Adversarial Structured Neural Network Pruning

Xingyu Cai
University of Connecticut

Jinfeng Yi
JD AI Research

Fan Zhang
Zhejiang University

Sanguthevar Rajasekaran*
University of Connecticut

ABSTRACT

In recent years, convolutional neural networks (CNN) have been successfully employed for performing various tasks due to their high capacity. However, just like a double-edged sword, high capacity results from millions of parameters, which also brings a huge amount of redundancy and dramatically increases the computational complexity. The task of pruning a pretrained network to make it thinner and easier to deploy on resource-limited devices is still challenging. In this paper, we employ the idea of adversarial examples to sparsify a CNN. Adversarial examples were originally designed to fool a network. Rather than adjusting the input image, we view any layer as an input to the layers afterwards. By performing an adversarial attack algorithm, the sensitivity information of the network components could be observed. With this information, we perform pruning in a structured manner to retain only the most critical channels. Empirical evaluations show that our proposed approach obtains the state-of-the-art structured pruning performance.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Neural networks**; *Artificial intelligence*; *Discrete space search*.

ACM Reference Format:
Xingyu Cai, Jinfeng Yi, Fan Zhang, and Sanguthevar Rajasekaran. 2019. Adversarial Structured Neural Network Pruning. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM ’19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3357384.3358150

1 INTRODUCTION

In the past decade, deep neural networks (DNNs) have made remarkable achievements in different AI domains, such as ImageNet challenge, Go game, machine translation, etc. However, the complexity of a DNN has also increased, in order to provide the required capacity for those sophisticated tasks. An over-parameterized network not only requires more and more computation power in both training and inference phases, but also sometimes even hurts the performance of the DNN. In fact, Dropout is the best example to show that a dense network performs much worse than its sparse counterpart, in terms of generalization ability. Exploiting the benefits from sparsity and capacity, researchers are finally able to achieve a better performance compared to a directly trained dense network.

It is well known that a significant fraction of a modern DNN’s parameters are redundant. To identify the redundancy in a network, an exhaustive search in the structure space is believed to be not practical, thus researchers have brought up many ideas to eliminate redundancy without loosing capacity. One typical attempt is knowledge distilling, which could entirely change the original network architecture. The others generally fall into the category of network pruning, where the original model skeleton is kept, but the less significant components are carefully removed. Weight level pruning is typically performed on smaller sized networks such as LeNet, while structured or channel-wise pruning is performed on larger CNNs such as VGG and ResNet. Both distilling and pruning could lead to a better interpretability and hence have attracted growing research interests in recent years. In this paper, we address the sparsification problem via structured pruning, and provide an efficient solution obtaining state-of-the-art results.

Related Work

Observing that the network parameters (weights) can be expressed in a matrix form, matrix approximation techniques have been adopted to achieve sparsity. For example, [4] proposed a low-rank matrix factorization to sparsify weights and provide feature predictions. As an extension, tensor decomposition is also utilized in [5] to speed up the training of convolutional neural networks.

Other than factorization approaches, pruning methods have been shown to be more efficient. In [7], the authors pruned the parameters that have smaller values, and found it can largely sparsify the network while keeping the same classification accuracy. Later in [6], the authors combined pruning with the idea of low-precision representation, to achieve higher compression ratios in terms of storage. Following a Bayesian learning framework, [17] offered a mixture of Gaussians as priors to optimize the loss function and achieved simultaneous pruning and training. In [13], the authors use variational dropout to provide a method called SparseVD that achieves the best known weight sparsification performance, in several network structures. Quantization approaches such as SqueezeNet [9], use a 6-bit representation to obtain 510x reduction of model size of AlexNet. [10] even observed that an extreme 2 or 3 digits representation could be enough to achieve a high accuracy.

To obtain interpretable results, structured pruning attracts more and more attention recently. Typically, structured pruning directly operates on CNN channels or blocks, rather than individual weights. The benefit of preserving the channel structure is to take advantage of modern algebra libraries

[*] Corresponding author.

CIKM ’19, November 3–7, 2019, Beijing, China

ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3358150

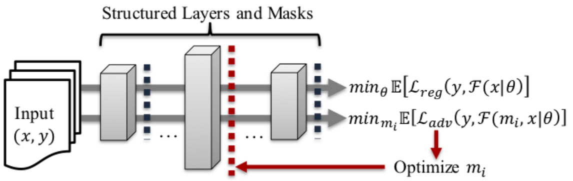

Structured Layers and Masks
Input
(x, y)
$min_\theta \mathbb{E}[\mathcal{L}_{reg}(y, \mathcal{F}(x|\theta)]$
$min_{m_i} \mathbb{E}[\mathcal{L}_{adv}(y, \mathcal{F}(m_i, x|\theta)]$
Optimize m_i

Figure 1: Adversarial Structured Pruning Diagram

that apply fast matrix operations. Adding or removing entire channels rather than individual weights could achieve much faster computation in practice. In [18], the authors propose a method called Structured Sparsity Learning (SSL) to regularize the filters in DNNs. A Bayesian approach to prune channels is proposed in [14]. Similarly, [11] uses hierarchical priors and achieve very compact networks compared to previous methods. In [12], the authors adopt an ℓ_0 regularization to effectively remove redundant neurons or channels. To take advantage of convexity, the authors of [8] put masks on the network sub-blocks, and apply ℓ_1 penalty on those masks. To question the common practice that smaller values mean less importance, the authors in [19] proposed a channel pruning approach that does not rely on this belief, and showed it to be also effective in their experiments.

Adversarial example [16] is a small perturbation (usually indistinguishable by human) in the input that leads to erroneous DNN outputs. It has been shown that DNNs have nearly zero defense against adversarial attacks [1]. To generate adversarial samples, many optimization based techniques are proposed such as Carlini & Wagner Attack [2], Elastic-Net Attack [3], etc. The general idea is to apply gradient based approaches to minimize a constraint adversarial loss function, which is designed to fool the network, e.g., output the wrong label. The constraint guarantees that the perturbed input is very close to the original, due to the fact that the larger the perturbation, the easier will it be to find adversarial examples. In [15], adversarial loss is used to construct a novel dropout scheme, and achieve very good accuracies in several models.

Overview of the Proposed Approach

In this paper, we address the problem of structured sparsification (channel-wise pruning) given a pretrained network. Inspired by adversarial attacks, we employ this idea to identify network components that are sensitive to the adversarial loss. Unlike a defense method that wants to eliminate such sensitivity, we instead would like to keep them and remove the insensitive parts to achieve sparisity. In contrast to the other pruning methods that do random perturbations on the output, or simply prune small values, the adversarial loss points us to the "best direction", which guides us to preserve the channels that contribute most to the final network output.

It is worth noting that **Adversarial Dropout** [15] shares similar spirits, by applying masks trained on adversarial loss. However, the main difference between our work and Adv-dropout is that: 1) the detailed method is entirely different (illustrated in the next section); 2) we are solving a different problem. Since the storage size is out of interest, quantization methods are not adopted or compared in this paper.

The key contributions of this paper include: We propose a sparsification method that alternates between a regular training step and an adversarial pruning step, in a layer-wise manner. To the best of the authors' knowledge, this is the first attempt to sparsify a network via solving a constrained adversarial optimization problem. We adaptively adjust the constraint hyper-parameters in the adversarial pruning phase for different layers, to achieve the best layer sparsity. We achieve state-of-the-art results in popular models like VGG and ResNet. We empirically analyze the sparsity-accuracy trade-off, and the impact on the adversarial robustness of the pruned network.

2 ADVERSARIAL STRUCTURED PRUNING

2.1 Notations and Definitions

Let $\mathcal{F} : \mathcal{R}^p \to \mathcal{R}^q$ be the network model of interest, where p and q are the input and output dimensions. Without loss of generality, we narrow the model for classification tasks, thus q could be the number of categories. $\mathcal{F}$ is parameterized by θ, and θ consists of all trainable parameters like weights and bias. We denote $x \in \mathcal{R}^p$ as the input, associated with a label $y \in \{1, \ldots, q\}$. (x, y) are pairs from the training set $\mathcal{X}$.

Assume that $\mathcal{F}$ has a total of K structured layers, e.g., Convolution + Batch-Norm + ReLU. Let $w = \{w_i\}, i \in [0, K-1]$ represent the output of each structured layer. For example, in a convolutional layer, $w_{i,j} \in \mathcal{R}^{d \times d}$ could be the channel j's output, which is a feature map of size $d \times d$. **We append a mask to each channel to switch on or off the entire channel.** Formally speaking, for each $w_{i,j}$, we associate a mask $m_{i,j} \in \{0, 1\}$. Then the masked output for layer i's channel j (assuming in total n channels) is written as $o_{i,j} = w_{i,j} \odot \text{broadcast}(m_{i,j}), \quad i \in [0, K-1], \ j \in [0, n]$, where $\odot$ is the Hadamard (element-wise) product, and broadcast() duplicates scalar $m_{i,j}$ to a matrix shaped $d \times d$. The final output of the model can be denoted as $\mathcal{F}(x|\theta)$. Since layer $(i-1)$'s masked output is layer i's input, we can write the layer-wise relationship as: $w_i = \mathcal{F}_i(o_{i-1}|x, \theta), \ i \in [0, K-1]$. This is illustrated in Figure 1.

2.2 Regular Training Step

The method alternates between a regular training phase and an adversarial pruning phase. For the regular training, just as a standard DNN training over the training set $\mathcal{X}$, we solve

$$\theta = \arg\min_{\theta} \mathbb{E}_{(x,y)\in\mathcal{X}}[\mathcal{L}_{reg}(y, \mathcal{F}(x|\theta))] + \alpha R(\theta) \tag{1}$$

where $\mathcal{L}_{reg}$ is the regular training loss, typically being a cross entropy loss for classification tasks. R is the regularizer with a factor of α. This is done via standard techniques like SGD.

2.3 Adversarial Pruning Step

Constructing Adversarial Loss. For an input x, we solve a following problem to get an adversarial sample $x + \delta$:

$$\begin{aligned} &\min_{\delta} \ \mathbb{E}_{(x,y)\in\mathcal{X}}[\mathcal{L}_{adv}(x, \epsilon, \delta|\theta)] \\ &s.t. \ (1-\epsilon)||x|| \le ||x+\delta|| \le (1+\epsilon)||x|| \end{aligned} \tag{2}$$

where δ is the perturbation on the input, ϵ is the allowable deviation from x's original norm, and typically less than 0.1.

Conventionally, it is often the ℓ_∞ norm that is used to ensure that each pixel will not be significantly modified, but ℓ_1 is also used to promote sparsity. $\mathcal{L}_{adv}$ is the adversarial loss that tries to fool the network deviating from the correct label. In this paper, we adopt a simple form: $\mathcal{L}_{adv} = -\mathcal{L}_{reg}(y, \mathcal{F}(x+\delta|\theta))$, which means that we simply want to maximize the regular loss to the correct label, but are not concerned with the distribution of the final softmax output.

Minimizing Adversarial Loss. In the adversarial pruning phase, for layer i, we fix all the other parameters (θ, m_j for $j \neq i$), but only optimize m_i. In particular, we solve

$$\min_{\hat{m}_i \in \mathcal{C}_i} -\mathbb{E}_{(x,y)\in\mathcal{X}}[\mathcal{L}_{reg}(y, \mathcal{F}(\hat{m}_i, x|\theta)] \tag{3}$$

where $\mathcal{C}_i$ is the constraint set parameterized by ϵ_i, i.e., $\mathcal{C}_i = \{\hat{m}_i : (1-\epsilon)||w_i||_\infty \leq ||\hat{m}_i \odot w_i||_\infty \leq (1+\epsilon)||w_i||_\infty\}$. To solve Equation 3, we use the projected gradient descent (PGD) method. Initializing all $m_i = \mathbf{1}$, the PGD's update step is: $\hat{m}_i \leftarrow \mathrm{Proj}_{\mathcal{C}_i}(\hat{m}_i - \eta\nabla_{\hat{m}_i}\mathbb{E}[\mathcal{L}_{adv}])$, where η is the step size. $\mathrm{Proj}_{\mathcal{C}_i}$ operator ensures projection onto the $\mathcal{C}_i$. Due to ℓ_∞ norm, the projection is simplified to be a hard thresholding function applied on each element $\hat{m}_{i,j}$:

$$\mathrm{Proj}_{\mathcal{C}_i}(\hat{m}_i) = \begin{cases} \hat{m}_{i,j}, & 1-\epsilon \leq \hat{m}_{i,j} \leq 1+\epsilon \\ 1+\mathrm{sign}(\hat{m}_{i,j}-1)\epsilon, & \text{otherwise} \end{cases} \tag{4}$$

Sparsification. After obtaining the updated soft-mask $\hat{m}_i$, we then perform pruning based on $\hat{m}_i$. There will be two cases.

Case 1 happens when the solution of Equation 3 hits $\mathcal{C}_i$'s boundaries. More precisely, $\hat{m}_{i,j} = 1 \pm \epsilon$, for some $j \in [0, |\hat{m}_i| - 1]$. Note that this can always happen as long as we set ϵ small enough, meaning that we can shrink the constraint set to let the unconstrained solution of Equation 3 be outside of $\mathcal{C}_i$. Thus the $\mathrm{Proj}_{\mathcal{C}_i}$ operator would clamp some elements to project back to $\mathcal{C}_i$. In this case, we simply keep the elements that are hitting or very close to the boundaries, and remove the others. So the clipping functions would be

$$m_i = \mathrm{Clip}(\hat{m}_i) = 1 \text{ if } |\hat{m}_{i,j}| \geq 1+\tau\epsilon; \;\; 0, \text{ otherwise} \tag{5}$$

where τ is a threshold parameter. In our experiments we set $\tau = 0.9$. By performing such clipping, we are removing components that are not sensitive to the adversarial loss.

Case 2 is that the $\mathrm{Proj}_{\mathcal{C}_i}$ operator does not actually perform clamping, so the unconstrained solution is already inside $\mathcal{C}_i$. Thus we need to change the strategy, and only remove the smallest s elements, in a greedy way. For example, in the previous iteration, layer i already removed s_i components, so at this iteration we just try to remove $s_i + k_i$ smallest elements, where k_i is dynamically adjusted starting from 1. The dynamic adjusting of k_i can be described as: $k_i \leftarrow 2k_i$ if performance (test accuracy) did not drop in last iteration, otherwise $k_i \leftarrow k_i/2$. So in case 2, the clipping function is defined as:

$$m_i = \mathrm{Clip}(\hat{m}_i, s) = 0 \text{ if } |\hat{m}_{i,j}| \text{ is within } s \text{ smallest}; \;\; 1, \text{ otherwise} \tag{6}$$

2.4 Putting Together

After obtaining m_i by clipping, to alleviate the problem of accuracy drop due to sparsity, for each layer i, we need to perform a regular training (Equation 1) to adjust θ correspondingly, using Adam method. If regular training could not yield a satisfactory performance within T iterations, we reverse the mask to be previous m_i. Usually we adopt a learning rate decaying schedule, e.g., every I ($I < T$) iterations reduce the learning rate to 1/10, for the regular training. Note that the adversarial phase does not need to have any scheduling in practice. Starting from the layer 0 (closest to the input), we perform both training for each layer. After iterating all the layers, we loop back. If all the layers are no longer able to be sparsified, we terminate the entire process.

3 EXPERIMENTAL EVALUATION

Following the recent papers [14], [19], we experiment on VGG-like and ResNet20. The structured pruning is performed on the channel level. We report the each layer's sparsity, the error rate, the **pruning ratio** ρ **and FLOPs saving ratio** β. ρ is the dense model's parameter number, divided by the pruned model's parameter number, i.e., $\rho = |\theta_{\text{orig}}|/|\theta_{\text{sparse}}|$, and β is dense model's FLOPs divided by the pruned model's.

We run VGG-like and ResNet20 experiments 5 times each, then report the averaged results. The hyper parameters include: $T = 10, \tau = 0.9$; adv learning rate $\eta_{\text{adv}} = 1e-2$; reg learning rate $\eta_{\text{reg}} = 1e-2$ for VGG-like and $1e-4$ for ResNet20; starting $\epsilon = 1e-3$ for VGG-like and 0.1 for ResNet20. Note that most of them are simply set by default, but ϵ for VGG-like is obtained after several tests. This is because the behavior for earlier and later layers of VGG are very different, where the default $\epsilon = 0.1$ is not suitable for later layers. Such phenomenon of VGG is also reported in [8].

VGG-like on CIFAR10 Dataset

VGGs are known to have a large redundancy, and we follow the literature to test on the 16-layer VGG-like model on CIFAR10 dataset. VGG-like has two linear layers, and takes a parameter k as a factor to control the number of neurons or channels in each layer, where in standard case $k = 1.0$. We evaluate the VGG-like model with $k = 1.0$ and $k = 1.5$ (same as in [14]). The sparsification results are shown in Table 1.

From the table we can easily see that the proposed scheme performs much better than the other methods. Especially for the later layers that have a large number of channels (512 or 768), the adversarial sparsification method can significantly eliminate redundancy. This is due to the smaller value of adversarial boundary $\epsilon = 10^{-3}$ applied on this model. In our experiments, we also tested a default $\epsilon = 0.1$ and found it does not perform very well. The default $\epsilon = 0.1$ works normally in the early layers, however, we observe that no elements hit the ϵ boundary from the beginning for the later layers. Thus these layers start clipping with Equation 6 instead of Equation 5, and lose most of the benefit of adversarial pruning. This is why we need to set ϵ to be smaller than the default value, and found 10^{-3} could work well for these later layers.

ResNet20 on CIFAR10 Dataset

In this subsection, we tested on another popular network model ResNet20, on CIFAR10 dataset. We adopt the same

Table 1: Sparsity and FLOPs in VGG-like, $k = 1.0$ and $k = 1.5$

k	Algorithm	Layer Sparsity														Error %	ρ (Params)	β (FLOPs)
1.0	Original	64	64	128	128	256	256	256	512	512	512	512	512	512	512	7.2	1.0	1.0
	SparseVD [13]	64	62	128	126	234	155	31	81	76	9	138	101	413	373	7.2	3.168	1.946
	StructuredBP [14]	64	62	128	126	234	155	31	79	73	9	59	73	56	27	7.5	6.255	2.061
	StructuredBPa[14]	44	54	92	115	234	155	31	76	55	9	34	35	21	280	9.0	7.705	2.319
	ours 1	32	38	86	86	155	149	100	150	3	2	2	2	18	12	7.9	**10.571**	**2.692**
	ours 2	24	48	91	96	193	179	82	40	4	2	3	3	4	7	8.9	**12.464**	**2.485**
1.5	Original	96	96	192	192	384	384	384	768	768	768	768	768	768	768	6.8	1.0	1.0
	SparseVD[13]	96	78	191	146	254	126	27	79	74	9	137	100	416	479	7.0	4.691	2.575
	StructuredBP[14]	96	77	190	146	254	126	26	79	70	9	71	82	79	49	7.2	8.616	2.711
	StructuredBPa[14]	77	74	161	146	254	125	26	78	66	9	47	55	54	237	7.9	9.719	2.845
	ours	37	56	117	136	274	192	40	3	1	2	1	2	8	4	7.83	**18.747**	**3.199**

model as in [19] and compare the sparsity with their reported results. The ResNet20 includes 3 ResNet groups, where each group consists of 3 blocks, and each block has 2 convolutional layers. The sparsification result is shown in Table 2.

Table 2: Sparsity and FLOPs in ResNet20

Alg	Orig	[19]	**ours**
Group 1	16×6	12-12-6-6-11-11	1-2-1-1-12-10
Group 2	32×6	32-32-28-28-28-28	11-16-20-16-14-15
Group 3	64×6	47-47-34-34-25-25	52-56-47-35-51-18
Err %, ρ, β	8.0, 1.0, 1.0	9.1, 1.593, 1.451	**9.0, 1.613, 2.055**

Sparsity-Accuracy Trade-off

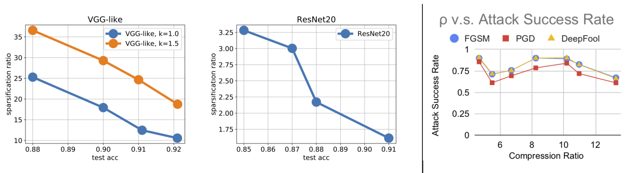
It is well known that the lower the accuracy, the more elements could be safely removed from the network. We preset the accuracy level that the network needs to maintain during the sparsification process, then try to prune as much as possible. Figure 2 (a) (b) show the sparsity-accuracy trade-off for VGG-like and ResNet20 networks.

Impact on Adversarial Robustness

Since we employ adversarial attack methods to prune the network and preserve the sensitive channels, it is interesting to evaluate the adversarial robustness of the pruned network. We perform 3 commonly-used attack techniques, FGSM, PGD and DeepFool (from Foolbox package), on the VGG-like ($k = 1.0$) network, to see how attack success rate r changes when the compression ratio ρ increases. The empirical study reveals no significant robustness decay on the pruned network. The attack success rate r remains similar when ρ changes from 4 to 13. Details can be found from Figure 2 (c).

4 CONCLUSIONS AND FUTURE WORK

In this paper, we propose a novel Adversarial Structured Pruning method. This approach leverages an iterative process that alternates between a regular training and an adversarial pruning step, in a layer-wise manner. The adversarial pruning step solves a constrained optimization problem where the results are used as a guideline for channel pruning. The components that are insensitive to the adversarial loss, would be removed without sacrificing the model capacity. The experiments reveal that this method achieves the state-of-the-art structured pruning performance. The accuracy-sparsity trade-off is empirically studied, which can be used as a guideline in practice. We also carry out a preliminary study on the adversarial robustness of the pruned network, and the initial results do not show any robustness decay after pruning. In future, we can adopt the quantization techniques to further improve the compression rate.



(a) VGG-like (b) ResNet20 (c) ρ vs r

Figure 2: (a) (b) sparsity-acc trade-off; (c) ρ vs r

REFERENCES

[1] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420* (2018).

[2] Nicholas Carlini and David Wagner. 2017. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 3–14.

[3] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. 2017. EAD: elastic-net attacks to deep neural networks via adversarial examples. *arXiv preprint arXiv:1709.04114* (2017).

[4] Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. 2013. Predicting parameters in deep learning. In *NIPS*. 2148–2156.

[5] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*. 1269–1277.

[6] Song Han, Huizi Mao, and William J Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *International Conference on Learning Representations (ICLR)* (2016).

[7] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *NIPS*. 1135–1143.

[8] Zehao Huang and Naiyan Wang. 2017. Data-driven sparse structure selection for deep neural networks. *arXiv preprint arXiv:1707.01213* (2017).

[9] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5MB model size. *arXiv preprint* (2016).

[10] Cong Leng, Hao Li, Shenghuo Zhu, and Rong Jin. 2017. Extremely low bit neural network: Squeeze the last bit out with admm. *arXiv preprint* (2017).

[11] Christos Louizos, Karen Ullrich, and Max Welling. 2017. Bayesian compression for deep learning. In *NIPS*.

[12] Christos Louizos, Max Welling, and Diederik P Kingma. 2017. Learning Sparse Neural Networks through L_0 Regularization. *arXiv preprint* (2017).

[13] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. 2017. Variational Dropout Sparsifies Deep Neural Networks. *arXiv preprint* (2017).

[14] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. 2017. Structured bayesian pruning via log-normal multiplicative noise. In *NIPS*. 6778–6787.

[15] Sungrae Park, Jun-Keon Park, Su-Jin Shin, and Il-Chul Moon. 2017. Adversarial Dropout for Supervised and Semi-supervised Learning. *arXiv preprint arXiv:1707.03631* (2017).

[16] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[17] Karen Ullrich, Edward Meeds, and Max Welling. 2017. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008* (2017).

[18] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *NIPS*. 2074–2082.

[19] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. 2018. Rethinking the Smaller-Norm-Less-Informative Assumption in Channel Pruning of Convolution Layers. *arXiv preprint arXiv:1802.00124* (2018).