Group Pruning using a Bounded-ℓ_p norm for Group Gating and Regularization

Chaithanya Kumar Mummadi[1,2][0000−0002−1173−2720], Tim Genewein[1][0000−0001−8039−4027]*, Dan Zhang[1][0000−0003−0930−9162], Thomas Brox[2][0000−0002−6282−8861], and Volker Fischer[1][0000−0001−5437−4030]

[1] Bosch Center for Artificial Intelligence, Robert Bosch GmbH, Germany
[2] University of Freiburg, Germany

Abstract. Deep neural networks achieve state-of-the-art results on several tasks while increasing in complexity. It has been shown that neural networks can be pruned during training by imposing sparsity inducing regularizers. In this paper, we investigate two techniques for group-wise pruning during training in order to improve network efficiency. We propose a gating factor after every convolutional layer to induce channel level sparsity, encouraging insignificant channels to become exactly zero. Further, we introduce and analyse a bounded variant of the ℓ_1 regularizer, which interpolates between ℓ_1 and ℓ_0-norms to retain performance of the network at higher pruning rates. To underline effectiveness of the proposed methods, we show that the number of parameters of ResNet-164, DenseNet-40 and MobileNetV2 can be reduced down by 30%, 69%, and 75% on CIFAR100 respectively without a significant drop in accuracy. We achieve state-of-the-art pruning results for ResNet-50 with higher accuracy on ImageNet. Furthermore, we show that the light weight MobileNetV2 can further be compressed on ImageNet without a significant drop in performance.

1 Introduction

Modern deep neural networks are notoriously known for requiring large computational resources, which becomes particularly problematic in resource-constrained domains, such as in automotive, mobile or embedded applications. Neural network *compression* methods aim at reducing the computational footprint of a neural network while preserving task performance (e.g. classification accuracy) [4, 34]. One family of such methods, *Network pruning*, operates by removing unnecessary weights or even whole neurons or convolutional featuremaps ("channels") during or after training, thus reducing computational resources needed at test time or deployment. A simple relevance-criterion for pruning weights is weight-magnitude: "small" weights contribute relatively little to the overall computation (dot-products and convolutions) and can thus be removed.

However, weight-pruning leads to unstructured sparsity in weight matrices and filters. While alleviating storage demands, it is non-trivial to exploit

* Currently at DeepMind

unstructured sparsity for reducing computational burden during forward-pass operation. This effect becomes even more pronounced on today's standard hardware for neural network computation (GPUs), which is typically designed for massively parallel operation. In contrast to individual-weight pruning, neuron- and featuremap-pruning allows dropping whole slices of weight matrices or tensors, which straightforwardly leads to a reduction of forward-pass FLOPS, energy consumption as well as on- and off-line memory requirements. However, it is more intricate to determine the relevance of whole neurons/featuremaps than that of weights.

In this paper, we propose and evaluate a method for *group-wise* pruning. A group typically refers to all weights that correspond to a neuron or convolutional filter, but could in principle also be chosen to correspond to different sub-structures such as larger parts of a layer or even whole blocks/layers in architectures with skip-connections. The central idea of our method is the addition of a "trainable gate", that is a *parameterized, multiplicative factor*, per group. During training, the gate-parameter is learned for each gate individually, allowing the network to learn the relevance of each neuron/featuremap. After training, groups of low relevance can be straightforwardly identified and pruned without significant loss in accuracy. The resulting highly structured sparsity patterns can be readily used to reduce the size of weight-matrices or -tensors. An important aspect of our method is that we use a sparsity-inducing regularizer during training to force a maximally large number of gates towards zero. We empirically compare different choices for this sparsity-inducing regularizer and in addition to previously proposed ℓ_1 or ℓ_2 norms, we propose and evaluate a smoothened version of the ℓ_0 norm (which can also be viewed as a saturating version of an ℓ_p norm). The latter allows for a certain decoupling of parameter-importance and parameter-magnitude, which is in contrast to standard regularizers that penalize parameters of large magnitude regardless of their importance.

- We investigate the effect of group pruning using bounded ℓ_p norms for group gating and regularization on different network architectures (LeNet5, DenseNet, ResNet and MobileNetV2) and data-sets (MNIST, CIFAR100, ImageNet) achieving comparable or superior compression and accuracy.
- We show that our gating function drives the gating factors to become exactly zero for the insignificant channels during training.
- Applying ℓ_2 regularizer on our gating parameters, rather than on weights, leads to significant pruning for ResNet and DenseNet without a drop in accuracy and further improves the accuracy of MobileNetV2 on both CIFAR100 and ImageNet.
- We also propose a bounded variant of the common ℓ_1 regularizer to achieve higher pruning rates and retain generalization performance.

2 Related work

Neural Network compression. Most approaches in the literature resort to *quantization* and/or *pruning*. In this context, quantization refers to the reduction

of required bit-precision of computation — either of weights only [3, 9, 12, 40] or both weights and activations [5, 33, 20, 11, 38]. Network pruning attempts to reduce the number of model parameters and is often performed in a single step after training, but some variants also perform gradual pruning during training [7, 12, 14] or even prune and re-introduce weights in a dynamic process throughout training [13, 10]. In contrast to individual weight pruning [12], group-pruning methods (pruning entire neurons or feature-maps that involve groups of weights) lead to highly structured sparsity patterns which easily translate into on-chip benefits during a forward-pass [36, 41, 2].

Pruning and quantization can also be combined [12, 35, 6, 1]. Additionally, the number of weights can be reduced *before* training by architectural choices as in SqueezeNet [21] or MobileNets [17]. As we show in our experiments, even parameter-optimized architectures such as MobileNets can still benefit from post-training pruning.

Relevance determination and sparsity-inducing regularization. Many pruning methods evaluate the relevance of each unit (weight, neuron or featuremap) and remove units that do not pass a certain relevance-threshold [12, 10, 13, 24]. Importantly, optimizing the relevance-criterion that is later used for pruning thus becomes a secondary objective of the training process — in this case via weight-magnitude regularization. An undesirable side-effect of ℓ_1- or ℓ_2-weight-decay [15] when used for inducing sparsity is that important, non-pruned weights still get penalized depending on their magnitude, leading to an entanglement of parameter-importance and magnitude. An ideal sparsity-inducing regularizer would act in an (approximately) binary fashion, similar to how the ℓ_0 norm simply counts number of non-zero parameters, but is not affected by the magnitude of the non-zero parameters. The problem of determining the relevance of model parameters has also been phrased in a Bayesian fashion via *automatic relevance determination* (ARD) and sparse Bayesian learning [22, 28, 31], which has recently been successfully applied to weight-pruning [29], weight ternarization [1] and neuron-/featuremap-pruning by enforcing group-sparsity constraints [8, 25, 6, 32]. These methods require (variational) inference over the parameter posterior instead of standard training.

Neuron-/featuremap-pruning. Determining the importance of neurons or feature maps is non-trivial [36, 41, 2]. Approaches are based on thresholding the norm of convolutional kernels or evaluating activation-statistics. However, both approaches come with certain caveats and shortcomings [30, 39]. Some methods try to explicitly remove neurons that do not have much impact on the final network prediction [18, 23, 30]. Other methods propose a more complex optimization procedure with intermediate pruning steps and fine-tuning [16, 27], such that the non-pruned network can gradually adjust to the missing units.

Our approach is closely related to [26], who also use trainable, multiplicative gates for neuron-/featuremap-pruning. However, in their formulation gates Bernoulli random variables. Accordingly, learning of their gate parameters is done via (variational) Bayesian inference. In contrast, our method allows network training in a standard-fashion (with an additional regularizer term) without

requiring sampling of gate parameters, or computing expected gradients across such samples. Other closely related works are [24, 39], who induce sparsity on the multiplicative scaling factor γ of Batch Normalization layers and later use the magnitude of these factors for pruning channels/featuremaps. Similarly, [19] use a trainable, linear scaling factor on neurons / featuremaps with an ℓ_1-norm sparsity-inducing regularizer. We perform experiments to directly compare our method against all the above closely related works. Additionally, we reimplement the technique proposed by [24] and treat it as a baseline to compare our results against it in all experiments.

3 Bounded-$\ell_{p,0}$ norm

The p-norm (a.k.a. ℓ_p-norm) and 0-norm of a vector $x \in \mathbb{R}^n$ of dimension n are respectively defined as:

$$\|x\|_p := \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} \qquad \|x\|_0 := \sum_{i=1}^{n} (1 - \mathbf{1}_0(x_i)). \tag{1}$$

Here, $\mathbf{1}_a(b)$ being the function which is one iff $a = b$ and zero otherwise. While the p-norms constitute norms in the mathematical sense, the 0-norm (a.k.a. *discrete metric*), does not due to the violation of the triangle inequality. It is constant almost everywhere and hence gradient based optimization techniques are unusable. We use a differentiable function adapted from [37], which around 0 interpolates, controlled by a parameter $\sigma > 0$, between the p- and 0-norm:

Definition 1. *For $\sigma > 0$, we call the mapping $\|.\|_{bound\text{-}p,\sigma} : \mathbb{R}^n \to \mathbb{R}_+$ with*

$$\|x\|_{bound\text{-}p,\sigma} := \sum_{i=1}^{n} 1 - exp\left(-\frac{|x_i|^p}{\sigma^p}\right) \tag{2}$$

*the **bounded-$\ell_{p,0}$ norm** or **bounded-ℓ_p norm**. Fig. 1 illustrates the bounded-$\ell_{p,0}$ norm with $p = 1, 2$ and different σ. One sees that $\|x\|_{bound\text{-}p,\sigma}$ is bounded to $[0, n)$ and differentiable everywhere except $x_i = 0$ for one or more coefficients of x. Further, in contrast to the 0-norm, it has a non-zero gradient almost everywhere.*

Lemma 1. *The bounded-$\ell_{p,0}$ norm has the following properties:*

- *For $\sigma \to 0^+$ the bounded-norm converges towards the 0-norm:*

$$\lim_{\sigma \to 0^+} \|x\|_{bound\text{-}p,\sigma} = \|x\|_0. \tag{3}$$

- *In case $|x_i| \approx 0$ for all coefficients of x, the bounded-norm of x is approximately equal to the p-norm of x weighted by $1/\sigma$:*

$$\|x\|_{bound\text{-}p,\sigma} \approx \left\|\frac{x}{\sigma}\right\|_p^p \tag{4}$$

Proof. See Section A1 for proof.

4 Methodology

With the use of the bounded-ℓ_p norm introduced in the previous section, we subsequently present a simple and straightforward technique to perform *group wise pruning* in deep CNNs. Here, *group* is referred to as a set of weights, e.g., a filter in a convolutional layer associated to a feature map or, in case of a fully connected layer, a single target neuron.

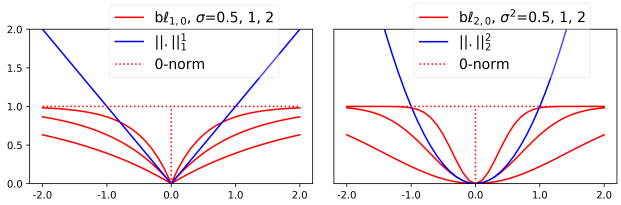

Fig. 1. Illustration of bounded-$\ell_{p,0}$ (b$\ell_{p,0}$) norms with $p \in \{1, 2\}$: Interpolation from ℓ_1-norm to 0-norm (left) and from ℓ_2-norm to 0-norm (right) with different σ.

Bounded-ℓ_1 regularizer: It is a common practice to use sparsity inducing ℓ_1 penalty to shrink parameters during training. [24] has performed channel-wise pruning by imposing ℓ_1 penalty on the scaling factor γ of Batch Normalization (BN) layers that correspond to featuremaps in convolutional layers. We denote these scaling factors as *linear gates* in this work. Thus, the total loss L consists of the standard empirical loss l and an additional ℓ_1 penalty on the linear gates:

$$\mathcal{L} = \sum_{(x,y)} l\left(f\left(x, W\right), y\right) + \lambda \sum_{\gamma \in G} |\gamma| \tag{5}$$

where f denotes the deep neural network, x, y denote training input and target, W denotes the network weights, γ denotes a single scaling factor from the aggregated set of all linear gates G. The ℓ_1 regularizer acts upon all linear gates and pushes them towards zero. The channels with linear gates whose magnitude is less than the relevance threshold are then pruned to obtain a narrow network. Here, the linear gates should accomplish two different tasks i) get optimized along the other network parameters to improve the network performance and ii) shrink down towards zero to induce channel level sparsity in the network. The hyperparameter λ defines the strength of the regularizer and controls the trade-off between primary objective and ℓ_1 penalty. Increasing λ would yield higher pruning rates at the cost of reduced network performance. The ℓ_1 regularizer penalizes each parameter at a same rate irrespective of its role and importance in accomplishing the primary objective. In general, not all parameters should receive equal penalty. We address this issue by employing a norm as defined

in Equation (2) as a sparsity inducing regularizer with $p = 1$ and denote it as bounded-ℓ_1 regularizer as it is bounded to $[0, 1]$.

Figure 1 (left) shows that the bounded-ℓ_1 norm is a variant of the normal ℓ_1 norm and both penalize larger parameters. Importantly for the bounded variant, the penalty on larger weights does not increase as strong as for the normal norm, and only smaller weights are penalized comparably. Larger parameters, for which the bounded variant saturates, become primarily subject to the task loss. In other words, for the bounded variant, the penalty for large parameters becomes decoupled from the size of the parameters and converges to a constant value whereas for small parameters the penalty is relative large and forces them to even smaller values. Similar to the ℓ_1 penalty, the bounded ℓ_1 norm can be added as a regularization term in the objective function.

$$\mathcal{L}^* = \sum_{(x,y)} l\left(f\left(x, W\right), y\right) + \lambda \sum_{\gamma \in G} \left[1 - e^{-\frac{|\gamma|}{\sigma}}\right] \tag{6}$$

The gradient of the parameter γ w.r.t. ℓ_1 and bounded-ℓ_1 regularization equals:

$$\frac{\partial \mathcal{L}_{\text{reg}}}{\partial \gamma} = \lambda \cdot \text{sign}\left(\gamma\right), \quad \frac{\partial \mathcal{L}^*_{\text{reg}}}{\partial \gamma} = \lambda \cdot \text{sign}\left(\gamma\right) \frac{e^{-\frac{|\gamma|}{\sigma}}}{\sigma} \tag{7}$$

The above equations indicate that the ℓ_1 norm updates gradients at a scale of λ irrespective of their magnitude. On the other hand, bounded-ℓ_1 norm provides no or small gradients for parameters with higher magnitude and large gradients for smaller parameters. In this manner, parameters with larger values receive gradients mainly from the first part of $\mathcal{L}^*$, being informative to accomplish the primary classification task.

Another interesting property of such norm is: The hyperparameter σ scales the regularization strength by controlling the interpolation between the ℓ_1- and 0-norm. As σ gets smaller, the bounded-ℓ_1 norm converges to the 0-norm according to Lemma 1. Larger σ allows regularization of all parameters whereas smaller σ guides the regularizer to penalize only parameters of smaller magnitude while liberating the larger ones. Larger values of σ enforce weaker regularization and smaller values enforce stronger regularization (also compare Fig. 1).

Given the behavior of σ, we can schedule it by gradually reducing its value during training. In doing so, the norm initially regularizes a larger number of parameters and then gradually shrinks down the insignificant ones towards zero while simultaneously filtering out the important ones. We can imagine the scheduling of σ as opening the gates of the 0-norm to make it differentiable which allows the insignificant parameters to fall into the valley of the norm and gradually close the gates to leave out the important parameters. It is fairly straightforward to include the hyperparameter σ also in the case of the ℓ_1 norm by replacing $|\gamma|$ with $\frac{|\gamma|}{\sigma}$ in Equation (5) but it is similar to scaling the hyperparameter λ to $\frac{\lambda}{\sigma}$ in this case. The scheduling of σ in ℓ_1-norm increases its regularization strength and pushes down all the parameters towards zero which affects task performance of the network.

Bounded-ℓ_2 for group gating: Both the ℓ_1 and bounded-ℓ_1 regularizers bring down the scalar parameters towards zero but never make them exactly zeros (refer Figure 3). This limitation always demands the setting of a relevance threshold to prune the parameters and then later requires fine-tuning for a number of iterations to stabilize the task performance of the pruned network. To this end, we propose to use the same bounded-ℓ_p norm that is defined in Equation (2) as an additional layer in the network with $p = 2$ and $\sigma = 1$. To this, we refer to as a gating layer of *exponential gates* (with gating parameters g) which is placed after every convolutional or fully connected layer or before a BN layer in the network. This layer serves as a multiplicative gating factor for every channel in the preceding convolutional layer. The gating layer has the same number of gates as the number of channels where each gate gets multiplied to an output channel of a convolutional layer.

$$x = \text{conv}\left(input\right); \quad y_k = x_k \cdot \left(1 - e^{-g_k^2}\right) \tag{8}$$

where x and y are the output of the convolutional and gating layer respectively and k indexes the channel of the convolutional layer. Since the gates are added as a layer in the network, we train the gating parameters g together with the network weights W. In contrast to the *linear gates* γ of BN, we impose the penalty only on the parameters g of *exponential gates*, yielding the loss function:

$$\mathcal{L} = \sum_{(x,y)} l\left(f\left(x, g, W\right), y\right) + \lambda \sum_{g \in G} R\left(g\right) \tag{9}$$

The first part of the loss function corresponds to the standard empirical loss of the neural network and $R(.)$ is the penalty term on the gating parameters g which could be either ℓ_2, ℓ_1, or the bounded-ℓ_1 regularizer. Two interesting properties of the *exponential gates* which makes them distinctive from the *linear gates* are i) its values are bounded to the range $[0, 1)$, ii) the quadratic exponential nature of the gates fused with the regularizer shrink down the outcome of the gates towards zero rapidly. The regularized *exponential gates* which are jointly optimized with the network weights act as a channel selection layer in the network. These gates actively differentiate the insignificant channels from significant ones during the training phase and gradually turn them off without affecting the network's performance. In Section 5, we empirically show that these exponential gating layers assist the regularizers to drive the insignificant channels to become exactly zero and later compress the network after removing such channels.

The *exponential gating layer* can be added to the network with or without BN. In case the gating layer is followed by BN, the statistics from the nulled-out channels remains constant across all the mini-batches since the gate is deterministic and gets multiplied to every input sample. Thus, both the running estimates of its computed mean and variance of the BN is zero for the nulled-out channels. The multiplicative scaling factor γ of BN does not show any effect on those channels but its additive bias β might change the zero channels to non-zero. This can be seen as adding a constant to the zero channels which can be easily

alleviated by few iterations of fine-tuning the pruned network. In case the gating layers are added to a CNN without BN, we can prune channels in the network without any need of explicit fine-tuning since the insignificant channels become exactly zero after getting multiplied with the gates during the training phase. As a final note, the additional *exponential gating layer* increases the number of trainable parameters in the network but these gates can be merged into the weights of the associated convolutional filter after pruning.

In next section, we empirically evaluate the above-proposed techniques to achieve channel level sparsity, namely, i) bounded-ℓ_1 norm to prune a larger number of parameters and preserve the task accuracy, and ii) additional gating layer in CNNs to support the regularizers to achieve exactly zero channels.

5 Experimental Results

We demonstrate the significance of both, the *exponential gating layer* and the *bounded-*ℓ_1 regularizer, on different network architectures and datasets, i.e., LeNet5-Caffe on MNIST, DenseNet-40, ResNet-164, MobileNetV2 on CIFAR100 and ResNet-50, MobileNetV2 on ImageNet dataset. We refer to Sec. A2 for the experiment details such as data preprocessing, architecture configuration, and hyperparameter selection. We use the threshold point 10^{-4} on the linear gates and threshold zero on the exponential gates to prune the channels.

CIFAR100 The results are summarized in Figure 2. We compare the trade-off between classification accuracy on test data against the pruning rates obtained from different regularizers and gates. We report the average results over 3 different runs. Here, $\sigma_{constant}$ refers to the hyperparameter σ that is set to a constant value throughout the training process. We also investigated the influence of scheduling σ in case of bounded-ℓ_1 regularizer and compared the results against scheduling σ in ℓ_1 regularizer.

From Figure 2, it can be seen that the bounded-ℓ_1 regularizer on the linear gates results in a higher pruning rate with an accuracy comparable to the ℓ_1 regularizer in ResNet-164 and provides a higher accuracy than the ℓ_1 regularizer in MobileNetV2. On the other hand, the addition of exponential gating layers in ResNet-164 and MobileNetV2 greatly increases the pruning rates and accuracy upon the linear gate. The bounded-ℓ_1 regularizer further improves the accuracy of ResNet-164 with exponential gating layer to 77.28% and 76.58% at different regularization strengths with pruning rates 30.73% and 47% respectively. In case of MobileNetV2, ℓ_1 on exponential gating layer results pruning rate of 75.83% with an accuracy 75.33%.

In contrast to the other networks, the pruning results of bounded-ℓ_1 regularizer and exponential gating layer in DenseNet-40 are identical to the results of the ℓ_1 regularizer on linear gate. However, the addition of exponential gating layer when combined with the ℓ_2 regularizer encourages channel pruning with a marginal drop in performance in both ResNet-164 and DenseNet-40 architectures, whereas the gate improves the classification performance in case of MobileNetV2. We can also observe that scheduling σ for the ℓ_1 regularizer significantly drops the

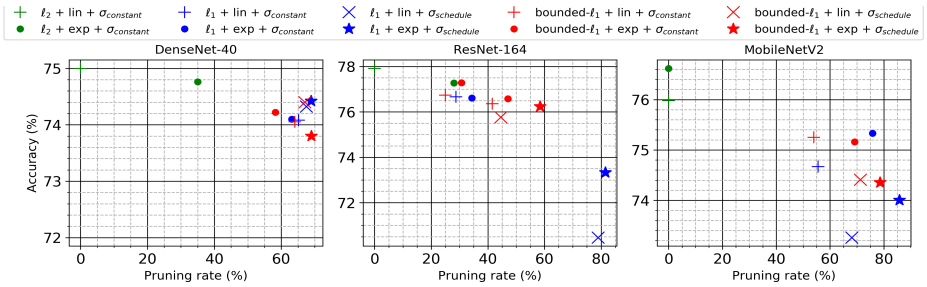

Fig. 2. Comparing trade-off between pruning rates and accuracies of different regularizers ℓ_2, ℓ_1 and bounded-ℓ_1 with different gates (linear, exponential) at constant and scheduled σ on DenseNet-40, ResNet-164 and MobileNetV2 on CIFAR100. In DenseNet-40, the scheduled ℓ_1 regularizer on exponential gate achieves slightly higher pruning and accuracy rate than the other methods. In ResNet-164, two identical markers represent settings with different regularization strengths. Here, bounded-ℓ_1 on exponential gate achieves higher pruning rates with approximately same line of accuracy with other methods. In MobileNetV2, bounded-ℓ_1 on linear gate outperforms ℓ_1 on linear gate in terms of accuracy with approximately similar pruning rate for both the cases of σ (constant and scheduled). However, ℓ_1 on exponential gate with constant σ preserves the accuracy with higher pruning rate. Thus, the networks with exponential gating layers has higher pruning rates than the linear gates with the accuracy close to baseline. On the other hand, bounded-ℓ_1 improves accuracy on linear gates in MobileNetV2 and on both gates in ResNet-164 when compared with ℓ_1 regularizer.

accuracy and increases the pruning rate in both MobileNetV2 and ResNet-164. Scheduling in the bounded-ℓ_1 regularizer also increases the pruning rate while retaining the accuracy close to the baseline margin. In MobilenetV2, scheduling the regularizer in bounded-ℓ_1 yields higher accuracy and pruning rate on linear gate when compared to the scheduled ℓ_1 regularizer. In ResNet-164, the pruning rate raises from 47% to 58.5% with an accuracy drop from 76.58% to 76.23% in case of exponential gate with scheduled bounded-ℓ_1 regularizer. On the other hand, the impact of the scheduler remains comparable, for both the regularizers in DenseNet-40 and scheduling ℓ_1 regularizer on exponential gate increases the pruning rate to 69% with 74.42% accuracy.

We compare pruning rates between linear and exponential gates and their accuracy trade-off at different threshold points in Figure 3. We prune channels with gate values less or equal to the threshold and further fine-tune the network for a maximum of three epochs. Across the three different architectures both gates maintain the same accuracy until a critical threshold. The pruning rate of the exponential gates are significantly larger than the linear gates in ResNet-164, MobileNetV2 and comparable in DenseNet-40. In particular, the magnitude of non-zero exponential gates lies in $[10^{-3}, 0.1]$ and pruning at the threshold larger than 10^{-3} removes all channels in the network. Below 10^{-3} the exponential gates achieve the optimum performance, i.e., largest pruning rate without loss of the classification accuracy. It is noted that the threshold zero is attainable, indicating

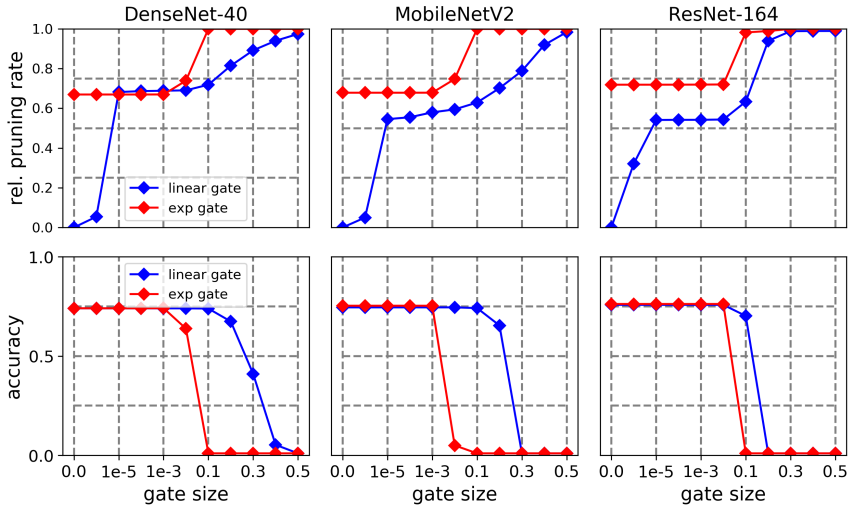

Fig. 3. Comparison of pruning rates (top row) and accuracies (bottom row) on CIFAR100 over different threshold points between linear gate (blue) and the exponential gate (red), both of which are applied in combination with ℓ_1 regularization. Units that do not pass the threshold on the gate values $|g|$ for the linear gates and $(1 - e^{-g^2})$ for exponential gates are pruned. Here, pruning rate of networks with exponential gates is superior or comparable to the linear gate at different threshold values. With the exponential gates, the achieved best pruning rates are insensitive to the selection of the threshold within the range $[0, 10^{-3}]$. In particular, the threshold zero being (nearly) optimum indicates that the exponential gates can exactly zeroing out removable channels. This observation also holds when combining with the ℓ_2 and bounded-ℓ_1 regularizers.

that exponential gates can exactly null out removable channels. On contrary, the linear ones gate them with a sufficiently small value (about 10^{-5} in the case of Figure 3), thereby necessitating the search of a precise pruning threshold.

MNIST We also test our method on the MNIST dataset using the *LeNet5-Caffe* model. We compare our results with ℓ_0 regularization from [26]. We present different models that are obtained from different regularizers and the weight decay is set to be zero when using the ℓ_1 or bounded-ℓ_1 regularization. From the results shown in Table 1, it can be observed that network with *exponential gating layer* on different regularizers yield more narrow models than the previous method with lower test errors.

ImageNet We also present pruning results of ResNet-50 and MobileNetV2 for the ImageNet dataset. On ResNet-50, we primarily investigate the significance of the *exponential gating layer* with ℓ_1 and ℓ_2 regularization. From Figure 4, it

Table 1. Comparing pruning results of architecture *LeNet-5-Caffe 20-50-800-500* on MNIST dataset from different regularizers like ℓ_0 from [26] and ℓ_1, ℓ_2, Bounded-ℓ_1 on the network with *exponential gating layers*. We show the resulting architectures obtained from different pruning methods and their test error rate. It can be seen that our architectures are narrower than the one from previous method with comparable or smaller test error rates.

Method	Pruned architecture	Error(%)
ℓ_0, [26]	20-25-45-462	0.9
ℓ_0, [26]	9-18-65-25	1.0
ℓ_2, λ_2 = 5e-4	8-19-117-24	0.79
ℓ_1, λ_1 = 1e-3	8-13-37-25	0.98
bounded-ℓ_1, λ_1 = 4e-3	9-17-43-25	0.92
bounded-ℓ_1, λ_1 = 3e-3	9-20-54-27	0.67

Table 2. Results on MobileNetV2 trained for 100 epochs on ImageNet. *Bounded-ℓ_1 on linear gate* achieves higher accuracy than ℓ_1 *on linear gate* and closer to the standard training with reduced number of parameters. On the other hand, exponential gate with ℓ_1 regularizer reduces number of parameters without a significant drop in accuracy and improves accuracy when combined with ℓ_2 regularizer. Here M stands for Millions.

Network- MobileNetV2	Top-1 %	Top-5 %	#Params	#FLOPS
Standard training λ_2=1e-5	70.1	89.25	3.56 M	320.2 M
$\ell_1 + lin$, λ_1 = 5e-5, λ_2 = 1e-5	69.54	89.14	3.37 M	275.0 M
bounded-$\ell_1 + lin$, λ_1 = 5e-5, λ_2 = 1e-5	69.9	89.17	3.40 M	280.0 M
$\ell_2 + exp$, λ_2 = 4e-5	**70.7**	90.0	3.56 M	312.8 M
$\ell_1 + exp$, λ_1 = 5e-5, λ_2 = 4e-5	69.9	89.438	**3.00** M	280.0 M

can be seen that the *exponential gating layer* combined with the ℓ_2 regularizer outperforms ResNet-101(v1) from [39] in terms of pruning rate and accuracy. ℓ_1 regularization further penalizes the gating parameters and achieves 39% and 73% sparsity in the network with a drop of 1.3% and 5.4% Top-1 accuracy respectively at different regularization strengths. We compare these results and show that our method prunes more parameters than the previous pruning methods with the same line of accuracy.

On MobileNetV2, we compare ℓ_1 against bounded-ℓ_1 on linear gate and ℓ_1 against ℓ_2 on exponential gate. From Table 2, it can be observed that the *bounded-ℓ_1 on linear gate* achieves higher accuracy than its counterpart ℓ_1 *on linear gate* with a slightly higher number of parameters. On the other hand, ℓ_1 penalty on exponential gate prunes a larger number of parameters and approximately keeps

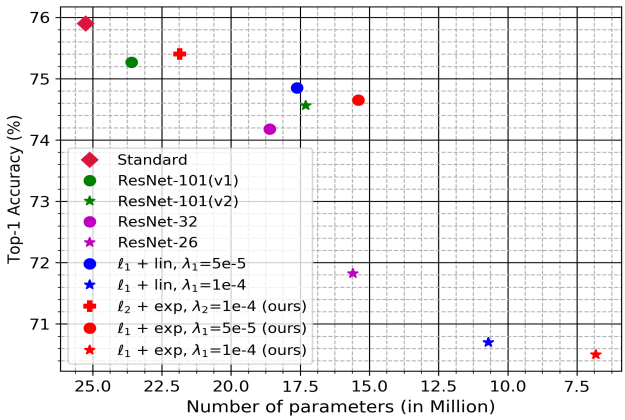

Fig. 4. Comparing our pruning results of ResNet-50 (ℓ_2 & ℓ_1 on *exponential gating layer*, ℓ_2 + *exp* and ℓ_1 + *exp*) on ImageNet dataset against the previous methods like ResNet-101(v1) and ResNet-101(v2) from [39], ResNet-32 and ResNet-26 which are obtained from block pruning on ResNet-50 [19]. We also compared our results against the method ℓ_1 *on linear gate* (ℓ_1 + *lin*) from [24] by implementing it on ResNet-50. Here, 'Standard' refers to the baseline model without pruning. We show the trade-off between top1-accuracy and number of remaining parameters of the network from different methods. It can be seen that the network from ℓ_2 + *exp* has about similar accuracy as ResNet-101(v1) but prunes 1.75M more parameters than the latter. Similarly, network from ℓ_1 + *exp* with $\lambda_1 = 5e\text{-}5$ and $1e\text{-}4$ prunes 2M and 3.5M more parameters respectively and has comparable accuracy with the other methods.

the accuracy of standard training whereas ℓ_2 on exponential gate improves the Top-1 accuracy by 0.6%.

6 Conclusion

In this work, we propose a straightforward and easy to implement novel approach for group-wise pruning of DNNs. We introduce *exponential gating layers*, which learn importance of the channels during training and drive the insignificant channels to become exactly zero. Additionally, we propose *bounded-*ℓ_1 regularization to penalize the gate parameters based on their magnitude. Different combinations of these techniques (gating functions and regularizers) are evaluated for a set of common DNN architectures for image classification. We found that the combination of exponential gating function with an ℓ_1 or its bounded variant is superior than the other approaches (cf. Fig. 2). Finally, these techniques result in higher compression rates with accuracy comparable to existing pruning approaches on ImageNet (cf. Fig. 4).

References

1. Achterhold, J., Koehler, J.M., Schmeink, A., Genewein, T.: Variational network quantization. ICLR2018 (2018)
2. Alvarez, J.M., Salzmann, M.: Learning the number of neurons in deep networks. In: Adv. in Neural Info. Process. Syst. (NIPS). pp. 2270–2278 (2016)
3. Chen, W., Wilson, J., Tyree, S., Weinberger, K., Chen, Y.: Compressing neural networks with the hashing trick. Int. Conf. on Machine Learning (ICML) pp. 2285–2294 (2015)
4. Cheng, Y., Wang, D., Zhou, P., Zhang, T.: A survey of model compression and acceleration for deep neural networks. arXiv:1710.09282 (2017)
5. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training deep neural networks with weights and activations constrained to + 1 or -1. arXiv:1602.02830 (2016)
6. Federici, M., Ullrich, K., Welling, M.: Improved Bayesian compression. arXiv:1711.06494 (2017)
7. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding small, trainable neural networks. arXiv:1803.03635 (2018)
8. Ghosh, S., Yao, J., Doshi-Velez, F.: Structured variational learning of Bayesian neural networks with horseshoe priors. arXiv:1806.05975 (2018)
9. Gong, Y., Liu, L., Yang, M., Bourdev, L.: Compressing deep convolutional networks using vector quantization. arXiv:1412.6115 (2014)
10. Guo, Y., Yao, A., Chen, Y.: Dynamic network surgery for efficient dnns. Adv. in Neural Info. Process. Syst. (NIPS) pp. 1379–1387 (2016)
11. Gysel, P., Pimentel, J., Motamedi, M., Ghiasi, S.: Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. IEEE Trans. on Neural Networks and Learning Syst. (2018)
12. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In: Int. Conf. on Learning Representations (ICLR) (2016)
13. Han, S., Pool, J., Narang, S., Mao, H., Tang, S., Elsen, E., Catanzaro, B., Tran, J., Dally, W.J.: Dsd: Regularizing deep neural networks with dense-sparse-dense training flow. Int. Conf. on Learning Representations (ICLR) (2017)
14. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. Adv. in Neural Info. Process. Syst. (NIPS) pp. 1135–1143 (2015)
15. Hanson, S.J., Pratt, L.Y.: Comparing biases for minimal network construction with back-propagation. Adv. in Neural Info. Process. Syst. (NIPS) pp. 177–185 (1989)
16. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: International Conference on Computer Vision (ICCV). vol. 2 (2017)
17. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861 (2017)
18. Hu, H., Peng, R., Tai, Y.W., Tang, C.K.: Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv:1607.03250 (2016)
19. Huang, Z., Wang, N.: Data-driven sparse structure selection for deep neural networks. arXiv:1707.01213 (2017)
20. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Quantized neural networks: Training neural networks with low precision weights and activations. J. of Machine Learning Research (JMLR) **18**(1), 6869–6898 (2017)

21. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. arXiv:1602.07360 (2016)
22. Karaletsos, T., Rätsch, G.: Automatic relevance determination for deep generative models. arXiv:1505.07765 (2015)
23. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. Int. Conf. on Learning Representations (ICLR) (2017)
24. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Computer Vision (ICCV), 2017 IEEE International Conference on. pp. 2755–2763. IEEE (2017)
25. Louizos, C., Ullrich, K., Welling, M.: Bayesian compression for deep learning. Advances in Neural Information Processing Systems (2017)
26. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through l0 regularization. ICLR 2018 (2018)
27. Luo, J.H., Wu, J., Lin, W.: Thinet: A filter level pruning method for deep neural network compression. ICCV2017 (2017)
28. MacKay, D.J.: Probable networks and plausible predictions - a review of practical Bayesian methods for supervised neural networks. Network: Computation in Neural Systems **6**(3), 469–505 (1995)
29. Molchanov, D., Ashukha, A., Vetrov, D.: Variational dropout sparsifies deep neural networks. ICML 2017 (2017)
30. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference. ICLR2017 (2017)
31. Neal, R.M.: Bayesian Learning for Neural Networks. Ph.D. thesis, University of Toronto (1995)
32. Neklyudov, K., Molchanov, D., Ashukha, A., Vetrov, D.: Structured Bayesian pruning via log-normal multiplicative noise. arXiv:1705.07283 (2017)
33. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: European Conference on Computer Vision. pp. 525–542. Springer (2016)
34. Sze, V., Chen, Y.H., Yang, T.J., Emer, J.: Efficient processing of deep neural networks: A tutorial and survey. arXiv:1703.09039 (2017)
35. Ullrich, K., Meeds, E., Welling, M.: Soft weight-sharing for neural network compression. ICLR 2017 (2017)
36. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in Neural Information Processing Systems. pp. 2074–2082 (2016)
37. Weston, J., Elisseeff, A., Schölkopf, B., Tipping, M.: Use of the zero-norm with linear models and kernel methods. J. of Machine Learning Research (JMLR)
38. Wu, S., Li, G., Chen, F., Shi, L.: Training and inference with integers in deep neural networks. arXiv:1802.04680 (2018)
39. Ye, J., Lu, X., Lin, Z., Wang, J.Z.: Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. arXiv:1802.00124 (2018)
40. Zhou, A., Yao, A., Guo, Y., Xu, L., Chen, Y.: Incremental network quantization: Towards lossless cnns with low-precision weights. arXiv:1702.03044 (2017)
41. Zhou, H., Alvarez, J.M., Porikli, F.: Less is more: Towards compact cnns. In: European Conference on Computer Vision. pp. 662–677. Springer (2016)

Group Pruning using a Bounded-ℓ_p norm for Group Gating and Regularization

Supplementary material

A1 Proof of Lemma 1 (Lemma 1):

To improve readability, we will restate Lemma 1 from the main text:
The mapping $\|.\|_{\text{bound-}p,\sigma}$ has the following properties:

- For $\sigma \to 0^+$ the bounded-norm converges towards the 0-norm:

$$\lim_{\sigma \to 0^+} \|x\|_{\text{bound-}p,\sigma} = \|x\|_0. \tag{1}$$

- In case $|x_i| \approx 0$ for all coefficients of x, the bounded-norm of x is approximately equal to the p-norm of x weighted by $1/\sigma$:

$$\|x\|_{\text{bound-}p,\sigma} \approx \left\| \frac{x}{\sigma} \right\|_p^p \tag{2}$$

Proof: The first statement Equation (1) can easily be seen using:

$$\lim_{\sigma \to 0} \exp(-\frac{|x_i|^p}{\sigma^p}) = \mathbf{1}_0(x_i)$$

For the second statement Equation (2) we use the taylor expansion of exp around zero to get:

$$\begin{aligned} \|x\|_{\text{bound-}p,\sigma} &= \sum_{i=1}^{n} 1 - \exp\left(-\frac{|x_i|^p}{\sigma^p}\right) \\ &= \sum_{i=1}^{n} 1 - \sum_{j=0}^{\infty} \left(-\frac{|x_i|^p}{\sigma^p}\right)^j \frac{1}{j!} \end{aligned} \tag{3}$$

For $|x_i| \approx 0$ we keep only the leading coefficient $j = 1$ yielding:

$$\|x\|_{\text{bound-}p,\sigma} \approx \sum_{i=1}^{n} \frac{|x_i|^p}{\sigma^p} = \left\| \frac{x}{\sigma} \right\|_p^p.$$

A2 Experiment Details

Both CIFAR100 and ImageNet datasets are augmented with standard techniques like random horizontal flip and random crop of the zero-padded input image and further processed with mean-std normalization. The architecture MobileNetV2 is originally designed for the task of classification on ImageNet dataset. We adapt the network[1] to fit the input resolution 32×32 of CIFAR100. ResNet-164 is a pre-activation ResNet architecture containing 164 layers with bottleneck structure while DenseNet with 40 layer network and growth rate 12 has been used. All the networks are trained from scratch (weights with random initialization and bias is disabled for all the convolutional and fully connected layers) with a hypeparameter search on regularization strengths λ_1 for ℓ_1 or bounded-ℓ_1 regularizers and weight decay λ_2 on each dataset. The scaling factor γ of BN is initialized with 1.0 in case of *exponential gate* while it is initialized with 0.5 for *linear gate* as described in [24] and bias β to be zero. The hyperparameter σ in bounded-ℓ_1 regularizer is set to be 1.0 when the scheduling of this parameter is disabled. All the gating parameters g are initialized with 1.0.

We use the standard categorical cross-entropy loss and an additional penalty is added to the loss objective in the form of weight decay and sparsity induced ℓ_1 or bounded-ℓ_1 regularizers. Note that ℓ_1 and bounded-ℓ_1 regularization acts only on the gating parameters g whereas weight decay regularizes all the network parameters including the gating parameters g. We reimplemented the technique proposed in [24] which impose ℓ_1 regularization on scaling factor γ of Batch Normalization layers to induce channel level sparsity. We refer this method as ℓ_1 *on linear gate* and compare it against our methods *bounded-*ℓ_1 *on linear gate*, ℓ_1 *on exponential gate* and *bounded-*ℓ_1 *on exponential gate*. We train ResNet-164, DenseNet-40 and ResNet-50 for 240, 130 and 100 epochs respectively. Furthermore, learning rate of ResNet-164, DenseNet-40 and ResNet-50 is dropped by a factor of 10 after $(30, 60, 90)$, $(120, 200, 220)$, $(100, 110, 120)$ epochs. The networks are trained with batch size 128 using the SGD optimizer with initial learning rate 0.1 and momentum 0.9 unless specified. Below, we present the training details of each architecture individually.

LeNet5-Caffe: Since this architecture does not contain Batch Normalization layers, we do not compare our results with the method ℓ_1 *on linear gate*. We train the network with *exponential gating layers* that are added after every convolution/fully connected layer except the output layer and apply different regularizers like ℓ_1, bounded-ℓ_1 and weight decay separately to evaluate their pruning results. We set the weight decay to zero when training with ℓ_1 or bounded-ℓ_1 regularizers. The network is trained for 200 epochs with the weight decay and 60 epochs in case of other regularizers.

[1] We changed the average pooling kernel size from 7×7 to 4×4 and the stride from 2 to 1 in the first convolutional layer and also in the second block of bottleneck structure of the network.

ResNet-50: We train the network with *exponential gating layers* that are added after every convolutional layer on ImageNet dataset. We evaluate performance of the network on different values of regularization strength λ_1 like 10^{-5}, 5×10^{-5} and 10^{-4}. The weight decay λ_2 is enabled for all the settings of λ_1 and set to be 10^{-4}. We analyzed the influence of *exponential gate* and compared against the existing methods.

ResNet-164: We use a dropout rate of 0.1 after the first Batch Normalization layer in every Bottleneck structure. Here, every convolutional layer in the network is followed by an *exponential gating layer*.

DenseNet-40: We use a dropout of 0.05 after every convolutional layer in the Dense block. Here, the *exponential gating layer* is added after every convolutional layer in the network except the first convolutional layer.

MobileNetV2: On CIFAR100, we train the network for 240 epochs where learning rate drops by 0.1 at 200 and 220 epochs. A dropout of 0.3 is applied after the global average pooling layer. On ImageNet, we train this network for 100 epochs which is in contrast to the standard training of 400 epochs. We start with learning rate 0.045 and reduced it by 0.1 at 30, 60 and 90 epochs. We evaluate performance of the network on *exponential gate* over the *linear gate* with ℓ_1 regularizer and also tested the significance of bounded-ℓ_1 on *linear gate*. *Exponential gating layer* is added after every standard convolutional/depthwise separable convolutional layer in the network.

On CIFAR100, we investigate the influence of weight decay, ℓ_1 and bounded-ℓ_1 regularizers, the role of *linear* and *exponential gates* on every architecture. We also study the influence of scheduling σ in both ℓ_1 and bounded-ℓ_1 regularizers on this dataset. For MobileNetV2, we initialize σ with 2.0 and decay it at a rate of 0.99 after every epoch. In case of ResNet-164 and DenseNet-40, we initialize the hyperparameters λ_1 and λ_2 with 10^{-4} and 5×10^{-4} respectively and σ with 2.0. We increase the λ_1 to 5×10^{-4} after 120 epochs and σ drops by 0.02 after every epoch until the value of σ reaches to 0.2 and later decays at a rate of 0.99.