Growing Efficient Deep Networks by Structured Continuous Sparsification

Xin Yuan
University of Chicago
yuanx@uchicago.edu

Pedro Savarese
TTI-Chicago
savarese@ttic.edu

Michael Maire
University of Chicago
mmaire@uchicago.edu

Abstract

We develop an approach to training deep networks while dynamically adjusting their architecture, driven by a principled combination of accuracy and sparsity objectives. Unlike conventional pruning approaches, our method adopts a gradual continuous relaxation of discrete network structure optimization and then samples sparse subnetworks, enabling efficient deep networks to be trained in a growing and pruning manner. Extensive experiments across CIFAR-10, ImageNet, PASCAL VOC, and Penn Treebank, with convolutional models for image classification and semantic segmentation, and recurrent models for language modeling, show that our training scheme yields efficient networks that are smaller and more accurate than those produced by competing pruning methods.

1 Introduction

Deep neural networks (DNNs) have achieved dramatic accuracy improvements in a variety of machine learning tasks such as image classification [26, 45], object detection [7, 32], semantic segmentation [1, 33] and language modeling [49]. Even though DNNs are typically overparameterized, recent work [14, 20, 48] shows that their performance on numerous tasks can be further improved by increasing their depth and width. Despite their success on benchmark datasets, the training and deployment of DNNs in many real-world applications is limited by their large number of parameters and computational costs. To address this, model compression and architecture search methods that learn more efficient DNN models have been proposed, yielding faster training and inference.

Efficiency improvements to DNNs have been extensively studied in previous works [18, 19, 23, 50]. For example, [22, 41] propose the use of binary weights and activations, benefiting from reduced storage costs and efficient computation through bit-counting operations. Other prominent approaches focus on finding efficient alternatives to standard spatial convolutions, *e.g.* depth-wise separable convolutions [44], which applies a separate convolutional kernel to each channel followed by a point-wise convolution over all channels [3, 18, 50]. Pruning methods [10, 11, 12] aim to generate a light-wise version of a given network architecture by removing individual weights [11, 12, 38] or structured parameter sets [15, 28, 35] (*e.g.* filters in convolutional layers).

However, existing methods typically rely on retraining or fine-tuning phases after reducing the number of parameters so that accuracy is maintained, resulting in significant computational costs. Moreover, the majority of these methods train the full-sized model prior to pruning and do not aim to diminish train-time computational costs. Recently proposed network architecture search (NAS) methods [31, 36, 40, 42, 51, 52] utilize AutoML techniques to design efficient architectures under practical resource constraints. Nonetheless, most NAS methods operate on a large "supernet" architecture, yielding a computationally expensive search phase. In addition, few of the recently-proposed methods are one-shot and hence require additional computation to retrain the final architecture in order to achieve high performance for deployment.
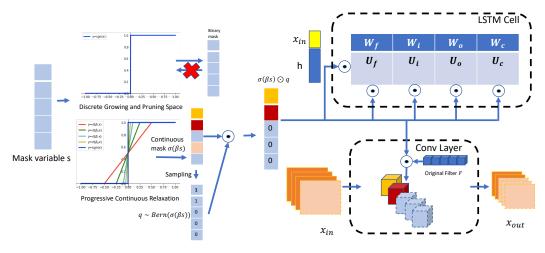
Figure 1: Framework of our proposed method. (**Left**) Our learning by continuation method is comprised of three main components: (1) To tackle the optimization hardness from the discrete growing and pruning space, we follow [43] and replace the sign operation $\text{sign}(s)$ with a gradual smooth function $\sigma(\beta s)$: the sign function is shown in blue, while red, green, cyan and yellow show $\sigma(\beta s)$ with bandwidths $\beta_r < \beta_g < \beta_c < \beta_y$. (2) β is controlled by a carefully designed bandwidth scheduler towards $\lim_{\beta \to \infty} \sigma(\beta s) = \text{sign}(s)$. (3) A binary stochastic auxiliary variable q sampled according to $\sigma(\beta s)$ is introduced to implicitly reduce computational cost during the progressive training stage. (**Right**) The relaxation can be applied to both CNN layers and RNN cells on various computer vision and natural language processing tasks. *Best viewed in color.*

We propose a method to dynamically grow deep networks by continuously sparsifying structured parameter sets, resulting in efficient architectures and decreasing the computational cost not only of inference, but also of training. Unlike existing pruning or architecture search schemes that maintain a full-sized network or a "supernet", we implicitly adapt architectures during training with different structured sparsity levels. More specifically, we first build a discrete space to maintain and explore adaptive train-time architectures of different complexities in a growing and pruning manner. To overcome the hardness of optimizing over a discrete space, we perform learning via continuation methods by approximating a discrete operation through a scaled smooth function. We design a *bandwidth scheduler* that is used to control the optimization hardness during this low-cost training procedure. The framework is illustrated in Figure 1. We conduct extensive experiments on classification tasks (CIFAR-10, ImageNet), semantic segmentation (PASCAL VOC) and word-level language modeling (PTB) to demonstrate the effectiveness of our methods for both convolutional neural network (CNN) and recurrent neural network (RNN) architectures.

2 Related Work

Network Pruning: Network pruning methods can be split into two groups: those that prune individual weights and those that prune structured components. For individual weight-based pruning, elements of the weight matrices can be removed based on some criteria. For example, [12] propose to prune network weights with small magnitude, and build a deep compression pipeline [11]. Sparse VD [38] yields extremely sparse solutions both in fully-connected and convolutional layers by using variational dropout. [34] learns sparse networks by approximating ℓ_0-regularization with a stochastic reparameterization. [39] presents a magnitude-based pruning approach for RNNs where the top-k elements of the weights are set as 0 at each iteration. However, these methods that produce sparse weight matrices only lead to speedup on dedicated hardware with supporting libraries.

In structured methods, pruning is applied at the level of neurons, channels, or even layers. For example, L1-pruning [28] removes channels based on the norm of their filters. [15] uses group sparsity to smooth the pruning process after training. ThiNet [35] greedily prunes the channel that has the smallest effect on the next layer's activation values. MorphNet [8] regularizes weights towards zero until they are small enough such that the corresponding output channels are marked for removal from the network. Intrinsic Structured Sparsity (ISS) [46] works on LSTMs [17] by collectively removing the columns and rows of the weight matrices via group LASSO.

Our work is more related to structured pruning methods in the sense that a slim architecture is generated at the end of training. In addition, our work also focus on adapting the train-time structured sparsification in a discrete growing and pruning space.

Lottery Ticket Hypothesis and Continuous Sparsification: The Lottery Ticket Hypothesis [6] conjectures that sparse sub-networks and their randomly initialized weights can obtain a comparable accuracy with the original network when trained in isolation. [43] further proposes Continuous Sparsification, a method to speed up ticket search, which approaches a complex optimization problem by relaxing the original objective, turning it into an intermediate and easier problem in terms of optimization. By gradually increasing the difficulty of the underlying objective during training, it results in a sequence of optimization problems converging to the original, intractable objective. In our method, we directly adopt Continuous Sparsification [43] to formulate a gradual relaxation scheme in the context of structured pruning.

3 Method

3.1 Discrete Growing and Pruning Space

Given a network topology, we build a discrete space to maintain adaptive train-time architectures of different complexities in a growing and pruning manner. A network topology can be seen as a directed acyclic graph consisting of an ordered sequence of nodes. Each node $x_{in}^{(i)}$ is the input feature and each edge is a computation cell with *structured* hyperparameters (*e.g.* filter numbers in convolutional layers or hidden neuron numbers in recurrent cells). The discrete growing and pruning space can be parameterized by associating a mask variable $m \in \{0, 1\}$ with each computation cell (edge), which enables a train-time pruning ($m = 1 \to 0$) and growing ($m = 0 \to 1$) dynamics.

For a convolutional layer with l_{in} input channels, l_{out} output channels (filters) and $k \times k$ sized kernels, the i-th output feature is computed based on the i-th filter, *i.e.* for $i \in \{1, \ldots, l_{out}\}$:

$$x_{out}^{(i)} = conv(x_{in}, \mathcal{F}^{(i)} \cdot m^{(i)}), \tag{1}$$

where $m^{(i)} \in \{0, 1\}$. For a recurrent cell, without loss of generality, we focus on LSTMs [17] with l_h hidden neurons, a common variant[1] of RNNs that learns long-term dependencies:

$$f_t = \sigma_g((W_f \odot (\mathbf{e}m^T))x_t + (U_f \odot (mm^T))h_{t-1} + b_f)$$
$$i_t = \sigma_g((W_i \odot (\mathbf{e}m^T))x_t + (U_i \odot (mm^T))h_{t-1} + b_i)$$
$$o_t = \sigma_g((W_o \odot (\mathbf{e}m^T))x_t + (U_o \odot (mm^T))h_{t-1} + b_o)$$
$$\tilde{c}_t = \sigma_h((W_c \odot (\mathbf{e}m^T))x_t + (U_c \odot (mm^T))h_{t-1} + b_c)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad h_t = o_t \odot \sigma_h(c_t) \quad s.t. \quad m \in \{0, 1\}^{l_h}, \mathbf{e} = 1^{l_h}, \tag{2}$$

where σ_g is the sigmoid function, $\odot$ denotes element-wise multiplication and σ_h is the hyperbolic tangent function. x_t denotes the input vector at the time-step t, h_t denotes the current hidden state, and c_t denotes the long-term memory cell state. W_f, W_i, W_o, W_c denote the input-to-hidden weight matrices and U_f, U_i, U_o, U_c denote the hidden-to-hidden weight matrices. m is shared across all the gates to control the sparsity of hidden neurons.

We can optimize the trade-off between model performance and structured sparsification by considering the training objective

$$\min_{w,m} \quad L_E(f(x; w \odot m)) + \lambda \|m\|_0, \tag{3}$$

where f can be the operation of convolutional layers in Eq. 1 or LSTM cells in Eq. 2 with trainable weights, $w \odot m$ is a general expression of structured sparsified weight matrices in our proposed space and L_E corresponds to a loss function. (*e.g.* cross-entropy loss for classification), the ℓ_0 term penalizes the number of non-zero mask values thus encouraging sparsity, λ is a trade-off parameter between L_E and the ℓ_0 penalty. In the growing and pruning space, a model is optimal if it can minimize the combined cost of the description of model complexity $\|m\|_0$ and the loss L_E between the model and the data. However, optimizing $\|m\|_0$ is computationally intractable due to the combinatorial nature of binary states.

---

[1]The proposed growing space can be readily applied to the compression of GRUs [2] and vanilla RNNs.

3.2 Continuous Relaxation and Optimization

Learning by Continuation: To make the search space continuous and the optimization feasible, we adopt the framework proposed in [43], used to derive Continuous Sparsification.

First, we reparameterize m as the binary sign of a continuous variable s:

$$\text{sign}(s) = \begin{cases} 1, & \text{if} \quad s > 0 \\ 0, & \text{if} \quad s < 0 \end{cases}, \tag{4}$$

and rewrite the objective in Eq. 3 as

$$\min_{w, s \neq 0} \quad L_E(f(x; w \odot \text{sign}(s))) + \lambda \left\| \text{sign}(s) \right\|_1 . \tag{5}$$

Following [43], we attack the hard and discontinuous optimization problem in Eq. 5 by starting with an *easier* objective which becomes *harder* as the training proceeds. As in [43], we use a sequence of functions whose limit is the sign operation. Instead of using the sigmoid activation function, we adopt the hard sigmoid function[2] $\sigma(\beta s) = \min(\max((\beta s) + 0.5, 0), 1)$. Similar to the sigmoid function, we have that for any $s \neq 0$, $\lim_{\beta \to \infty} \sigma(\beta s) = \text{sign}(s)$, where $\beta > 0$ is a bandwidth parameter.

Auxiliary Discrete Variable: Using continuation methods, we can express our final objective as:

$$\min_{w, s \neq 0} \quad L_E(f(x; w \odot \sigma(\beta s) \odot q)) + \lambda \left\| \sigma(\beta s) \right\|_1 , \tag{6}$$

where q is sampled from $\text{Bern}(\sigma(\beta s))$. By increasing β, $\sigma(\beta s)$ becomes harder to optimize while the objectives converges to original discrete one. Different from [43], we introduce an 0-1 sampled auxiliary variable q based on the probability value $\sigma(\beta s)$. Thus we (1) effectively reduce training computational cost since any train-time architecture is sampled as a structured sparse one; (2) avoid using a suboptimal thresholding criterion to generate the inference architecture at the end of training.

Bandwidth Scheduler: We start training deep networks using Eq. 6 with $\sigma(\beta s)$, where the initial β value is set as $\beta_0 = 1$. We adapt its bandwidth β to control the optimization difficulty by instantiating a bandwidth scheduler in two ways: *globally* and *structure-wise separately*. A global bandwidth scheduler is called at the end of each training epoch and updates β on all activation functions σ following

$$\beta = \min(\beta_0 \cdot (1 + \gamma \cdot \text{n\_iters})^p, \beta_{\max}) , \tag{7}$$

where β_0 is the initial bandwidth which is set as 1, n_iters is the number of training iterations so far, $\beta_{\max}$ is used to constrain β to a certain range. In our experiments, we set $\beta_{\max}$ as 100. Constants γ and p are hyperparameters that govern the increasing speed of the bandwidth during the progressive training procedure. Note that such adaptive control system can be customized for different resource requirements (*e.g.* training computation cost) by tuning γ and p. A structure-wise separate bandwidth scheduler requires specifying one additional step: for each mask variable, instead of using a global counter *n_iters*, we set a separate counter *n_sampled_iters* which is increased only when its associated mask value is sampled as 1 in Eq. 6. Similarly we instantiate this scheduler with

$$\beta = \min(\beta_0 \cdot (1 + \gamma \cdot \text{n\_sampled\_iters})^p, \beta_{\max}) \tag{8}$$

Intuitively, the structure-wise separate bandwidth scheduler is more compelling because it allows bandwidth to increase at different rates for different mask variables: those more frequently sampled masks, indicative of a higher probability not to be pruned, will become more stable due to the higher optimization difficulty; Those less sampled masks at early stages may still have the chance to be grown under a relatively lower β. However, the global scheduler may fail to handle such cases. In our experiments, we report performance using the structure-wise scheduler and we also conduct investigation on the two alternatives during training.

In summary, Algorithm 1 shows full details of our optimization procedure with the structure-wise separate bandwidth scheduler.

---

[2]Note that the original hard sigmoid function is defined as $\min(1, \max(0, s))$ in [34]

Algorithm 1 : Optimization

Input: Training set and label set: $\boldsymbol{X} = (\boldsymbol{x}_i)_{i=1}^n$, $\boldsymbol{Y} = (\boldsymbol{y}_i)_{i=1}^n$
Output: Target efficient model S
Initialize: w as random weights and s as 0 in S; γ, p as float constants, $\beta_0, \beta_{\max}$ as 1, 100, update
interval $T > 1$, n_sampled_iters as all 1 vectors associated with each σ function.
for $r = 1$ **to** R **do**
     Sample random mini-batch x_i, y_i from $\boldsymbol{X}, \boldsymbol{Y}$
     Sample $q \sim \text{Bern}(\sigma(\beta s))$ and record the index idx where q value is 1.
     Train w and m using Eq. 6 with SGD.
     Update n_sampled_iters$[idx] += 1$
     if $(r \% T == 0)$ **then**
         Update β using Eq. 8
     end if
end for
return S
---

4 Experiments

4.1 Experimental Setup

Datasets: Evaluation is conducted on various tasks to demonstrate the effectiveness of our proposed method. For image classification, we use CIFAR-10 [25] and ImageNet [4]: CIFAR-10 consists of 60,000 images of 10 classes, with 6,000 images per class. The train and test sets contain 50,000 and 10,000 images respectively. ImageNet is a large dataset for visual recognition which contains over 1.2M images in the training set and 50K images in the validation set covering 1,000 categories. For semantic segmentation, we use the PASCAL VOC 2012 [5] benchmark which contains 20 foreground object classes and one background class. The original dataset contains 1,464 (train), 1,449 (val), and 1,456 (test) pixel-level labeled images for training, validation, and testing, respectively. The dataset is augmented by the extra annotations provided by [13], resulting in 10,582 training images. For language modeling, we use the word level Penn Treeban (PTB) dataset [37] which consists of 929k training words, 73k validation words and 82k test words with 10,000 unique words in its vocabulary.

Unpruned Baseline Models: For CIFAR-10, we use VGG-16 [45] with BatchNorm [24], ResNet-20 [14] and WideResNet-28-10 [48] as baselines. We adopt a standard data augmentation scheme (shifting/mirroring) following [21, 30], and normalize the input data with channel means and standard deviations. Note that we use the CIFAR version of ResNet-20, VGG-16, and WideResNet-28-10. VGG-16, ResNet-20, and WideResNet-28-10 are trained for 160, 160 and 200 epochs with a batch size of 128 and initial learning rate of 0.1. For VGG-16 and ResNet-20 we divide learning rate by 10 at epochs 80 and 120 and set the weights decay and momentum as 10^{-4} and 0.9. For WideResNet-28-10, the learning rate is divided by 5 at epochs 60, 120 and 160; the weight decay and momentum are set to 5×10^{-4} and 0.9. For ImageNet, we train the baseline ResNet-50 and MobileNetV1 model following the respective papers. We adopt the same data augmentation scheme as in [9] and report top-1 validation accuracy. For semantic segmentation, the performance is measured in terms of pixel intersection-over-union (IOU) averaged across the 21 classes (mIOU). We use Deeplab-v3-ResNet-101[3] [1] as the baseline model following the training details in [1]. For language modeling, we use vanilla two-layer stacked LSTM [49] as a baseline. The dropout keep ratio is 0.35 for the baseline model. The vocabulary size, embedding size, and hidden size of the stacked LSTMs are set as 10,000, 1,500, and 1,500, respectively, which is consistent with the settings in [49].

Implementation Details: There are two kinds of trainable variables in our method, denoted as model weights and mask weights. As a one-shot method, for model weights, we adopt the same hyperparameters with the corresponding unpruned baseline models, except that dropout keep ratio for language modeling is set as 0.5. For mask variables, we initialize the weights as 0 and use SGD training with initial learning rate of 0.1, weight decay of 0 and momentum of 0.9 on all datasets. The learning rate scheduler is the same with its corresponding model weights. The trade-off parameter λ is set as 0.01 on classification and semantic segmentation tasks, and 0.1 for language modeling tasks. For the bandwidth scheduler, we report model performance trained with structure-wise separate

---
[3]https://github.com/chenxi116/DeepLabv3.pytorch

Table 1: Overview of the pruning performance of each algorithm for various CNN architectures on CIFAR-10. For each algorithm and network architecture, the table reports the retained params and ratio (Params, M, %), and retained FLOPs ratio (FLOPs, %) of pruned models.

Model	Method	Val Acc(%)	Params(M)	FLOPs(%)
VGG-16 [45]	Original	92.9 (+0.0)	14.99 (100%)	100
	L1 [28]	91.8 (-1.1)	2.98 (19.9%)	19.9
	SoftNet [15]	92.1 (-0.8)	5.40 (36.0%)	36.1
	ThiNet [35]	90.8 (-2.1)	5.40 (36.0%)	36.1
	Provable [29]	92.4 (-0.5)	**0.85 (5.7%)**	**15.0**
	Ours	__92.9 (-0.0)__	1.50 (10.0%)	__16.5__
ResNet-20 [14]	Original	91.3 (+0.0)	0.27 (100%)	100
	L1 [28]	__90.9 (-0.4)__	0.15 (55.6%)	55.4
	SoftNet [15]	90.8 (-0.5)	0.14 (53.6%)	**50.6**
	ThiNet [35]	89.2 (-2.1)	0.18 (67.1%)	67.3
	Provable [29]	90.8 (-0.5)	**0.10 (37.3%)**	__54.5__
	Ours	**91.1 (-0.2)**	__0.11 (39.1%)__	59.8
WideResNet -28 -10 [48]	Original	96.2 (+0.0)	36.5 (100%)	100
	L1 [28]	__95.2 (-1.0)__	7.6 (20.8%)	49.5
	BAR(16x V) [27]	92.0 (-4.2)	**2.3 (6.3%)**	**1.5**
	Ours	**95.6 (-0.6)**	__2.6 (7.1%)__	__18.6__

Table 2: Overview of the pruning performance of each algorithm for various CNN architectures on ImageNet. For each algorithm and network architecture, the table reports the retained params and ratio (Params, M, %), and retained FLOPs ratio (FLOPs, %) of pruned models.

Model	Method	Top-1 Val Acc(%)	Params(M)	FLOPs(%)
ResNet-50 [14]	Original	76.1 (+0)	23.0 (100%)	100
	L1 [28]	74.7 (-1.4)	19.6 (85.2%)	77.5
	SoftNet [15]	74.6 (-1.5)	N/A	**58.2**
	Provable [29]	**75.2 (-0.9)**	**15.2 (65.9%)**	70.0
	Ours	**75.2 (-0.9)**	__16.5 (71.7%)__	__64.5__
MobileNetV1 (128) [18]	Original(25%)	45.1 (+0)	0.47 (100%)	100
	MorphNet [8]	__46.0 (+0.9)__	N/A	110
	Netadapt [47]	**46.3 (+1.2)**	N/A	__81__
	Ours	__46.0 (+0.9)__	0.41 (87.2%)	**70**

scheduler where (γ, p) are set as (0.0005, 0.7) for classification and segmentation models, and (0.0005, 1.2) for language modeling, respectively. All β_0 and $\beta_{\max}$ are set as 0 and 100. We also conduct parameter sensitivity analysis of sparsity and accuracy in terms of γ and p.

4.2 Results

VGG-16, ResNet-20, and WideResNet-28-10 on CIFAR-10: Table 1 shows the pruning results in terms of validation accuracy, retained parameters, and FLOPs of VGG-16, ResNet-20, and WideResNet-28-10 on CIFAR-10. We compare with various pruning algorithms that we implement and run alongside our algorithm. We can see that ours achieves either larger pruning ratio or less degradation in accuracy. Our pruned VGG-16 and ResNet-20 can achieve comparable parameters and FLOPs reduction with recently proposed Provable [29] method while outperforming it by 0.5% and 0.3% in validation accuracy. For very aggressively pruned WideResNet-28-10, we observe that BAR [27] might not have enough capacity to achieve negligible accuracy drop even with the knowledge distillation [16] during the training process.

ResNet-50 and MobileNetV1 on ImageNet: To validate the effectiveness of the proposed method on large-scale datasets, we further prune the widely used ResNet-50 and MobileNetV1 (128 × 128 resolution) on ImageNet and compare the performance of our method to the results reported directly in the respective papers, as shown in Table 2. In MobileNetV1 experiments, following the same setting with Netadapt [47], we apply our method on MobileNetV1 with 0.5 multiplier while setting

Table 3: Results on the PASCAL VOC dataset, showing mIOU, retained parameters and ratio (M, %), and retained FLOPs ratio (%).

Model	Method	mIOU	Params(M)	FLOPs(%)
Deeplab	Original	76.5 (+0)	58.0 (100%)	100
-v3	L1 [28]	75.1 (-1.4)	45.7 (78.8%)	62.5
-ResNet-101 [1]	Ours	**76.2 (-0.3)**	**33.8 (58.2%)**	**45.5**

Table 4: Results on the PTB dataset, showing validation and test perplexity, retained parameters and ratio (M, %), final sparse lstm structure, and retained FLOPs ratio (%).

Method	Perplexity (val,test)	Final Structure	Weight(M)	FLOPs(%)
Original [49]	(82.57, 78.57)	(1500, 1500)	66M (100%)	100
ISS [46]	(82.59, **78.65**)	(373, 315)	21.8M (33.1%)	13.4
Ours	(**82.16**, 78.67)	**(319, 285)**	**20.9M (31.7%)**	**12.8**

the original model's multiplier as 0.25 for comparison. Note that 50%-MobileNetV1(128) is one of the most compact networks, and thus is more challenging to simplify than other larger networks. Our method can still generate a sparser MobileNetV1 model compared with competing methods.

Deeplab-v3-ResNet-101 on PASCAL VOC 2012: We also test the effectiveness of our proposed method on the semantic segmentation task by pruning the Deeplab-v3-ResNet-101 model on the PASCAL VOC 2012 dataset. We apply our method to both the ResNet-101 backbone and ASPP module. Compared to the baseline, our pruned network reduces the FLOPs by 54.5% and the parameters amount by 41.8% while approximately maintaining mIoU (76.5% to 76.2%). See Table 3.

2-Stacked-LSTMs on PTB: We compare our proposed method with ISS based on vanilla two-layer stacked LSTM. As shown in Table 4, our method finds very compact model structure, while achieving similar perplexity on both validation and test sets. To be specific, our method achieves a 3.2× model size reduction and 7.8× FLOPs reduction from baseline model. Note that for fair comparison, we only prune the LSTM structure while keeping the embedding layer unchanged, following the same setting with ISS. Our method can achieve more compact structure than ISS, further reducing the hidden units from (373, 315) to (319, 285). These improvements may be due to the fact that our method dynamically grows and prunes the hidden neurons towards a better trade-off between model complexity and performance than that of ISS, which simply uses the group lasso to penalize the norms of all groups collectively for compactness.

4.3 Analysis

Dynamic Train-time Cost: One advantage of our method over conventional pruning methods is that we effectively decrease the computational cost not only of inference but also of training via structured continuous sparsification. Figure 2 shows the dynamics of train-time layer-wise FLOPs and stage-wise retained filters ratios of VGG-16 and ResNet-20 on CIFAR-10, respectively. From Figure 2(b) and 2(d) we see that our method preserves more filters of earlier stage (1 and 2) in VGG-16 and earlier layers within each stage of ResNet-20. Also, the layer-wise final sparsity of ResNet-20 is more uniform due to the residual connections.

Ours as Structured Regularization: We investigate the value of our proposed automatic pruning method serving as a more efficient training method with structure regularization. We re-initialize the pruned ResNet-20 and two-layer stacked LSTM and re-train them from scratch on CIFAR-10 and PTB, respectively. Comparing with their reported pruned model performance, we notice a performance degradation on both ResNet-20 (accuracy 91.1% → 90.8% (-0.3)) and LSTM models (test perplexity 78.67 → 86.22 (+7.55)). Our method appears to have a positive effect in terms of regularization or optimization dynamics, which is lost if one attempts to directly train the final compact structure.

Parameter Sensitivity: We analyze the sparsity and performance sensitivity relative to the bandwidth scheduler (structure-wise separately) parameters. We measured the performance with respect to a combination of γ and p. Specifically, we measure the normalized parameters sparsity and validation accuracy of ResNet-20 on the CIFAR-10 dataset as shown in Figure 3. From Figure 3(a) we can
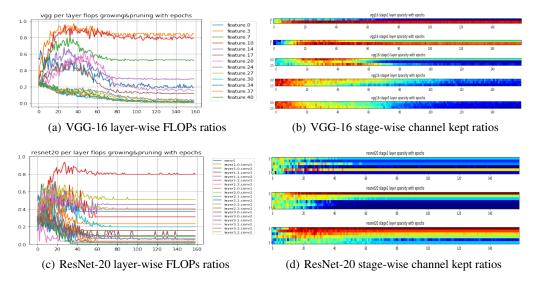
(a) VGG-16 layer-wise FLOPs ratios

(b) VGG-16 stage-wise channel kept ratios

(c) ResNet-20 layer-wise FLOPs ratios

(d) ResNet-20 stage-wise channel kept ratios

Figure 2: Track of train-time FLOPs and channel kept ratios.
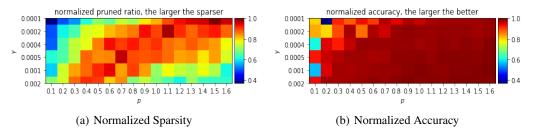

(a) Normalized Sparsity

(b) Normalized Accuracy

Figure 3: Parameter sensitivity of bandwidth scheduler hyperparameters γ and p.

acquire some knowledge on how to localize hyperparameters γ and p to achieve highly sparse networks. Figure 3(b) shows that when γ and p are located in relative large range (*e.g.* right-bottom), the validation accuracy is robust to changes in these hyperparameters.

Investigation on Bandwidth Scheduler: We investigate the effect of both global scheduler and structure-wise separate scheduler by conducting experiments on CIFAR-10 using VGG-16, ResNet-20, and WideResNet-28-10. The results using structure-wise separate scheduler are as reported in Table 1. For global scheduler, we note that to achieve similar sparsity, the pruned models suffer from accuracy drops of 0.5%, 0.2%, and 1.2%. With the global scheduler, optimization of all filters' masks stops at very early epochs and the following epochs of training are equivalent to directly training a stabilized compact structure. This may lock the network into a suboptimal structure, compared to our separate scheduler which dynamically grows and prunes over a longer duration.

5 Conclusion

In this paper, we propose a simple yet effective method to grow efficient deep networks via structured continuous sparsification, which decreases the computational cost not only of inference but also of training. The method is simple to implement and quick to execute, which aims at automating the network structure sparsification process for general purposes. The pruning results for widely used deep networks on various computer vision and language modeling tasks show that our method consistently generates smaller and more accurate networks compared to competing methods.

There are many interesting directions to be investigated further. For example, while our current sparsification process is designed with a generic objective, it would be interesting to incorporate model size and FLOPs constraints into training objective in order to target a particular resource. Additionally, our method's growing and pruning space is anchored to a given network topology. Future work could explore an architectural design space in which large subcomponents of the network are themselves candidates for pruning.

References

[1] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv:1706.05587*, 2017.

[2] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[3] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

[5] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The PASCAL visual object classes challenge: A retrospective. *IJCV*, 2015.

[6] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

[7] Ross B. Girshick. Fast R-CNN. In *ICCV*, 2015.

[8] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. MorphNet: Fast & simple resource-constrained structure learning of deep networks. In *CVPR*, 2018.

[9] Sam Gross and Michael Wilber. Training and investigating residual nets. *http://torch.ch/blog/2016/02/04/resnets.html*, 2016.

[10] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient DNNs. In *NIPS*, 2016.

[11] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *ICLR*, 2016.

[12] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *NIPS*, 2015.

[13] Bharath Hariharan, Pablo Arbelaez, Lubomir D. Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *ICCV*, 2011.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[15] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *IJCAI*, 2018.

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

[17] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[18] 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:1704.04861*, 2017.

[19] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q. Weinberger. CondenseNet: An efficient DenseNet using learned group convolutions. *CVPR*, 2018.

[20] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[21] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

[22] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NIPS*, 2016.

[23] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *arXiv:1602.07360*, 2016.

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[25] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The CIFAR-10 dataset. *http://www.cs.toronto.edu/~kriz/cifar.html*, 2014.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[27] Carl Lemaire, Andrew Achkar, and Pierre-Marc Jodoin. Structured pruning of neural networks with budget-aware regularization. In *CVPR*, 2019.

[28] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient ConvNets. In *ICLR*, 2017.

[29] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *ICLR*, 2020.

[30] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv:1312.4400*, 2013.

[31] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *ICLR*, 2019.

[32] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. In *ECCV*, 2016.

[33] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[34] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. *ICLR*, 2018.

[35] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. ThiNet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.

[36] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *NIPS*, 2018.

[37] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 1993.

[38] Dmitry Molchanov, Arsenii Ashukha, and Dmitry P. Vetrov. Variational dropout sparsifies deep neural networks. In *ICML*, 2017.

[39] Sharan Narang, Greg Diamos, Shubho Sengupta, and Erich Elsen. Exploring sparsity in recurrent neural networks. In *ICLR*, 2017.

[40] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[41] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[42] Pedro Savarese and Michael Maire. Learning implicitly recurrent CNNs through parameter sharing. In *ICLR*, 2019.

[43] Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. *arXiv:1912.04427*, 2019.

[44] Laurent Sifre and PS Mallat. *Rigid-motion scattering for image classification*. PhD thesis, Ecole Polytechnique, CMAP, 2014.

[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

[46] Wei Wen, Yuxiong He, Samyam Rajbhandari, Minjia Zhang, Wenhan Wang, Fang Liu, Bin Hu, Yiran Chen, and Hai Li. Learning intrinsic sparse structures within long short-term memory. In *ICLR*, 2018.

[47] Tien-Ju Yang, Andrew G. Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. NetAdapt: Platform-aware neural network adaptation for mobile applications. In *ECCV*, 2018.

[48] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[49] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv:1409.2329*, 2014.

[50] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. *CVPR*, 2018.

[51] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv:1611.01578*, 2016.

[52] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *CVPR*, 2018.