Learning Sparse Neural Networks Through Mixture-Distributed Regularization

Chang-Ti Huang[1], Jun-Cheng Chen[2], Ja-Ling Wu[1]
1. National Taiwan University, Taipei, Taiwan
2. Academia Sinica, Taipei, Taiwan
cthuang@cmlab.csie.ntu.edu.tw,pullpull@citi.sinica.edu.tw,wjl@cmlab.csie.ntu.edu.tw

Abstract

L_0-norm regularization is one of the most efficient approaches to learn a sparse neural network. Due to its discrete nature, differentiable and approximate regularizations based on the concrete distribution [31] or its variants are proposed as alternatives; however, the concrete relaxation suffers from high-variance gradient estimates and is limited to its own concrete distribution. To address these issues, in this paper, we propose a more general framework for relaxing binary gates through mixture distributions. With the proposed method, any mixture pair of distributions converging to $\delta(0)$ and $\delta(1)$ can be applied to construct smoothed binary gates. We further introduce a reparameterization method for the smoothed binary gates drawn from mixture distributions to enable efficient gradient gradient-based optimization under the proposed deep learning algorithm. Extensive experiments are conducted, and the results show that the proposed approach achieves better performance in terms of pruned architectures, structured sparsity and the reduced number of floating point operations (FLOPs) as compared with other state-of-the-art sparsity-inducing methods.

1. Introduction

In the post-ImageNet era, the most advanced technology in computer vision and deep learning uses large datasets along with very deep neural network (DNN) architectures. However, modern DNN architectures, e.g. VGG and ResNet, are hard to be executed efficiently, especially when using on-chip devices. In addition, "big" models usually have tons of redundant parameters and weights, tend to overfit the training data, leading to poor generalization. To address the issues, an effective solution, model compression, aims to reduce the amount of parameters while keeping the performance loss within acceptable limits. The common model compression approaches include (1) network quantization [7, 48, 43], (2) light-weight architecture design [20, 21] and neural architecture search [19], (3) knowledge distillation [17, 37], and (4) network pruning [49, 29, 33].

However, most of the aforementioned approaches still require pre-trained models to perform model compression, and thus introduce additional energy and memory consumption. To tackle with this problem, a promising way to realize a jointly trained-and-compressed model simultaneously is to enforce sparsity constraints on parameters as regularization during training. By doing so, it yields a more compact model, speeding up training and inference. The L_2 (weight decay) and L_1 (LASSO) [46] norms are common constraints of regularization; however, they are prone to shrinking large-value signals. Whereas, L_0 norm is defined as the number of non-zero elements in model parameters such that its penalty is a constant for non-zero ones. Although the L_0 norm is an effective regularizer in sparsification task, its discrete nature makes it a non-differentiable term, which has been shown to be an intractable problem in efficient gradient-based optimization. Discrete nature of the L_0 norm regularization gives the objective zero gradients w.r.t. its parameters. Quite a lot of recent works are dedicated to provide estimations for tackling this challenge.

One alternative approach [31] for relaxing discrete latent variables is to emulate binary gates through the Concrete relaxation [32]. A parameter is present when the output of a relaxed binary gate is "1", otherwise it is pruned. The number of the presented parameters will be estimated using the cumulative distribution function (CDF) as the penalty in the L_0 norm regularization. The method is novel, however the Concrete relaxation suffers from high-variance estimates of gradients. Moreover, the estimator is limited only to its own Concrete distribution and does not have the ability to generalize to other multivariate distributions.

In this paper, we propose a more general framework of network reduction called Mixture-Distributed Regularization (MDR). MDR is no longer limited to one specific distribution, but can be generalized to any mixture of distributions converging to $\delta(0)$ and $\delta(1)$ while still allowing for reparameterization. This way we achieve structured group sparsity and acceleration on modern neural network (NN) architectures. By comparing variance estimates of gradients between the mixture distribution and the Concrete relaxation,

it follows that the mixture distribution gives lower gradient variance estimates than that of the Concrete relaxation. In general, it is worth noting that for the proposed approach, except small amount of additional memory and training costs introduced by binary gates, the other part of training is the same as the original model since the proposed approach performs joint model training and neuron pruning in a single shot end-to-end fashion without any retraining while keeping similar performance level as the original model. We demonstrate competitive classification and compression results on various datasets.

2. Related Work

Due to a large amount of related works in the literature, we focus on network pruning for its similar nature to our framework. Pruning is one of the earliest methods [28] for designing an efficient NN, which is proposed to reduce the network complexity and overfitting, and to improve generalization. Pruning can be further subdivided into weight pruning and neuron pruning in terms of pruned objects. Weight pruning has been shown that a large amount of model parameters (up to 99%) can be pruned in common architectures [13, 12, 11, 33, 3, 6]. However, weight pruning is in general inefficient since the dimensionality of the weight matrix is not taken into consideration. One must rely on additional coding schemes (e.g. CSC or CSR format) to access the sparse weight matrix, so that weight pruning can truly save computations. Whereas for neuron pruning, the input/output weights of a pruned neuron are also pruned, leading to computation savings. Therefore, there are subsequent results using empirical Bayesian reasoning that take the computational structure of NNs into account [35, 30]. Unfortunately, most of the methods need to train a dense network first, then prune redundant weights/neurons from the pre-trained model. Alvarez *et al.* [1, 2] propose to use group sparsity and low-rank regularizers to perform joint learning and model compression during training. Similarly, our method which inherits the advantages of [31] can also directly train a sparse NN model from scratch, thus allowing conditional computation for accelerating training [5, 4]. In addition, our approach can induce structured sparsity layer-by-layer, i.e., we can use different λ values for different $\mathrm{Dropout}$-like layers so that each layer is trained separately with various sparsity. This will tend to have fewer parameters in the layer with the choice of relatively large λ value, with which the aforementioned methods are not competitive.

Relaxing the L_0 norm into a deterministic and differentiable function can be done via gradient estimators. While the REINFORCE [47] and control variates [10, 42, 8] are unbiased gradient estimators, the former suffers from high-variance estimates and the latter require a surrogate function which extends quite a lot of calculations. The reparameterization trick [24, 39] is another way to calculate gradients without encountering any stochastic node during backpropagation; however, it is originally applicable to training variational auto-encoders (VAEs), not to the discrete nature of the L_0 norm. An alternative approach that smooths discrete latent variables to continuous ones is the Concrete relaxation [32, 22]. Specifically, [31] applied the binary Concrete [32], and introduced stretching then hard-sigmoid, to sparsify NNs. In fact, several distributions have been previously done with Bernoulli-like distributions. Under the spike-and-exp distribution [40], the mixing density of the scales converges to a Bernoulli distribution. [30] proposed Bayesian compression with mixtures of normal distributions, while [35] injected noise to neurons via log-normal approximations. [45, 44] introduced overlapping transformations for training discrete VAEs. The reparameterization method for the mixture distribution is first described in [9] and derived as a binary special case in [44]. However, [9] applied to multivariate distributions over mixture weights, and [44] performed stochastic variational inference with mixtures of binary distributions. To the best of our knowledge, we are the first to introduce this reparameterization method for compressing NN models.

3. Relaxing L_0 norms through mixture distributions

In this Section, we briefly describe the problem formulation for L_0-norm regularization and required preliminary knowledge for the mixture distribution. To begin with, we'll consider a general optimization scenario. Let $(\boldsymbol{x}, \boldsymbol{y})$ be input/output pairs of a dataset parameterized by $\boldsymbol{\theta}$, $f(\cdot)$ be a DNN and $\mathcal{C}(\cdot)$ be the loss function criterion. We wish to optimize

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\mathcal{C}\left(f_\theta\left(\boldsymbol{x}\right), \boldsymbol{y}\right)\right]. \tag{1}$$

3.1. Formulation

A way to compress the model is using the L_0 norm regularization of parameters. Let $\|\boldsymbol{\theta}\|_0$ be the L_0 penalty defined as the number of non-zero elements in model parameters. We wish to minimize

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\mathcal{C}\left(f_\theta\left(\boldsymbol{x}\right), \boldsymbol{y}\right)\right] + \lambda\|\boldsymbol{\theta}\|_0, \tag{2}$$

where λ corresponds to a loss weight of the L_0 norm regularization. The reason for choosing the L_0 norm is that it penalizes non-zero elements fairly, i.e., the penalty is a constant. In contrast, the common L_2 and L_1 norms are prone to shrinking large-value parameters $\boldsymbol{\theta}$, enforcing more penalties on elements farther away from 0. In the sparsification tasks, parameters approaching to zero are desirable. For any non-zero element, we properly give the same penalty, which matches the definition of the L_0 norm. By enforcing

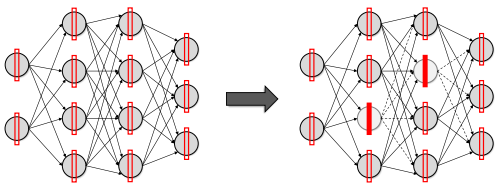


Figure 1: Visualization of binary gates. A neuron is present when the binary gate is on (hollow red). Once the binary gate turns off (solid red), the corresponding neuron (fade node) will be dismissed and its output weights (dashed line) will be pruned. Note that the framework is conceptually similar to Dropout-like layers as in [30, 35, 31] which inject noise to input neurons for fully-connected layers and to output feature maps for convolutional layers, achieving structured group sparsity.

sparsity constraints on parameters, the regularizer penalizes the number of non-zero elements and finally induces sparsity. However, the discrete nature of the L_0 norm makes it an intractable term. Fortunately, a previous work provided a recipe for approximating L_0 norms [31].

Let $\boldsymbol{z}_i$ be a Bernoulli random variables and q_i denote the probability of sampling 1, i.e. $q_i \equiv q(z_i = 1|\theta_i)$. The L_0 norm can be relaxed by reparameterizing model parameters $\boldsymbol{\theta}$ as:

$$\theta_i = \hat{\theta}_i z_i, \qquad \|\boldsymbol{\theta}\|_0 = \sum_{i=1}^{|\theta|} q_i, \tag{3}$$

where z_i is considered as a binary gate. $\hat{\theta}_i$ is present when the output of z_i is "1", otherwise it is pruned. We visualize neurons and their corresponding binary gates in Figure 1. The number of presented parameters can thus be approximated by the Bernoulli probability of sampling 1 as the penalty at the second term on the right-hand side of Eq. (2). Unfortunately, the approximator in Eq. (3) again doesn't allow for gradient-based optimization due to the discontinuous nature of binary states z_i.

3.2. Mixture distributions

In [31], discrete gates are relaxed into continuous representations using the binary Concrete [32, 22], which allows for using automatic differentiation to calculate gradients of the objective. However, the framework introduced in [31] has its limitations. Firstly, it applies only to a specific binary Concrete distribution, and does not have any flexibility to scale to other multivariate distributions. In addition, the binary Concrete contains two hyper-parameters (i.e. location and temperature), implying multiple evaluations required during fine-tuning. Owing to this reason, we introduce a more general approach for smoothed binary gates, ζ's. The proposed method provides additional flexibility through a variety of mixtures of distribution families. Before jumping into the details of the proposed mixture-distributed regularization in Section 4, we first introduce the mixture distributions and several frequently used ones as follows.

A mixture distribution is the probability distribution derived from the individual distributions. Let $r(\zeta|z)$ be the PDF and $R(\zeta|z)$ be the CDF of a given distribution. We obtain $r(\zeta|z) = \prod_i r(\zeta_i|z_i)$ through transforming each discrete z_i into its continuous analogue ζ_i. The individual distributions called mixture components are combined to form a mixture distribution $Q(\zeta) = \sum_{z_i} q(z_i) r(\zeta|z_i)$, where $q(z_i)$ associated with each component z_i is called mixture weight. Any mixture pair of distributions converging to $\delta(\zeta = 0)$ and $\delta(\zeta = 1)$ inherits similar properties in terms of the binary Concrete, which can be generalized to be the relaxed binary gate ζ. With this inspiration, we can use a variety of distribution families (e.g. Gaussian, Beta and Exponential) to approximate ζ. Here, we demonstrate our approach using several practical mixture distributions.

Exponential mixture We first apply the mixture of two exponential distributions, of which the PDFs and CDFs are defined as

$$r_{\text{Exp}}(\zeta|z) = \begin{cases} \dfrac{\beta e^{-\beta\zeta}}{1-e^{-\beta}} & \text{if } z = 0 \\ \dfrac{\beta e^{\beta(\zeta-1)}}{1-e^{-\beta}} & \text{if } z = 1 \end{cases}, \tag{4}$$

$$R_{\text{Exp}}(\zeta|z) = \begin{cases} \dfrac{1-e^{-\beta\zeta}}{1-e^{-\beta}} & \text{if } z = 0 \\ \dfrac{e^{\beta(\zeta-1)} - e^{-\beta}}{1-e^{-\beta}} & \text{if } z = 1 \end{cases}, \tag{5}$$

where $\zeta \in [0, 1]$ and β is temperature. Defining the Bernoulli probability of sampling 1 as $q \equiv q(z = 1|\theta)$, the overall CDF for the mixture of exponential distributions can be derived by

$$\begin{aligned} \text{CDF}^{\text{mix}}_{\text{Exp}}(\zeta) &= (1-q)R_{\text{Exp}}(\zeta|z=0) + qR_{\text{Exp}}(\zeta|z=1) \\ &= \frac{1-q}{1-e^{-\beta}}[1-e^{-\beta\zeta}] + \frac{q}{1-e^{-\beta}}[e^{\beta(\zeta-1)} - e^{-\beta}]. \end{aligned} \tag{6}$$

As β increases, the gradient of ζ drawn from $r_{\text{Exp}}(\zeta|z)$ grows exponentially at $\zeta = 0.5$, and converges to ∞ as $\beta \to \infty$ (refer to Appendix A). It turns out that the exponential mixture suffers from higher gradient variance estimates, which are not conducive to MCMC sampling. Fortunately, there are some alternatives to deal with the high-variance problem.

Exponential-Uniform mixture The gradient of ζ can be forced finite by combining the exponential distribution with a uniform distribution. The PDF and CDF of the exponential-

uniform distribution can be written as

$$r_{\text{ExpUni}}(\zeta|z) = (1-\epsilon)r_{\text{Exp}}(\zeta|z) + \epsilon,$$
$$R_{\text{ExpUni}}(\zeta|z) = (1-\epsilon)R_{\text{Exp}}(\zeta|z) + \epsilon\zeta, \tag{7}$$

where ϵ is the probability density of the uniform distribution.

Power-Law function mixture We can also obtain lower gradient variances of ζ by replacing exponential distributions with heavy-tailed power-law functions. The PDFs and CDFs of power-law function distributions are given by

$$r_{\text{Power}}(\zeta|z) = \begin{cases} \frac{1}{\beta}\zeta^{\frac{1}{\beta}-1} & \text{if } z=0 \\ \frac{1}{\beta}(1-\zeta)^{\frac{1}{\beta}-1} & \text{if } z=1 \end{cases},$$
$$R_{\text{Power}}(\zeta|z) = \begin{cases} \zeta^{\frac{1}{\beta}} & \text{if } z=0 \\ 1-(1-\zeta)^{\frac{1}{\beta}} & \text{if } z=1 \end{cases}, \tag{8}$$

where $\beta > 1$ ensures heavier tails with finite area. As shown in Appendix A.1 in the supplementary materials, we qualitatively report the differences among exponential, exponential-uniform and power-law functions from the perspectives of the PDF, the inverse CDF, and the gradient of the inverse CDF. Further, we quantitatively compare the gradient variance estimates of the inverse CDF in Appendix A.2.

Now, we are ready to construct the proposed method for NN model compression using the mixture distributions described in the above.

4. The Mixture-Distributed Regularization (MDR)

To perform mixture-distributed regularization-based gradient optimization, we firstly construct our MDR regularizer, and then we provide the reparameterization method for optimizing emulated binary gates of the MDR.

4.1. Constructing the MDR

Let $\zeta_i \in \mathcal{R}$ be the relaxed, stretched and clamped smoothed binary gate under a given mixture distribution parameterized by ϕ_i, where following [31], drawn ζ_i is *stretched* to the (a, b) interval, and *clamped* into $[0, 1]$. We can reformulate Eq. (3) as

$$\theta_i = \hat{\theta}_i\zeta_i, \ \|\boldsymbol{\theta}\|_0 = \sum_{i=1}^{|\theta|}\left(1 - \text{CDF}^{\text{mix}}(\zeta_i \le 0|\phi_i)\right), \tag{9}$$

where we approximate the L_0 norm regularization through penalizing parameters greater than zero using the cumulative density. The parameters ϕ_i of CDF^{mix} could refer to the mixture weight q_i for the mixture distribution and the hyperparameters for each parametric distribution. In this paper, we select these hyperparameters through cross validation and ϕ_i mainly refers to the mixture weight q_i. Note that under $a < 0$ and $b > 1$, we can optimize the cumulative density within $(-\infty, \zeta_i]$ and $[\zeta_i, \infty)$. In addition, stretched ζ_i including $\{0, 1\}$ in its support gives us additional benefits that it can truly prune parameters as ζ_i approaches to zero. Applying the proposed MDR to the objective function in Eq. (2), we obtain

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \underbrace{\mathbb{E}_{p(\boldsymbol{x}|\boldsymbol{\theta})}\left[\mathcal{C}\left(f(\boldsymbol{x}; \boldsymbol{\theta} \odot \boldsymbol{\zeta}), \boldsymbol{y}\right)\right]}_{\mathcal{L}_E} + \underbrace{\lambda\sum_{i=1}^{|\theta|}\left(1 - \text{CDF}^{\text{mix}}(\zeta_i \le 0|\phi_i)\right)}_{\mathcal{L}_R}, \tag{10}$$

where $\odot$ performs the element-wise product, $\mathcal{L}_E$ corresponds to the error loss and $\mathcal{L}_R$ refers to the MDR loss. Minimizing $\mathcal{L}_E$ implies that the prediction is fitting to the ground truth well. Whereas, minimizing $\mathcal{L}_R$ penalizes the non-zero weights, inducing the model sparsity. By handling a balanced loss weight λ, we can train a compact model to get reasonable performance from scratch.

One thing left behind is how to represent the procedure of sampling ζ as a deterministic and differentiable function which allows for backpropagation. In the following, we develop the reparameterization method for optimizing the MDR.

4.2. Optimizing the MDR

In Eq. (10), $\boldsymbol{\zeta}$ are smoothed binary gates generated from a mixture distribution, where the sampling procedure is stochastically intractable. To optimize the objective, [32] used a differentiable mapping called inverse transform sampling. Given the inverse CDF of the binary Concrete, inverse transform sampling maps samples $u \sim \text{Uniform}(0, 1)$ to samples ζ from the PDF of the binary Concrete. The only condition for the transformation is that the inverse CDF of a given distribution must be solved *analytically*. However, not all of the distributions can compute the inverse CDF analytically. For the power-law function mixture, we wish to reformulate $\text{CDF}^{\text{mix}}_{\text{Power}}(\zeta) - u = 0$ as a quadratic equation. However, the term $(1-\zeta)^{\frac{1}{\beta}}$ in Eq. (8) is hard to form a quadratic equation. Apparently, inverse transform sampling does not work in certain cases. To tackle with the intractable problem, we develop the reparameterization method in the following.

4.2.1 Reparameterization for the MDR

For the mixture distribution that can not sample ζ through inverse transform sampling, we provide a reparameterization method. Reparameterization for the MDR is a special case derived from [9, 44], where [9] applied the reparameterization trick to multivariate distributions over mixture weights, and [44] performed stochastic variational inference with mixtures of binary distributions to train discrete VAEs.

Let $r(\zeta|z)$ be the PDF of the mixture component and q be the mixture weight where $q \equiv q(z=1|\theta)$. Applying the definition of a univariate CDF to the components $z = \{0, 1\}$ yields

$$\begin{aligned} \text{CDF}^{\text{mix}}(\zeta) \quad = \quad & \underbrace{(1-q)\int_{t=-\infty}^{\zeta} r(t|z=0)dt}_{\text{0-component}} \\ & + \underbrace{q\int_{t=-\infty}^{\zeta} r(t|z=1)dt}_{\text{1-component}} = u, \end{aligned} \tag{11}$$

where $u \sim \text{Uniform}(0, 1)$. Here, we take the gradient w.r.t. q from both sides of Eq. (11) giving $\partial \text{CDF}^{\text{mix}}(\zeta)/\partial q = \partial u/\partial q$. After some derivations and rearrangements (see Appendix B), the gradient w.r.t. q can be obtained as

$$\frac{\partial \zeta}{\partial q} = \frac{R\left(\zeta|z=0\right) - R\left(\zeta|z=1\right)}{(1-q)r\left(\zeta|z=0\right) + qr\left(\zeta|z=1\right)}. \tag{12}$$

As shown in Eq. (12), given the PDF and the CDF of the mixture components, we can take gradients of ζ w.r.t. q. Reparameterization for the MDR is highly required when the inverse CDF of the mixture distribution cannot be solved analytically. In that case, we cannot sample ζ through inverse transform sampling, instead we optimize the MDR through backpropagating $\partial\zeta/\partial q$.

4.2.2 Estimating ζ^* for testing

At test time, we provide an estimator for sampling a deterministic ζ^*. Let ζ^* be the smoothed binary gate after stretching and clamping, as we mentioned in Eq. (9). The final weights passing through the ζ^* are given by

$$\begin{aligned} \theta^* = \hat{\theta} \odot \zeta^*, \quad \zeta^* = \min(1, \max(0, \bar{\zeta})), \\ \bar{\zeta} = \zeta(b-a) + a. \end{aligned} \tag{13}$$

It is worth mentioning that the proposed estimator is more intuitive than the one given in [31], of which hard-concrete gate ζ at each layer of the networks is estimated by location $\log \alpha$, where $\log \alpha$ is initialized by sampling from a normal distribution with a mean that yields $\alpha/(\alpha+1)$ approximating to the Dropout. Emulating ζ through $\alpha/(\alpha+1)$ is somewhat counter-intuitive and error-prone. In contrast, we estimate ζ^* through the mixture weight q generated from a normal distribution with a mean $\mu = 1 - \text{Dropout}$. Hence, as the Dropout rate increases (i.e. prone to zero elements), we force the mixture weight q to be decreased (i.e. increasing the proportion of 0-component). It is intuitive yet convenient for estimating ζ^* at test time.

Algorithm 1 Optimizing parameters and the MDR simultaneously.

Require: parameters θ, mixture weights ϕ, reparameterized sampler $\zeta \sim Z(\zeta|\phi)$, step sizes α_1, α_2.
while not converged **do**
  $\zeta_i \sim Z(\zeta_i|\phi)$ ▷ Sample ζ
  $\hat{g}_\theta \leftarrow \nabla_\theta \mathcal{L}_E(\zeta_i|\theta) + \nabla_\theta \mathcal{L}_2(\theta, \phi)$ ▷ Estimate gradient of θ
  $\hat{g}_\phi \leftarrow \nabla_\phi \mathcal{L}_E(\zeta_i|\theta) + \nabla_\phi \mathcal{L}_R(\phi) + \nabla_\phi \mathcal{L}_2(\theta, \phi)$ ▷ Estimate gradient of ϕ
  $\theta \leftarrow \theta - \alpha_1 \hat{g}_\theta$ ▷ Update parameters
  $\phi \leftarrow \phi - \alpha_2 \hat{g}_\phi$ ▷ Update mixture weights
end while
return θ

In addition to the L_0 norm, the objective in [31] is also combined with the L_2 norm as the regularizer to obtain additional prior assumptions on $\boldsymbol{\theta}$. For a fair comparison, we also combine our MDR with the L_2 norm (weight decay) regularization. Besides, following [31] we regularize neurons/features using group sparsity constraints, which can be practical for reducing the computational complexity. To sum up, we provide the optimization algorithm for the proposed method in Algorithm 1. Note that the reparameterized sampler $Z(\zeta|\phi)$ refers to Eq. (12); $\mathcal{L}_E$ and $\mathcal{L}_R$ are defined as in Eq. (10), and $\mathcal{L}_2$ is the L_2 regularization.

5. Implementation details

We implement our methods in PyTorch [36] with a single graphics card NVIDIA GeForce GTX 1080 Ti, and optimize the neural network architectures using Adam [23] with the default hyperparameters for LeNet-300-100/LeNet-5-Caffe and SGD optimizer for WRN. In addition, we also perform optimization on our model parameters using the exponential moving average. Other implementation details are described as follows.

Parameters: One of the learned parameters of our models is the mixture weight q. q is initialized by sampling from a normal distribution with a mean $\mu = 1 - \text{Dropout}$ and a standard deviation 0.01. The others learned parameters are weights and biases. We initialize weight parameters using He normal initializer [14] with fan-out mode which preserves the magnitudes in the backwards pass, whereas biases (if

use) are initialized by filling zeros. Note that we clamp q into the range $[0, 1]$ over layers over minibatches, since the probability of the mixture component should not go beyond this range. If we do not impose the above constraint on q, the model will not converge in general.

On the other hand, for other hyperparameters, the stretching factors a, b and the temperature β are hyper-parameters of our models. We set $a = -0.1$, $b = 1.1$ for all of our experiments. As for the temperature which controls the shape of the MDR, we fix β throughout training. Instead, we optimize the mixture weight q which dominates the proportion of 0- and 1- components. Since β acts as a shape controller, different MDRs also match to different β values. To decide the proper β, we perform cross validation per MDR per architecture. That is, we validate the model compression and classification performances under different β values. For both LeNet-300-100 and LeNet-5-Caffe, we set $\beta = \{30, 25, 40\}$ for MDR-Exp, MDR-ExpUni and MDR-PowerLaw respectively. For Wide-ResNet, we respectively set $\beta = \{6, 8, 2\}$ for MDR-Exp, MDR-ExpUni, and MDR-PowerLaw on both CIFAR-10/CIFAR-100. For LeNet-300-100, we set weighting factor $\lambda = \{0.01, 0.02, 0.2\}$ for the layers in both MDR-Exp and MDR-ExpUni, and $\lambda = \{0.05, 0.2, 3\}$ for the layers in MDR-PowerLaw. For LeNet-5-Caffe, we set $\lambda = \{6, 0.3, 0.05, 5\}$ for MDR-Exp, $\lambda = \{5, 0.25, 0.01, 1\}$ for MDR-ExpUni and $\lambda = \{5, 0.25, 0.05, 5\}$ for MDR-PowerLaw. For Wide-ResNet, we set $\lambda = \{0.002, 0.002, 0.004\}$ for MDR-Exp, MDR-ExpUni, and MDR-PowerLaw on CIFAR-10; and set $\lambda = \{0.002, 0.001, 0.003\}$ for MDR-Exp, MDR-ExpUni, and MDR-PowerLaw on CIFAR-100.

Reparameterization: we defined the gradient of ζ w.r.t. q as in Eq. (12) that given the PDF and the CDF of the 0- and 1- components, we can calculate the term $\partial\zeta/\partial q$. Here, we present in detail the algorithm for reparameterizing ζ in Algorithm 2. Note that during backward pass [41], reverse-mode automatic differentiation (AD) [38] does not allow for backpropagating the gradients through the samples generated from a stochastic sampler (see sample noise in Algorithm 2). Fortunately, we can use detach commands in PyTorch that preclude the samples from a computational graph, saving additional AD operations.

Data preprocessing: For MNIST, we flat images into vectors with 784 dimensions, employ 4 subprocesses to use for data loading, and shuffle both training and validation sets with minibatch size setting to 100. As for CIFAR-10 and CIFAR-100, we use data argumentation as proposed in [50] for the purpose of a fair comparison. We perform mean/std normalization, horizontal flips, random cropping 32×32 from images padded by 4 pixels on each side (i.e. operator randomly crops a 32×32 bounding box from a 40×40 image), filling the additional pixels with reflections w.r.t. the original images. Also, we employ 4 subprocesses to load shuffled training and validation sets with minibatch size setting to 128.

Algorithm 2 The reparameterized sampler of ζ.

Require: mixture distribution $r(\zeta)$ and $R(\zeta)$, mixture weight $q \equiv q(z = 1|\phi)$
$u \sim p(u)$ ▷ Sample noise
$\zeta \sim q(\zeta|\phi, u)$ ▷ Sample ζ
$r(\zeta|z = 0), r(\zeta|z = 1)$ ▷ Compute PDFs of sampled ζ
$R(\zeta|z = 0), R(\zeta|z = 1)$ ▷ Compute CDFs of sampled ζ
$\frac{\partial\zeta}{\partial q} \leftarrow \frac{R(\zeta|z=0) - R(\zeta|z=1)}{(1-q)r(\zeta|z=0) + qr(\zeta|z=1)}$ ▷ Estimate gradient w.r.t. q
$g_\phi \leftarrow \frac{\partial\zeta}{\partial q} \cdot q$ ▷ Estimate gradient of ζ
$\hat{g_\phi} \leftarrow g_\phi - g_\phi$ ▷ Dummy gradient
$\zeta \leftarrow \zeta + \hat{g_\phi}$ ▷ Reparameterize ζ
return ζ

Training Strategy: The parameters are optimized using Adam [23] for LeNet-300-100/LeNet-5-Caffe, SGD optimizer for WRN respectively, and the exponential moving average (EMA) for all models. For Adam, we set the hyperparameters default with learning rate 0.001 using cross-entropy loss, trained for total 200 epochs in both architectures. As for SGD, we apply the learning rate schedule proposed in [50] that initially set to 0.1, dropped by 0.2 at 60, 120 and 160 epochs, trained for total 200 epochs. Besides, weight decay is set to 0.0005, momentum to 0.9, using cross-entropy loss. As for EMA, we set the weighting factor α to 0.99 s.t. $\mathsf{EMA}_t = \alpha \times \mathsf{EMA}_{t-1} + (1 - \alpha) \times \mathsf{P}_t$, where EMA_t is the value of the EMA at any time period t and P_t is the model parameter at a time period t. The exponential moving average is useful in training strategy since it smooths out short-term fluctuations of model parameters and highlights long-term trends.

As for Dropout [18], our method theoretically does not require the use of Dropout since the relaxed binary gate ζ generated from the MDR is similar to Dropout in design that drops certain parts of the model parameters, i.e. whether Dropout or the MDR can be interpreted as a (relaxed) binary gate under a given distribution converging to $\delta(\zeta = 0)$ and $\delta(\zeta = 1)$. In fact, our method is easy to implement since the MDR can be considered as separate Dropout-like layers with corresponding λ values. However for better estimations of our models, we initialize the mixture weight q to $1 -$ Dropout, i.e. $q = \{0.8, 0.5, 0.5\}$ for three layers in LeNet-300-100, $q = 0.5$ for all layers in LeNet-5-Caffe and $q = 0.7$ for the layers adopted Dropout in [50].

Network	Method	Neurons per layer	Error(%)	$\lvert\mathbf{W}_{\neq 0}\rvert(10^5)$	$\frac{\lvert\mathbf{W}_{\neq 0}\rvert}{\lvert\mathbf{W}\rvert}(\%)$	FLOPs
LeNet-300-100	Original	784-300-100	1.6	2.7	100.0	1×
	Sparse VD [33]	512-114-72	1.8	0.7	25.3	4×
	BC-GNJ [30]	278-98-13	1.8	0.3	10.8	9×
	BC-GHS [30]	311-86-14	1.8	0.3	10.6	9×
	L_0 [31]	266-88-33	1.8	0.3	10.0	10×
	ADMM-pruning [51]	-	1.6	0.1	4.4	-
	MDR-Exp	186-76-22	1.8	**0.2**	**6.0**	**17**×
	MDR-ExpUni	124-67-22	1.8	**0.1**	**3.8**	**27**×
	MDR-PowerLaw	148-64-25	1.8	**0.1**	**4.3**	**24**×
LeNet-5-Caffe	Original	20-50-800-500	0.8	4.31	100.0	1×
	Sparse VD [33]	14-19-242-131	1.0	0.40	9.3	3×
	SBP [35]	3-18-284-283	0.9	0.85	19.7	11×
	BC-GNJ [30]	8-13-88-13	1.0	0.04	1.0	8×
	BC-GHS [30]	5-10-76-16	1.0	**0.03**	**0.6**	**15**×
	L_0 [31]	9-18-65-25	1.0	0.06	1.4	6×
	ADMM-pruning [51]	-	0.8	0.06	1.4	-
	MDR-Exp	12-15-70-29	1.0	0.07	1.7	5×
	MDR-ExpUni	6-8-72-31	1.0	**0.04**	**0.9**	**14**×
	MDR-PowerLaw	10-18-57-18	1.0	0.06	1.4	5×

Table 1: Compression results on MNIST classification tasks using LeNet-300-100 and LeNet-5-Caffe. We validate methods based on the error(%), the ratio of non-zero weights $\frac{\lvert\mathbf{W}_{\neq 0}\rvert}{\lvert\mathbf{W}\rvert}(\%)$, and acceleration of expected FLOPs, where the results of ADMM-pruning [51] is attained after retraining.

Further, we follow [31] that samples the binary gate ζ once for each minibatch of training data points, i.e. training data points in a minibatch share the same ζ, and employs a gate per input neuron for fully-connected layers, and a gate per output feature map for convolutional layers, speeding up in training with the minimal or lossless performance.

6. Experiments

We validate the proposed method on the datasets of MNIST [26], CIFAR-10 and CIFAR-100 [25]. To adapt for the datasets, we apply our method to both classification and compression tasks using the well-known NN architectures: LeNet-300-100 [27], LeNet-5-Caffe[1] on MNIST, and wide residual networks (WRNs) [50] on CIFAR-10 and CIFAR-100.

6.1. LeNet-300-100 and LeNet-5-Caffe on MNIST

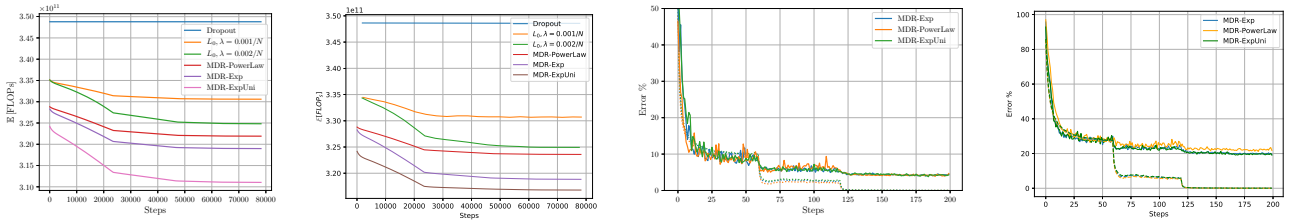
Firstly, following the settings of [31], we apply LeNet-300-100 which consists of three fully-connected (fc) layers on MNIST dataset. As we can see from Table 1[2], all of the proposed MDRs outperform the other baselines in terms of reduced neurons per layer, especially in the first fc layer, while keeping the error in a state-of-the-art level. Our methods obtain the highest acceleration of expected FLOPs[3] with the lowest ratio of non-zero weights $\frac{\lvert\mathbf{W}_{\neq 0}\rvert}{\lvert\mathbf{W}\rvert}$, which theoretically allow for speeding up training and inference time. It is worth noting that our MDRs do not need any pre-trained model; instead, we jointly train-then-sparsify NNs from scratch with decreasing FLOPs and increasing ratio of zero weights. As visualized in Appendix C, the relaxed binary gate ζ drawn from the MDR converges to $\delta(\zeta = 0)$ and $\delta(\zeta = 1)$ with extremely high mixture weight at the 0-component. This again shows that the proposed MDRs achieve structured group sparsity.

Secondly, we employ LeNet-5-Caffe on MNIST dataset. LeNet-5-Caffe consists of two convolutional layers followed by two fc layers. As we can see from Table 1, the proposed MDRs are also competitive with the baselines in terms of both classification and compression on CNN architectures. During training, we observe that the convolutional layers converge much slower than the fc layers (consider to see Appendix C). This is because CNNs perform parameter-efficient convolution with much fewer redundant

[1] https://github.com/BVLC/caffe/tree/master/examples/mnist

[2] Since we follow the implementation of Dropout-like layers as L_0 [31], we report the number of actual effective non-zero weights to the final classification for the last three columns for both methods. Theoretically, other non-zero weights which do not contribute to the final classification due to Dropout-like layers can be removed with some modification for the current implementation and with very small amount of additional computation cost.

[3] We follow the convention described in Appendix A.1 of [34] to yield an estimate of FLOPs, where we assume multiplication and addition take one FLOP respectively and $\max(x_1, ..., x_n)$ takes $n - 1$ FLOPs.



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

Figure 2: (a)(b) The expected value of FLOPs during training and (c)(d) the objective function of both training (dashed line) and validation (solid line) error on the benchmark task of CIFAR-10 and CIFAR-100 for WRNs respectively.

filters/parameters than that of fc layers. Once we enforce sparsity constraints, CNNs will in general tend to learn smoothly to get a balance between the network performance and the sparsity.

6.2. Wide-ResNet on CIFAR-10 and CIFAR-100

To demonstrate the scalability of the proposed method, we perform classification tasks to one of large modern architectures, i.e. WRNs, on CIFAR-10 and CIFAR-100 datasets. For WRNs, we apply the best module of WRN-28-10 which indicates depth to 28 and widening factor to 10. We perform the MDR regularizer on the output feature maps of the hidden layers where Dropout is employed as in [50]. In addition, we employ the L_2 regularization on the hidden layers with the weight decay coefficient reported in [50], except the layers performed the relaxed binary gates we follow [31] dividing the weight decay coefficient by 0.7.

Network	Method	CIFAR-10 (%)	CIFAR-100 (%)
ResNet-110	original [15]	6.43	25.16
	pre-act [16]	6.37	-
WRN-28-10	original [50]	4.00	21.18
	dropout [50]	3.89	18.85
	$L_0, \lambda = 0.001/N$ [31]	3.83	18.75
	$L_0, \lambda = 0.002/N$ [31]	3.93	19.04
	MDR-Exp	3.98	19.10
	MDR-ExpUni	**3.80**	19.07
	MDR-PowerLaw	3.95	19.09

Table 2: Classification results on the benchmark tasks of CIFAR-10 and CIFAR-100 using ResNet-110 and WRN-28-10. The baseline results are taken from [31, 16] with performance measure of error %.

Similarly, following the settings of [31], we can see from Table 2 and Figure 2(b) that the proposed MDR is competitive with ResNets [15, 16], WRNs [50], and other compression techniques [31]. For sparsification, we compare the proposed MDRs with Dropout network [50] and $\lambda \in \{0.001, 0.002\}$ settings in [31]. As we can observe in Figure 2(a), all of our methods especially MDR-ExpUni require much fewer numbers of FLOPs throughout training since the proposed MDRs give us additional flexibility to apply the mixture distributions with relatively higher proportion at the 0-component, which indicates more binary gates turning off (and thus more FLOPs savings). Furthermore, we obtain considerable acceleration in terms of FLOPs reduction, against the mehods in [31]. As we can see from Figure 2(a), our MDR-ExpUni retains $\mathbf{4}\times$ FLOPs reduction than the best setting of $L_0, \lambda = 0.001/N$ in [31] (i.e., 4% FLOPs reduction of ours versus 1% of [31]), without any accuracy drop. Besides, since the original WRN-28-10 model requires around $5.25e^{11}$ FLOPs per inference and our best setting, MDR-ExpUni, only needs around $3.12e^{11}$ FLOPs, our method reduces 41% computation as compared with the original WRN-28-10.

7. Conclusions

We proposed Mixture-Distributed Regularization (MDR), a more general framework to release the combinatorial complexity induced by the L_0 norm. With our method, any mixture pair of distributions converging to $\delta(\zeta = 0)$ and $\delta(\zeta = 1)$ has the ability to emulate the binary gate, while still allowing for gradient-based optimization through the reparameterization method. As a result, efficient neural networks can be constructed by employing the expected L_0 regularization without encountering any intractable term. The experimental results on the benchmark tasks show that the proposed method can achieve better performance than other sparisity-inducing methods for learning sparse neural networks in most cases in terms of reduced FLOPs or pruned architectures.

8. Acknowledgement

This work is partially supported by Ministry of Science and Technology, Taiwan, R.O.C, under Grants no. MOST 108-2218-E-001-004-MY2 and MOST 106-3114-E-002-009.

References

[1] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016. 2

[2] Jose M Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pages 856–867, 2017. 2

[3] Amir H Ashouri, Tarek S Abdelrahman, and Alwyn Dos Remedios. Retraining-free methods for fast on-the-fly pruning of convolutional neural networks. *Neurocomputing*, 370:56–69, 2019. 2

[4] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015. 2

[5] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 2

[6] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020. 2

[7] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016. 1

[8] Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. In *International Conference on Learning Representations (ICLR)*, 2018. 2

[9] Alex Graves. Stochastic backpropagation through mixture density distributions. *arXiv preprint arXiv:1607.05690*, 2016. 2, 5

[10] Shixiang Gu, Sergey Levine, Ilya Sutskever, and Andriy Mnih. Muprop: Unbiased backpropagation for stochastic neural networks. *arXiv preprint arXiv:1511.05176*, 2015. 2

[11] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016. 2

[12] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016. 2

[13] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015. 2

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 5

[15] 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*, pages 770–778, 2016. 8

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016. 8

[17] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015. 1

[18] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 6

[19] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. *arXiv preprint arXiv:1905.02244*, 2019. 1

[20] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1

[21] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 1

[22] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 2, 3

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015. 5, 6

[24] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. *International Conference on Learning Representations (ICLR)*, 2014. 2

[25] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 7

[26] Yann LeCun. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998. 7

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 7

[28] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990. 2

[29] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations (ICLR)*, 2019. 1

[30] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3288–3298, 2017. 2, 3, 7

[31] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through L_0 regularization. In *International Conference on Learning Representations (ICLR)*, 2018. 1, 2, 3, 4, 5, 7, 8

[32] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016. 1, 2, 3, 4

[33] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017. 1, 2, 7

[34] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations (ICLR)*, 2017. 7

[35] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems*, pages 6775–6784, 2017. 2, 3, 7

[36] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 5

[37] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. In *International Conference on Learning Representations (ICLR)*, 2018. 1

[38] Louis B Rall. Automatic differentiation: Techniques and applications. 1981. 6

[39] Danilo J Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1278–1286, 2014. 2

[40] Jason Tyler Rolfe. Discrete variational autoencoders. *International Conference on Learning Representations (ICLR)*, 2017. 2

[41] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. 6

[42] George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *Advances in Neural Information Processing Systems*, pages 2627–2636, 2017. 2

[43] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. In *International Conference on Learning Representations (ICLR)*, 2017. 1

[44] Arash Vahdat, Evgeny Andriyash, and William G Macready. Dvae#: Discrete variational autoencoders with relaxed Boltzmann priors. In *Advances in Neural Information Processing Systems*, 2018. 2, 5

[45] Arash Vahdat, William Macready, Zhengbing Bian, Amir Khoshaman, and Evgeny Andriyash. Dvae++: Discrete variational autoencoders with overlapping transformations. In *International Conference on Machine Learning*, pages 5042–5051, 2018. 2

[46] 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*, pages 2074–2082, 2016. 1

[47] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 2

[48] Junru Wu, Yue Wang, Zhenyu Wu, Zhangyang Wang, Ashok Veeraraghavan, and Yingyan Lin. Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions. *arXiv preprint arXiv:1806.09228*, 2018. 1

[49] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018. 1

[50] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. 6, 7, 8

[51] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *European Conference on Computer Vision (ECCV)*, pages 184–199, 2018. 7