelation is another popular technique in deep learning pipelines (Bengio & Bergstra, 2009; Cogswell et al., 2015; Rodríguez et al., 2016). Unlike sparsity-inducing regularizers, decorrelation methods try to make full use of the model's capacity by decorrelating the neurons. Although dropout and decorrelation can reduce over-fitting, they do not compress the network, hence do not address the issue of high memory cost. It should also be mentioned that our proposal can be seen as complementary to dropout and decorrelation, in the following sense: whereas dropout and decorrelation can reduce co-adaption of nodes during training, GrOWL regularization copes with co-adaptation by tying together the weights associated to co-adapted nodes.

3 Group-OWL Regularization for Deep Learning

3.1 The Group-OWL Norm

We start by recalling the definition of the group-OWL (GrOWL) regularizer and very briefly reviewing some of its relevant properties (Oswal et al., 2016).

Definition 1. *Given a matrix $W \in \mathbb{R}^{n\times m}$, let $w_{[i]\cdot}$ denote the row of W with the i-th largest ℓ_2 norm. Let $\lambda \in \mathbb{R}^n_+$, with $0 < \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. The GrOWL regularizer (which is a norm)*

$\Omega_\lambda : \mathbb{R}^{n\times m} \to \mathbb{R}$ *is defined as*

$$\Omega_\lambda(W) = \sum_{i=1}^{n} \lambda_i \left\| w_{[i]\cdot} \right\| \tag{1}$$

This is a group version of the OWL regularizer (Figueiredo & Nowak, 2016), also known as WSL1 (*weighted sorted* $\ell 1$ (Zeng & Figueiredo, 2014)) and SLOPE (Bogdan et al., 2015), where the groups are the rows of its matrix argument. It is clear that GrOWL includes group-Lasso as a special case when $\lambda_1 = \lambda_n$. As a regularizer for multiple/multi-task linear regression, each row of W contains the regression coefficients of a given feature, for the m tasks. It has been shown that by adding the GrOWL regularizer to a standard squared error loss function, the resulting estimate of W has the following property: rows associated with highly correlated covariates are very close or even exactly equal to each other (Oswal et al., 2016). In the linear case, GrOWL encourages correlated features to form predictive clusters corresponding to the groups of rows that are nearly or exactly equal. The rationale underlying this paper is that when used as a regularizer for DNN learning, GrOWL will induce both sparsity and parameters tying, as illustrated in Fig. 2 and explained below in greater detail.

3.2 Layer-Wise GrOWL Regularization For Feedforward Neural Networks

A typical feed-forward DNN with L layers can be treated as a function f of the following form:

$$f(x,\theta) \equiv h_L = f_L\left(h_{L-1}W_L + b_L\right), \quad h_{L-1} = f_{L-1}\left(h_{L-2}W_{L-1} + b_{L-1}\right), \quad ..., h_1 = f_1\left(xW_1 + b_1\right)$$

$\theta = \big(W_1, b_1, \ldots, W_L, b_L\big)$ denotes the set of parameters of the network, and each f_i is a component-wise nonlinear activation function, with the *rectified linear unit* (ReLU), the *sigmoid*, and the *hyperbolic tangent* being common choices for this function (Goodfellow et al., 2016).

Given labelled data $\mathcal{D} = \big((x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})\big)$, DNN learning may be formalized as an optimization problem,

$$\min_\theta \mathcal{L}(\theta) + \mathcal{R}(\theta), \quad \text{where } \mathcal{L}(\theta) = \sum_{i=1}^{m} L\Big(y^{(i)}, f\big(x^{(i)}, \theta\big)\Big) , \tag{2}$$

and $L\big(y, \hat{y}\big)$ is the loss incurred when the DNN predicts $\hat{y}$ for y, and $\mathcal{R}$ is a regularizer. The regularizer herein adopted is a sum of GrOWL penalties, each applied to each layer of the neural network, *i.e.,*

$$\mathcal{R}(\theta) = \sum_{l=1}^{L} \Omega_{\lambda^{(l)}}\big(W_l\big), \quad \lambda^{(l)} \in \mathbb{R}_+^{N_{l-1}} , \tag{3}$$

where N_l denotes the number of neurons in the l-th layer and $0 < \lambda_1^{(l)} \geq \lambda_2^{(l)} \geq \cdots \geq \lambda_{N_{l-1}}^{(l)} \geq 0$. Notice that $\mathcal{R}(\theta)$ does not depend on $b_1, ..., b_L$, which means these parameters are not regularized, as is common practice.

As indicated in Eq. (3), the number of groups in each GrOWL regularizer equals the number of neurons in the previous layer, *i.e.,* $\lambda^{(l)} \in \mathbb{R}^{N_{l-1}}$. In other words, *we treat the weights associated with each input feature as a group*. For fully connected layers, where $W_l \in \mathbb{R}^{N_{l-1}\times N_l}$, each group is a row of the weight matrix. When considering convolutional layers, where $W_l \in \mathbb{R}^{F_w \times F_h \times N_{l-1} \times N_l}$, with F_w and F_h denoting the width and height, respectively, of each filter, we first reshape W_l to a 2-dimensional array, *i.e.,* $W_l \to W_l^{2D}$, where $W_l^{2D} \in \mathbb{R}^{N_{l-1}\times(F_w F_h N_l)}$, and then apply GrOWL on the reshaped matrix. That is, if the l-th layer is convolutional, then

$$\mathcal{R}(W_l) = \Omega_{\lambda^{(l)}}\left(W_l^{2D}\right). \tag{4}$$

Each row of W_l^{2D} represents the operation on an input channel. The rationale to apply the GrOWL regularizer to each row of the reshaped weight matrix is that GrOWL can select the relevant features of the network, while encouraging the coefficient rows of each layer associated with strongly correlated features from the previous layer to be nearly or exactly equal, as depicted in Fig. 2. The goal is to significantly reduce the complexity by: **(i)** pruning unimportant neurons of the previous layer that correspond to zero rows of the (reshaped) weight matrix of the current layer; **(ii)** grouping the rows associated with highly correlated features of the previous layer, thus encouraging the

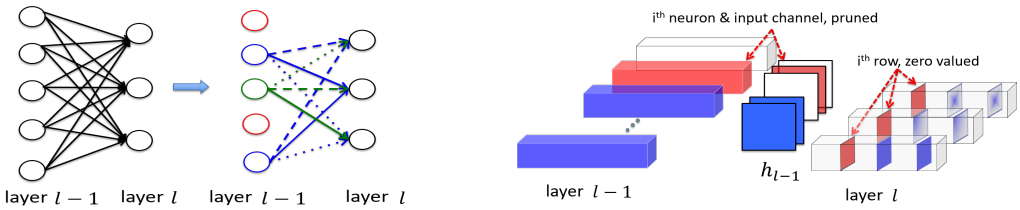

Figure 2: GrOWL's regularization effect on deep nets. **Fully connected layers (Left):** for each layer l GrOWL clusters the input features from the previous layer $l-1$ into different groups, *e.g.,* blue and green. *Within each neuron* of layer l, the weights associated with the input features from the same cluster (input arrows marked with the same color) share the same parameter value. For layer $l-1$, the neurons, corresponding with the zero valued rows of W_h, have zero inputs to the current layer and hence get removed automatically. **Convolutional layers (right):** each group (row) is predefined as the filters that associates with the same input channel, parameter sharing is enforced among the filters within each neuron that corresponds with the same cluster (marked as blue with different effects) of input channels.

coefficient rows in each of these groups to be very close to each other. As a consequence, in the retraining process, we could further compress the neural network by enforcing the parameters *within each neuron* that belong to the same cluster to share same values.

In the work of Alvarez & Salzmann (2016), each group is predefined as the set of parameters associated to a neuron, and group-Lasso regularization is applied to seek group sparsity, which corresponds to zeroing out redundant neurons of each layer. In contrast, we treat the filters corresponding with the same input channel as a group, and GrOWL is applied to prune the redundant groups and thus *remove the associated unimportant neurons of previous layer*, while grouping associated parameters of the current layer that correspond with highly correlated input features to different clusters. Moreover, shown in Section 4, group-Lasso can fail at selecting all relevant features of previous layers, and for the selected ones the corresponding coefficient groups are quite dissimilar from each other, making it impossible to further compress the DNN by enforcing parameter tying.

3.3 Proximal Gradient Algorithm

To solve (2), we use a *proximal gradient algorithm* (Bauschke & Combettes, 2011), which has the following general form: at the t-th iteration, the parameter estimates are updated according to

$$\theta^{(t+1)} = \text{prox}_{\eta\mathcal{R}}\left(\theta^{(t)} - \eta\nabla_\theta\mathcal{L}(\theta^{(t)})\right) \tag{5}$$

where, for some convex function Q, prox_Q denotes its proximity operator (Bauschke & Combettes, 2011), which is defined as $\text{prox}_Q(\xi) = \arg\min_\nu Q(\nu) + \frac{1}{2}\|\nu - \xi\|_2^2$. In the previous expression, $\|\nu - \xi\|_2^2$ denotes the sum of the squares of the differences between the corresponding components of ν and ξ, regardless of their organization (in our case, a collection of matrices and vectors).

Since $\mathcal{R}(\theta)$, as defined in (3), is separable across the weight matrices of different layers and zero for $b_1, ..., b_L$, the corresponding proximity operator is also separable, thus

$$W_l^{(t+1)} = \text{prox}_{\eta\Omega_\lambda^{(l)}}\left(W_l^{(t)} - \eta\,\nabla_{W_l}\mathcal{L}(\theta^{(t)})\right), \quad \text{for } l = 1, \ldots, L \tag{6}$$

$$b_l^{(t+1)} = b_l^{(t)} - \eta\,\nabla_{b_l}\mathcal{L}(\theta^{(t)}) \quad \text{for } \; l = 1, ..., L. \tag{7}$$

It was proved by Oswal et al. (2016) that the proximity operator of the GrOWL norm can be computed as follows. For some matrix $V \in \mathbb{R}^{N\times M}$, let $U = \text{prox}_{\Omega_\lambda}(V)$, and v_i and u_i denote the corresponding i-th rows. Then,

$$u_i = \frac{v_i}{\|v_i\|}\left(\text{prox}_{\Omega_\lambda}(\tilde{v})\right)_i \tag{8}$$

where $\tilde{v} = [\|v_1\|, \|v_2\|, \cdots, \|v_N\|]$. The proximal operator $\text{prox}_{\Omega_{\lambda^{(l)}}}$ for a vector in $\mathbb{R}^N$ (in which case GrOWL coincides with OWL) can be computed with $O(n\log n)$ complexity, where the core

computation is the so-called *pool adjacent violators algorithm* (PAVA (de Leeuw et al., 2009)) for isotonic regression. We provide one of the existing algorithms in Appendix A; for details of the algorithm, the reader is referred to the work of Bogdan et al. (2015) and Zeng & Figueiredo (2015). In this paper, we apply the proximal gradient descent per epoch, which generally yields better performance. The training method is summarized in Algorithm 1.

Algorithm 1

Input: parameters of the OWL regularizers $\lambda^{(l)}, ..., \lambda^{(L)}$, learning rate η
for each epoch T **do**
  for each iteration t in epoch T **do**
    Update the parameter via backpropagation (BP)
  end for
  Apply proximity operator via (6)
end for

3.4 Implementation Details

3.4.1 Setting the GrOWL Weights

GrOWL is a family of regularizers, with different variants obtained by choosing different weight sequences $\lambda_1, \ldots, \lambda_n$. In this paper, we propose the following choice:

$$\lambda_i = \begin{cases} \Lambda_1 + (p - i + 1)\Lambda_2, & \text{for } i = 1, ..., p, \\ \Lambda_1, & \text{for } i = p + 1, ..., n, \end{cases} \tag{9}$$

where $p \in \{1, ...n\}$ is a parameter. The first p weights follow a linear decay, while the remaining ones are all equal to Λ_1. Notice that, if $p = n$, the above setting is equivalent to OSCAR (Bondell & Reich, 2008). Roughly speaking, Λ_1 controls the sparsifying strength of the regularizer, while Λ_2 controls the clustering property (correlation identification ability) of GrOWL (Oswal et al., 2016). Moreover, by setting the weights to a common constant beyond index p means that clustering is only encouraged among the p largest coefficients, *i.e.,* only among relevant coefficient groups.

Finding adequate choices for p, Λ_1, and Λ_2 is crucial for jointly selecting the relevant features of the network and identifying the underlying correlations among them. In practice, we find that with properly chosen p, GrOWL is able to find more correlations of the underlying data than OSCAR. We explore different choices of p in Section 4.1.

3.4.2 Parameter Tying

After the initial training phase, at each layer l, rows of W_l that corresponds to highly correlated outputs of layer $l - 1$ have been made similar or even exactly equal. To further compress the DNN, we force rows that are close to each other to be identical. We first group the rows into different clusters [1] according to the *pairwise similarity* metric

$$\mathcal{S}_l(i, j) = \frac{W_{l,i}^T W_{l,j}}{\max\left(\|W_{l,i}\|_2^2, \|W_{l,j}\|_2^2\right)} \in [-1, 1], \tag{10}$$

where $W_{l,i}$ and $W_{l,j}$ denote the i-th and j-th rows of W_l, respectively.

With the cluster information obtained by using GrOWL, we enforce parameter sharing for the rows that belong to a same cluster by replacing their values with the averages (centroid) of the rows in that cluster. In the following retraining process , let $\mathcal{G}_k^{(l)}$ denote the k-th cluster of the l-th layer, then centroid $g_k^{(l)}$ of this cluster is updated via

$$\frac{\partial \mathcal{L}}{\partial g_k^{(l)}} = \frac{1}{\left|\mathcal{G}_k^{(l)}\right|} \sum_{W_{l,i} \in \mathcal{G}_k^{(l)}} \frac{\partial \mathcal{L}}{\partial W_{l,i}}. \tag{11}$$

[1] In this paper, we use the built-in *affinity propagation* method of the scikit-learn package (Buitinck et al., 2013). A brief description of the algorithm is provided in Appendix B where we also talk about why we choose this algorithm to cluster the rows.

4 NUMERICAL RESULTS

We assess the performance of the proposed method on two benchmark datasets: MNIST and CIFAR-10. We consider two different networks and compare GrOWL with group-Lasso and weight decay, in terms of the compression vs accuracy trade-off. For fair comparison, the training-retraining pipeline is used with the different regularizers. After the initial training phase, the rows that are close to each other are clustered together and forced to share common values in the retraining phase. We implement all models using Tensorflow Abadi et al. (2016). To evaluate the effect of the different regularizers, we use the following quantities: sparsity $= \frac{\#\text{zero params}}{\#\text{ total params}}$; compression rate $= \frac{\#\text{ total params}}{\#\text{ unique params}}$; parameter sharing $= \frac{\#\text{ nonzero params}}{\#\text{ unique params}}$.

4.1 DIFFERENT CHOICES OF GROWL PARAMETERS

We first evaluate GrOWL norm regularization on synthetic data. The covariance matrix Σ of the data X is set as a *block diagonal matrix* such that each block represents the covariances of a cluster of correlated features, and there is a gap g between two blocks. Within each cluster, the covariance between two features X_i and X_j is $\text{cov}(X_i, X_j) = 0.96^{|i-j|}$. we assume that the features from different clusters are independent of each other, while In this section, we set $n = 784, K = 10$ and block size equals 50, $g = 28$. We generate 10000 training examples and 1000 testing examples.

We train a neural network with a single fully connected layer that consists of 300 hidden units. In Fig 3, we plot the the first 25000 entries of the sorted similarity matrices obtained by applying GrOWL with different p (Eq 9) values. Although, by setting the weights to a common constant beyond index p means that clustering is only encouraged among the p largest coefficients, i.e., only among relevant coefficient groups, Figure 3 shows that, with probably chosen p, GrOWL is able to encourage more parameter tying than OSCAR ($p = n$). On the other hand, choosing smaller p values allows us to use large Λ_2, which encourages parameter tying among comparatively loose correlations. In practice, we found that choosing p around the target portion of nonzero parameters over total parameters yields better performance in general. The intuition is that we only want to identify correlations among the remaining important parameters/connections.

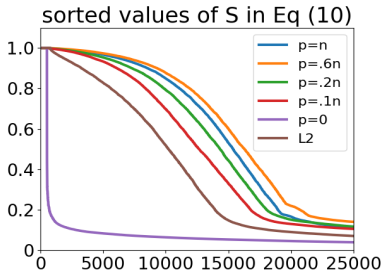

Figure 3: Regularization effect of GrOWL with different p values in Eq (9).

Fig 3 shows that weight decay (denoted as ℓ_2) can also push parameters together, though the parameter tying effect is not as obvious as that obtained by applying GrOWL. As has been observed in the literature (Bondell & Reich, 2008), weight decay often achieves better generalization as compared with sparsity-inducing regularizers. It achieves this via parameter shrinkage, especially in the highly correlated region, but it cannot yield a sparse model. In the following section, we explore the compression performance of GrOWL by comparing it with both group-Lasso and weight decay. For each case, the baseline performance is provided as the best performance obtained by running the original neural network (without compression) after sweeping the hyper-parameter on the weight decay regularizer over a range of values.

4.2 FULLY CONNECTED NEURAL NETWORK ON MNIST

The MNIST dataset contains 28×28 (784) pixels images of handwritten digits (0, 1, ..., 9), each located at the center of the image. Fig 4 (a) shows the (784×784) correlation matrix of the dataset (the margins are zero due to the redundant background of the images). We use a network with a single fully connected layer of 300 hidden units. The network is trained for 300 epochs and then retrained for an additional 100 epochs, both with momentum. The initial learning rate is set to 0.001 for both the training and retraining phases, and it is reduced by a factor of 0.96 every 10 epochs. The hyper-parameters are selected through grid search.

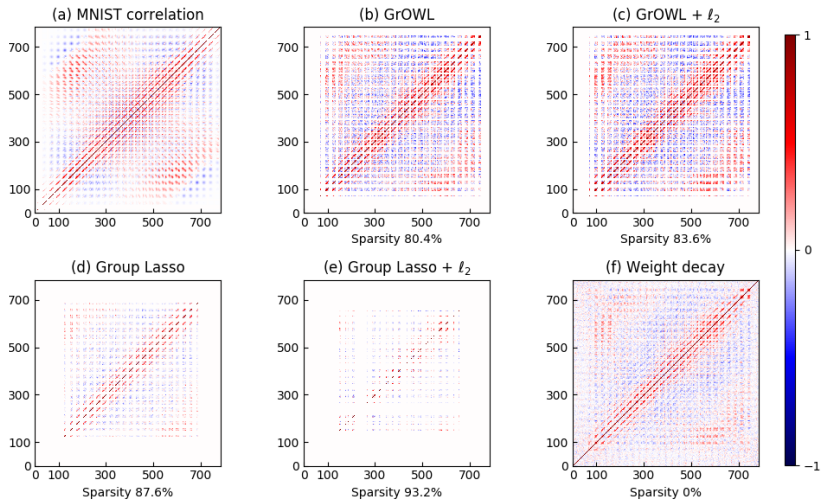

Figure 4: MNIST: comparison of the data correlation and the similarity maps (Eq (10)) of the parameter rows obtained by training the neural network with GrOWL, GrOWL+ℓ_2, group-Lasso, group-Lasso+ℓ_2 and weight decay.

Pairwise similarities (see Eq. (10)) between the rows of the weight matrices learned with different regularizers are shown in Fig. 4 (b–f). As we can see, GrOWL (+ℓ_2) identifies more correlations than group-Lasso (+ℓ_2), and the similarity patterns in Fig. 4 (b, c) are very close to that of the data (Fig. 4(a)). On the other hand, weight decay also identifies correlations between parameter rows, but it does not induce a sparse model. Moreover, as shown in Table 1, GrOWL obtains a higher level of parameter sharing than does weight decay.

The compression vs accuracy trade-off of the different regularizers is summarized in Table 1. As we can see, by applying the ℓ_2 norm regularizer together with group-Lasso or GrOWL, we achieve a higher compression rate while yielding even better accuracy. Table 1 also shows that, with even smaller sparsities after the initial training phase, GrOWL ($+\ell_2$) compresses the network more than group-Lasso ($+\ell_2$) does, due to the significant amount of correlation it identifies. This also implies that group-Lasso can only select a subset of the correlated features while GrOWL selects all of them. On the other hand, group-Lasso suffers from randomly selecting a subset of correlated features. We illustrate this effect in Fig. 5, which plots the indices of nonzero rows, showing that GrOWL stably selects relevant features while group-Lasso does not. The mean ratios of changed indices[2] are 11.09% and 0.58% for group-Lasso and GrOWL, respectively.

Table 1: Sparsity, parameter sharing, and compression rate results on MNIST. Baseline model is trained with weight decay and we do not enforce parameter sharing for baseline model. We train each model for 5 times and report the average values together with their standard deviations.

Regularizers	**Sparsity**	**Parameter Sharing**	**Compression**	**Accuracy**
baseline	$0.0 \pm 0\%$	1.0 ± 0	$1.0 \pm 0X$	$98.3 \pm 0.1\%$
weight decay	$0.0 \pm 0\%$	1.6 ± 0	$1.6 \pm 0X$	$98.4 \pm 0.0\%$
group-Lasso	$87.6 \pm 0.1\%$	1.9 ± 0.1	$15.8 \pm 1.0X$	$98.1 \pm 0.1\%$
group-Lasso+ℓ_2	$93.2 \pm 0.4\%$	1.6 ± 0.1	$23.7 \pm 2.1X$	$98.0 \pm 0.1\%$
GrOWL	$80.4 \pm 1.0\%$	3.2 ± 0.1	$16.7 \pm 1.3X$	$98.1 \pm 0.1\%$
GrOWL+ℓ_2	$83.6 \pm 0.5\%$	3.9 ± 0.1	$\mathbf{24.1 \pm 0.8X}$	$\mathbf{98.1 \pm 0.1\%}$

[2]The mean ratio of changed indices is defined as: $\frac{1}{n}\sum_{k=1}^{n} \|I_k - \bar{I}\|_1 / \|\bar{I}\|_0$, where n is the number of experiments, I_k is the index vector of kth experiment, and $\bar{I} = \frac{1}{n}\sum_{k=1}^{n} I_k$ is the mean index vector.

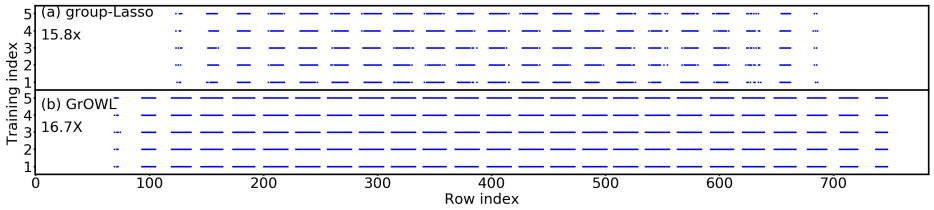

Figure 5: MNIST: sparsity pattern of the trained fully connected layer, for 5 training runs, using group-Lasso (2) and GrOWL. For similar compression rates (15.8X and 16.7X), GrOWL is more stable.

4.3 VGG-16 ON CIFAR-10

To evaluate the proposed method on large DNNs, we consider a VGG-like (Simonyan & Zisserman, 2014) architecture proposed by Zagoruyko (2015) on the CIFAR-10 dataset. The network is summarized in Table 3; comparing with the original VGG of Simonyan & Zisserman (2014), their fully connected layers are replaced with two small ones with 14.3% of the total parameters. A batch normalization layer is added after each convolutional layer and the first fully connected layer. Unlike Zagoruyko (2015), we don't use dropout. We first train under different regularizers over 150 epochs, then retrain for another 50 epochs, using the learning rate decay scheme described by He et al. (2016): the initial rates for the training and retraining phases are set to 0.01 and 0.001, respectively; the learning rate is multiplied by 0.1 every 60 epochs of the training phase, and every 20 epochs of the retraining phase. For GrOWL ($+\ell_2$), we set $p = 0.1\,n$ (see Eq. (9)) for all layers, where n denotes the number of rows of each layer.

Table 2: Sparsity (**S1**) and Parameter Sharing (**S2**) of VGG-16 on CIFAR-10. Layers marked by * are regularized. We report the averaged results over 5 runs.

Layers	Weight Decay (**S1**, **S2**)	group-Lasso (**S1**, **S2**)	group-Lasso + ℓ_2 (**S1**, **S2**)	GrOWL (**S1**, **S2**)	GrOWL + ℓ_2 (**S1**, **S2**)
conv1	0%, 1.0	0%, 1.0	0%, 1.0	0%,1.0	0%, 1.0
*conv2	0%, 1.0	34%, 1.0	40%, 1.0	20%, 1.0	34%, 1.0
*conv3	0%, 1.0	28%, 1.0	20%, 1.0	28%, 1.0	17%, 1.0
*conv4	0%, 1.0	34%, 1.0	29%, 1.0	30%, 1.0	27% 1.0
*conv5	0%, 1.0	12%, 1.0	11%, 1.0	8%, 1.0	14%, 1.0
*conv6	0%, 1.0	38%, 1.0	40%, 1.0	38%, 1.0	43%, 1.0
*conv7	0%, 1.0	46%, 1.0	51%, 1.0	40%, 1.0	50%, 1.0
*conv8	0%, 1.0	49%, 1.0	53%, 1.0	50%, 1.0	55%, 1.0
*conv9	0%, 1.0	78%, 1.0	78%, 1.0	74%, 1.1	75%, 1.2
*conv10	0%, 1.2	76%, 1.0	76%, 1.0	66%, 2.7	73%, 3.0
*conv11	0%, 1.2	84%, 1.0	87%, 1.0	81%, 3.7	88%, 3.7
*conv12	0%, 2.0	85%, 1.0	91%, 1.0	75%, 2.6	78%, 2.5
*conv13	0%, 2.1	75%, 1.1	90%, 1.1	78%, 1.9	71%, 4.2
*fc	0%, 4.2	78%, 1.0	91%, 1.1	69%, 2.7	81%, 2.2
softmax	0%, 1.0	0%,1.0	0%, 1.0	0%, 1.0	0%, 1.0
Compression	1.3 ± 0.1X	11.1 ± 0.5X	$\mathbf{14.5 \pm 0.5}$X	11.4 ± 0.5X	$\mathbf{14.5 \pm 0.5}$X
Accuracy	$\mathbf{93.1 \pm 0.0}\%$	$92.1 \pm 0.2\%$	$\mathbf{92.7 \pm 0.1}\%$	$92.2 \pm 0.1\%$	$\mathbf{92.7 \pm 0.1}\%$
Baseline	Accuracy: $93.4 \pm 0.2\%$, Compression: 1.0X				

The results are summarized in Table 2.[3] In the experiments, we found that it is quite hard to encourage parameter tying in the first 7 convolutional layers. The reason may be that the filters of these first 7 convolutional layers have comparatively large feature maps (from 32×32 to 8×8), which are

[3]For all of the regularizers, we use the preference value 0.8 to run the *affinity propagation* algorithm to cluster the rows at the end of initial training process. In Table 4 (Appendix D), we explore the clustering effects of the *affinity propagation* over different preference values.

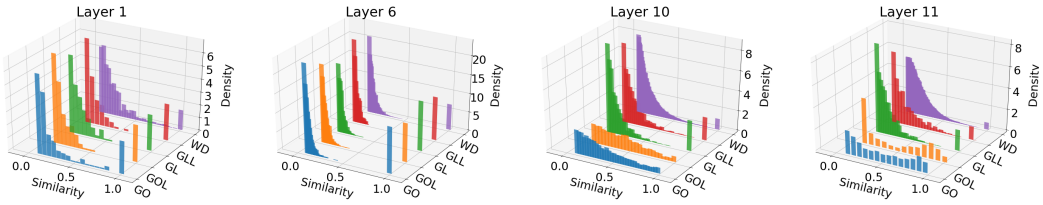

Figure 6: Output channel cosine similarity histogram obtained with different regularizers. Labels: **GO**:GrOWL, **GOL**:GrOWL+ℓ_2, **GL**:group-Lasso, **GLL**:group-Lasso+ℓ_2, **WD**:weight decay.

only loosely correlated. We illustrate this reasoning in Fig. 6, showing the cosine similarity between the vectorized output channels of layers 1, 6, 10, and 11, at the end of the training phase; it can be seen that the outputs of layers 10 and 11 have many more significant similarities than that of layer 6. Although the output channels of layer 1 also have certain similarities, as seen in Table 2, neither GrOWL ($+\ell_2$) nor weight decay tends to tie the associated weights. This may mean that the network is maintaining the diversity of the inputs in the first few convolutional layers.

Although GrOWL and weight decay both encourage parameter tying in layers 9-13, weight decay does it with less intensity and does not yield a sparse model, thus it cannot significantly compress the network. Li et al. (2016) propose to prune small weights after the initial training phase with weight decay, then retrain the reduced network; however, this type of method only achieves roughly $3X$ compression[4]. As mentioned by Li et al. (2016), layers 3-7 can be very sensitive to pruning; however, both GrOWL ($+\ell_2$) and group-Lasso ($+\ell_2$) effectively compress them with a minor accuracy loss.

On the other hand, similar to what we observed by running the simple fully connected neural network on MNIST, the accuracy and memory trade-off imrpoves significantly by applying GrOWL or group-Lasso together with ℓ_2. However, Table 2 also shows that the accuracy vs. memory trade-off achieved by GrOWL ($+\ell_2$) and group-Lasso ($+\ell_2$) are almost the same. We suspect that this is caused by the fact that the CIFAR-10 dataset is simple enough that one could still expect a good performance after strong network compression. We believe this gap in the compression vs accuracy trade-off can be further increased in larger networks on more complex datasets. We leave this question for future work.

5 Conclusion

We have proposed using the recent GrOWL regularizer for simultaneous parameter sparsity and tying in DNN learning. By leveraging on GrOWL's capability of simultaneously pruning redundant parameters and tying parameters associated with highly correlated features, we can significantly reduce the model complexity, with a slight or even no loss in generalization accuracy. We evaluate the proposed method on both a fully connected neural network and a deep convolutional neural network. The results show that GrOWL can compress large DNNs by a factor of 11.4 to 14.5.

The correlation patterns identified by GrOWL are close to those of the input features to each layer. This may be important to reveal the structure of the features, contributing to the interpretability of deep learning models. On the other hand, by automatically tying together the parameters corresponding to highly correlated features, GrOWL alleviates the negative effect of strong correlations that might be induced by the noisy input or the co-adaption tendency of deep neural networks.

However, the gap in the accuracy and memory trade-off obtained by applying GrOWL and group-Lasso to compress neural networks gets decreased as we move to large DNNs. Although we suspect this can be caused by running a much larger network on a simple dataset, this motivates us to explore different ways to apply GrOWL to compress neural networks. One possible method is applying GrOWL within each neuron by predefining each 2D convolutional filter as a group (instead all 2D convolutional filters corresponding to the same input features). By doing so, we encourage parameter

[4]Although parameter sharing is not considered by Li et al. (2016), according to Table 2, pruning following weight decay together with parameter sharing still cannot compress the network as much as GrOWL.

sharing among much smaller units, which in turn would further improve the diversity vs parameter sharing trade-off. We leave this as future work.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Alireza Aghasi, Nam Nguyen, and Justin Romberg. Net-trim: A layer-wise convex pruning of deep neural networks. *arXiv preprint arXiv:1611.05162*, 2016.

Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pp. 2270–2278, 2016.

H. Bauschke and P. Combettes. *Convex analysis and monotone operator theory in Hilbert spaces*. Springer, 2011.

Yoshua Bengio and James S Bergstra. Slow, decorrelated features for pretraining complex cell-like networks. In *Advances in neural information processing systems*, pp. 99–107, 2009.

Małgorzata Bogdan, Ewout van den Berg, Chiara Sabatti, Weijie Su, and Emmanuel J Candès. Slope–adaptive variable selection via convex optimization. *The annals of applied statistics*, 9(3): 1103, 2015.

Howard D Bondell and Brian J Reich. Simultaneous regression shrinkage, variable selection, and supervised clustering of predictors with oscar. *Biometrics*, 64(1):115–123, 2008.

P. Bühlmann, P. Rütimann, S. van de Geer, and C.-H. Zhang. Correlated variables in regression: clustering and sparse estimation. *Journal of Statistical Planning and Inference*, 143:1835–1871, 2013.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *ICML*, 2015.

Christopher Clark and Amos Storkey. Teaching deep convolutional neural networks to play go. *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

Michael Cogswell, Faruk Ahmed, Ross Girshick, Larry Zitnick, and Dhruv Batra. Reducing overfitting in deep networks by decorrelating representations. *arXiv preprint arXiv:1511.06068*, 2015.

Maxwell Collins and Pushmeet Kohli. Memory bounded deep convolutional networks. In *arXiv:1412.1442*, 2014.

J. de Leeuw, K. Hornik, and Patrick Mair. Isotone optimization in R: Pool adjacent-violators algorithm (PAVA) and active set methods. *Statistical Software*, 32:1–24, 2009.

Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pp. 2148–2156, 2013.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pp. 1269–1277, 2014.

Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

Mário AT Figueiredo and Robert D Nowak. Ordered weighted l1 regularized regression with strongly correlated covariates: Theoretical aspects. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pp. 930–938, 2016.

Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

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

Song Han, Huizi Mao, and William Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization, and Huffman coding. In *ICLR*, 2016.

Babak Hassibi, David G Stork, et al. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, pp. 164–164, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Y. LeCun. *Modèles connexionnistes de l'apprentissage*. PhD thesis, Université Pierre et Marie Curie, Paris, France, 1987.

Yann LeCun, John Denker, Sara Solla, Richard Howard, and Lawrence Jackel. Optimal brain damage. In *Neural Information Processing Systems (NIPS)*, volume 2, pp. 598–605, 1989.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Kenton Murray and David Chiang. Auto-sizing neural networks: With applications to n-gram language models. *arXiv preprint arXiv:1508.05051*, 2015.

Urvashi Oswal, Christopher Cox, Matthew Lambon-Ralph, Timothy Rogers, and Robert Nowak. Representational similarity learning with application to brain networks. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1041–1049, 2016.

Pau Rodríguez, Jordi Gonzàlez, Guillem Cucurull, Josep M Gonfaus, and Xavier Roca. Regularizing cnns with locally constrained decorrelations. *arXiv preprint arXiv:1611.01967*, 2016.

D. Rumelhart and J. McClelland. *Parallel Distributed Processing*. MIT Press, 1986.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Simone Scardapane, Danilo Comminiello, Amir Hussain, and Aurelio Uncini. Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (B)*, 58:267–288, 1996.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.

Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

Sergey Zagoruyko. 92.45% on cifar-10 in torch. http://torch.ch/blog/2015/07/30/cifar.html, 2015.

Xiangrong Zeng and Mário Figueiredo. The ordered weighted ℓ_1 norm: Atomic formulation, projections, and algorithms. *arXiv*, 1409.4271, 2015.

Xiangrong Zeng and Mário AT Figueiredo. Decreasing weighted sorted l1 regularization. *IEEE Signal Processing Letters*, 21(10):1240–1244, 2014.

Hao Zhou, Jose M Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *European Conference on Computer Vision*, pp. 662–677. Springer, 2016.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

APPENDIX A PROXGROWL

Various methods have been proposed to compute the proximal mapping of OWL (ProxOWL) . It has been proven that the computation complexity of these methods is $O(n \log n)$ which is just slightly worse than the soft thresholding method for solving ℓ_1 norm regularization. In this paper, we use Algorithm 2 that was originally proposed in Bogdan et al. (2015).

Algorithm 2 ProxGrOWL Bogdan et al. (2015) for solving $\text{prox}_{\eta,\Omega_\lambda}(z)$

Input: z and λ
Let $\widetilde{\lambda} = \eta\lambda$ and $\widetilde{z} = |Pz|$ be a nonincreasing vector, where P is a permutation matrix.
while $\widetilde{z} - \widetilde{\lambda}$ is not nonincreasing: **do**
Identify strictly increasing subsequences, *i.e.,* segments $i : j$ such that

$$\widetilde{z}_i - \widetilde{\lambda}_i < \widetilde{z}_{i+1} - \widetilde{\lambda}_{i+1} < \widetilde{z}_j - \widetilde{\lambda}_j \tag{12}$$

Replace the values of $\widetilde{z}$ and $\widetilde{\lambda}$ over such segments by their average value: for $k \in \{i, i+1, \cdots, j\}$

$$\widetilde{z}_k \leftarrow \frac{1}{j-i+1} \sum_{i \le k \le j} \widetilde{z}_k, \qquad \widetilde{\lambda}_k \leftarrow \frac{1}{j-i+1} \sum_{i \le k \le j} \widetilde{\lambda}_k \tag{13}$$

end while
Output: $\widehat{z} = \text{sign}\,(z)\, P^T(\widetilde{z} - \widetilde{\lambda})_+$.

APPENDIX B AFFINITY PROPAGATION

Affinity Propagation is a clustering method based on sending messages between pairs of data samples. The idea is to use these messages to determine the most representative data samples, which are called exemplars, then create clusters using these exemplars.

Provided with the precomputed *data similarity* $s(i,j), i \neq j$ and *preference* $s(i,i)$, there are two types information being sent between samples iteratively: 1) *responsibility* $r(i,k)$, which measures how likely that sample k should be the exemplar of sample i; 2) *availability* $a(k,i)$, which is the evidence that sample i should choose sample k as its exemplar. The algorithm is described in 3.

Algorithm 3 Affinity Propagation Frey & Dueck (2007)

Initialization: $r(i,k) = 0, a(k,i) = 0$ for all i, k
while not converge **do**
Responsibility updates:

$$r(i,k) \leftarrow s(i,k) - \max_{j \neq k}(a(j,i) + s(i,j))$$

Availability updates:

$$a(k,k) \leftarrow \sum_{j \neq k} \max\{0, r(j,k)\}$$

$$a(k,i) \leftarrow \min\left(0, r(k,k) + \sum_{j \notin \{k,i\}} \max\{0, r(j,k)\}\right)$$

end while
Making assignments:

$$c_i^* \leftarrow \arg\max_k r(i,k) + a(k,i)$$

Unlike k-means or agglomerative algorithm, Affinity Propagation does not require the number of clusters as an input. We deem this as a desired property for enforcing parameter sharing in neural network compression because it's impossible to have the exact number of clusters as a prior information. In practice, the input *preference* of Affinity Propagation determines how likely each sample will be chosen as an exemplar and its value will influence the number of clusters created.

APPENDIX C MNIST SPARE PATTERN PLOT

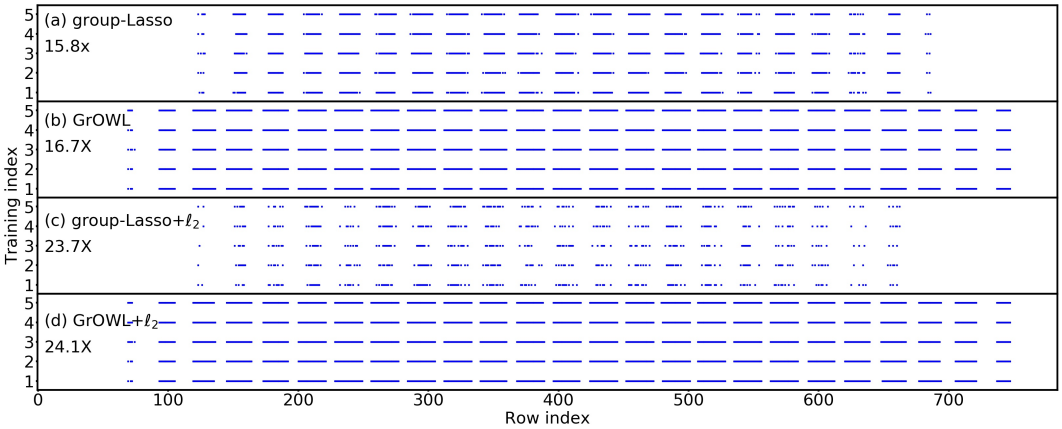

Figure 7: Similarity maps for parameter rows obtained by group-Lasso, GrOWL, group-Lasso+ℓ_2, GrOWL+ℓ_2. The mean ratio of changed indices2 are 11.09%, 0.59%, 32.07%, and 0.62% for the four different regularization methods, respectively.

APPENDIX D VGG-16 ON CIFAR-10

Table 3: Network statistics of VGG-16.

Layers	Output $w \times h$	#Channels in&out	#Params
conv1	32×32	3, 64	1.7E+03
*conv2	32×32	*64*, 64	3.7E+04
*conv3	16×16	*64*, 128	7.4E+04
*conv4	16×16	*128*, 128	1.5E+05
*conv5	8×8	*128*, 128	2.9E+05
*conv6	8×8	*128*, 256	5.9E+05
*conv7	8×8	*256*, 256	5.9E+05
*conv8	4×4	*256*, 512	1.2E+06
*conv9	4×4	*512*, 512	2.4E+06
*conv10	4×4	*512*, 512	2.4E+06
*conv11	2×2	*512*, 512	2.4E+06
*conv12	2×2	*512*, 512	2.4E+06
*conv13	2×2	*512*, 512	2.4E+06
*fc	1	*512*, 512	1.0E+06
sofrmax	1	512, 10	5.1E+03

Table 4: VGG: Clustering rows over different threshold values. For each experiment, we report clustering accuracy (**A**), compression rate (**C**), and parameter sharing **(S)** of layers 9-14. For each regularizer, we use different threshold values to cluster the rows at the end of initial training process, and then retrain the neural network correspondingly. The results are reported as the averages over 5 training and retraining runs..

Threshold Value	**0.6**	**0.7**	**0.8**	**0.9**
GrOWL	92.2%, 13.6, 3.5	92.2%, 12.5, 2.6	92.2%, 11.4, 2.1	92.2%, 10.9, 1.7
Group Lasso	92.2%, 12.1, 1.1	92.0%, 11.4, 1.1	92.1%, 11.0, 1.0	92.2%, 9.5, 1.0
GrOWL + ℓ_2	92.7%, 14.7, 2.3	92.5%, 15.4, 2.9	92.7%, 14.5, 2.3	92.6,%, 13.5, 1.8
GrLasso + ℓ_2	92.7%, 14.8, 1.2	92.7%, 14.5, 1.1	92.7%, 14.5, 1.0	92.6%, 14.3, 1.0
Weight Decay	93.2%, 1.8, 2.2	93.4%, 1.5, 1.7	93.1%, 1.3, 1.4	93.3%, 1.1, 1.1