Dirichlet Pruning for Neural Network Compression
Supplementatry material

1 Comparison between the posterior distributions learned by taking implicit gradients using the inverse CDF of Gamma and by taking explicit gradients using the analytic mean of Dirichlet.

We consider a model with a single hidden-layer for a binary classification problem, with varying dimensions for input and hidden-layer, where input dimension d_x and hidden dimension d_h pairs we used are ($d_x = 100, d_h = 20$); ($d_x = 500, d_h = 200$); and ($d_x = 1000, d_h = 500$). We assume that there is a known set of ground-truth switch values (which we simulated) in each of these models, as well as a known set of ground-truth parameter values W_1, W_2, b_1, b_2 (for one hidden layer MLP, which we also simulated). Then, we generate inputs from an isotropic Gaussian with zero vector mean and variance 1, which we propagate through each of the models for producing the class labels 0. We also generate inputs from an isotropic Gaussian with the vector of 2's for the mean and variance 0.2 and generated the class labels 1. Given these datasets, under each model, we then compute the posterior distributions over the importance switch.

In Fig. 1, we find that the posterior distribution (Left column) with the inverse CDF of the Gamma distribution provides smaller posterior variance (more certain) than the one with analytic mean, regardless of the dimension of the switch, while the difference in variance gets larger with a higher dimension of the switch. However, the difference in the posterior means in two cases is negligible. In terms of the computation time, using the analytic mean of Dirichlet is a clear winner (no need to sample from the posterior), significantly faster than the case with sampling and using the inverse CDF of the Gamma distribution for implicit gradient computation. For instance, when $d_x = 1000$ and $d_h = 500$, drawing 1000 samples from the Gamma distribution and computing the Monte Carlo integration of the integral, and computing the gradients of the integral wrt the parameters of the switch took[1] 14.317 seconds for performing one epoch training with mini-batch size 100 for the dataset size $N = 4000$. On the other hand, without drawing samples, simply taking the analytic mean of the Dirichlet to approximate the integral and computing the gradients of this approximation with respect to the parameters of the switch takes only 0.570 seconds for the same one epoch training.

2 Descriptions of the benchmark methods.

In the case of the compression experiment, the following benchmark methods are used:

- L1-norm and L2-norm. In the case of convolutional neurons we compute L1- or L2-norm of the parameters which belong to the given channel n_i for $n_i \in [1, 2, \ldots, N_l]$. In the case of fully-connected neuron for $n_i \in [1, 2, \ldots, N_l]$, we compute L1- or L2-norm of all the incoming weights from the previous layer to the parameter n_i
- Group Lasso (GL) [10] - the method combines non-structured regularization applied to every weight with structured sparsity regularization applied to every layer. Then for groups of parameters it uses group regularization during training using L2-norm (the name of the method is a misnomer).

[1] This computation time is measured on a desktop using Intel Xeon W-2135 CPUs at 3.70GHz.

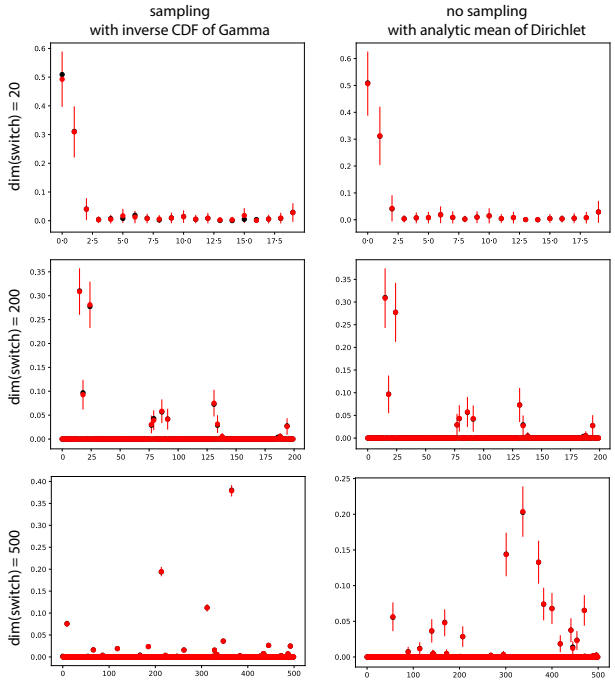

Figure 1: Comparison of posterior distributions with varying switch dimension. Black dots indicate true switch values from which we generated data. Red dots indicate the posterior means and the red vertical lines are one posterior standard deviation.

- Derivative-based pruning [1, 9], where we compute the importance of the channel based on the gradient of the following activation. $R^{H_l \times W_l \times C_l} \to R$. Let h_i be the output produced by parameter (or in our case channel or a set of parameters) and $\mathcal{C}$ be a cost function, negative log-likelihood being the most common choice. Then following [4] the cost of removing this channel is

$$\Delta\mathcal{C}(h_i = 0) = \mathcal{C}h_i h_i.$$

- Bayesian Compression for Deep Learning (BD) [3] - the approach proposes to phrase the compression problem as a variational inference optimization with hierarchical sparsity-inducing prior to prune groups (G) of parameters (nodes) in a CNN. The first method (BC-GNJ) uses the improper log-uniform prior and its alternative reparametrization known as normal-Jeffrey (NJ) prior. The second method (BC-GHS) incorporates prior hierarchy that uses two Half-Cauchy distributions, which induce horseshoe (HS) prior distribution of the weights.
- Flops as a Direct Optization Objective (FDOO) [8] - the method concentrates on the problem of reducing FLOPs (rather than parameters). They construct a loss function which incorporates the number of FLOPs as a variable. The loss function is phrased as a fully-factorized spike-and-slab posterior with the number of FLOPs as a sparsity inducing prior. The optimization of the FLOPS uses score function estimator (REINFORCE). Two results correspond to the best compression rates given two FLOPs budgets (100K and 200K FLOPs).
- Structured Bayesian Pruning via Multiplicative Noise (SBP) [5] - the authors introduce a dropout-like layer with a certain kind of multiplicative noise (Bernoulli or Gaussian). Besides, sparsity-inducing log-uniform prior is used, but it is placed over the noise variables rather than weights. The sparsification is obtained during the variational lower bound training.

- Data-free parameter pruning [7] - the method uses neuron similarity to remove redundant neurons. Assuming the same activation for all the outputs, the neurons which have similar weights are first combined and then discarded.

3 The layer-by-layer compressed architectures.

Method	Architecture	Error	FLOPs	Params
Dirichlet (150)	**6-8-40-20**	1.1	168K	**6K**
Dirichlet (mean)	**5-8-45-15**	1.1	140K	**5.5K**
Dirichlet (joint)	**6-7-35-17**	1.1	158K	**5.5K**
BC-GNJ [3]	8-13-88-13	1.0	288K	15K
BC-GHS [3]	5-10-76-16	1.0	159K	9K
RDP [6]	4-7-110-66	1.0	117K	16K
FDOO (100K) [8]	2-7-112-478	1.1	**113K**	63K
FDOO (200K) [8]	3-8-128-499	1.0	157K	76K
GL [10]	3-12-192-500	1.0	211K	112K
GD [7]	7-13-208-16	1.1	273K	29K
SBP [5]	3-18-284-283	**0.9**	226K	99K

Table 1: The structured pruning of **LeNet-5**.

Method	Architecture	Error	FLOPs	Params
Dirichlet	**41-41-75-75-82-113-108-99-109-99-99-74-74-69-69**	8.48	**38M**	**0.84M**
BC-GNJ	63-64-128-128-245-155-63-26 -24-24-20-14-12-11-15	8.3	142M	1.0M
BC-GHS	51-62-125-128-228-129-38-13-9-6-5-6-6-20	9.0	122M	0.8M
RDP	27-57-125-122-236-244-246-340-127-77-89-52-380-414	8.7	172M	3.1M

Table 2: The structured pruning of **VGG-16**.

Method	Architecture	Error	Comp. Rate	Params
Dirichlet	**65,75,70,90,149,149,164,149,305,325,304,306**	4.5	52.2%	**17.4M**
L_0 ARM [2]	82-75-82-87-164-169-156-161-317-317-317-324	4.4	49.9%	18.3M
L_0 ARM [2]	75-72-78-78-157-165-131-162-336-325-331-343	**4.3**	49.6%	18.4M

Table 3: The structured pruning of **WideResNet-28-10**.

Method	Architecture	Error	FLOPs	Params
Dirichlet	**6,11,16**	14M	0.24M%	**17.4M**

Table 4: The structured pruning of **Resnet-56**. The architecture indicates the number of channels of the inner layer within each block of two convolutioal layers, for each of the three section containing nine blocks.

4 The complete channel visualization for the VGG-16 first convolutional layer trained on CIFAR-10.

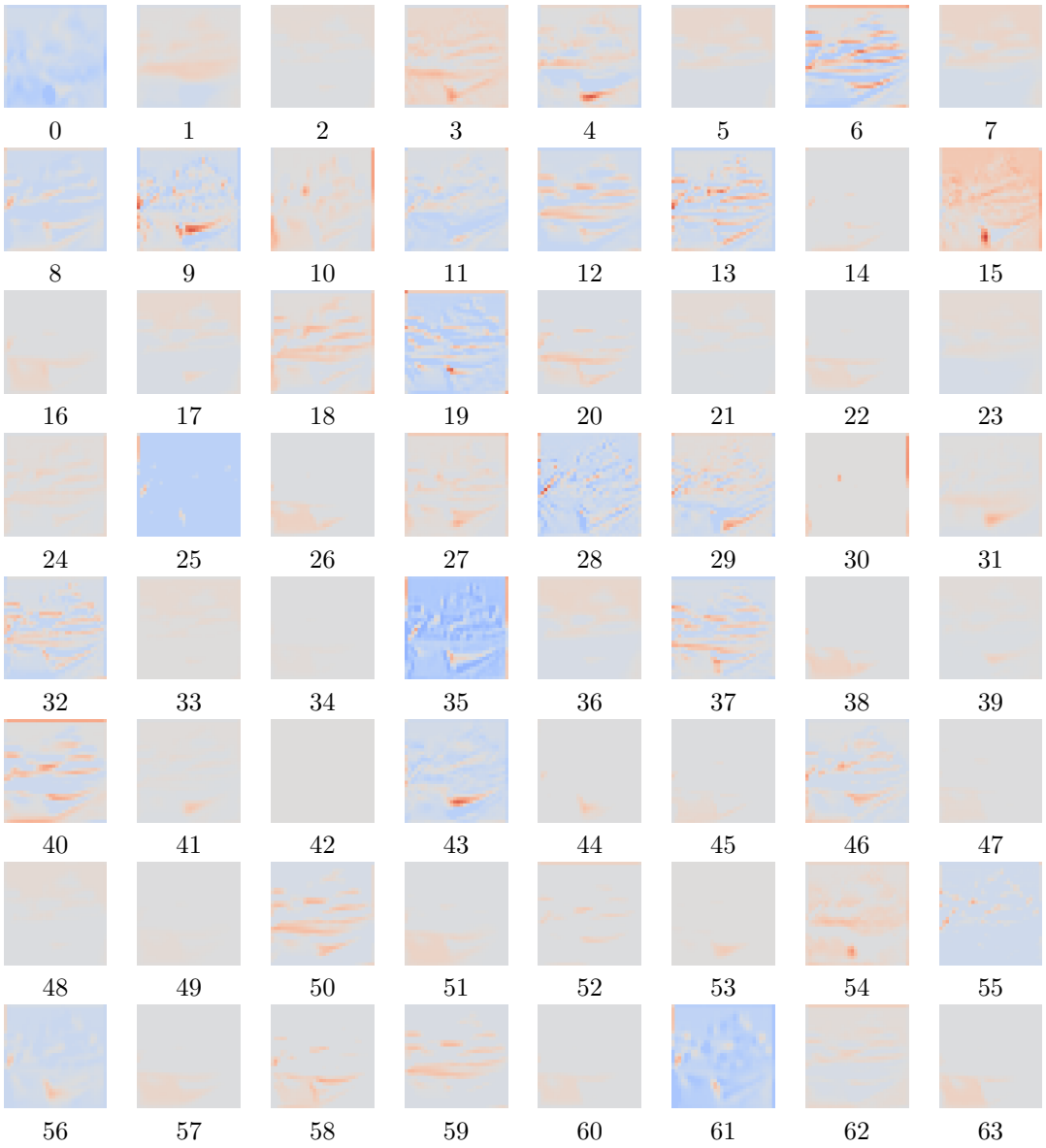

Figure 2: The figure presents the complete feature maps of the first convolutional layer learnt by the VGG network (presented also in Figure 1 of the main paper). The colormap indicates higher pixel values with red and lower with blue. The top filters learnt by the importance switch method are 28,15,4,29,54,9,13,3,43,6. Notice that these feature maps include the features with high activation values.

References

[1] 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.

[2] Yang Li and Shihao Ji. l_0-arm: Network sparsification via stochastic binary optimization. *arXiv preprint arXiv:1904.04432*, 2019.

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

[4] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440*, 3, 2016.

[5] 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.

[6] Changyong Oh, Kamil Adamczewski, and Mijung Park. Radial and directional posteriors for bayesian neural networks. *arXiv preprint arXiv:1902.02603*, 2019.

[7] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

[8] Raphael Tang, Ashutosh Adhikari, and Jimmy Lin. Flops as a direct optimization objective for learning sparse neural networks. *arXiv preprint arXiv:1811.03060*, 2018.

[9] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

[10] 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.