Learning Best Combination for Efficient N:M Sparsity

Anonymous Author(s)
Affiliation
Address
email

Abstract

By forcing at most N out of M consecutive weights to be non-zero, the recent N:M network sparsity has received increasing attention for its two attractive advantages: 1) Promising performance at a high sparsity. 2) Significant speedups on NVIDIA A100 GPUs. Recent studies require an expensive pre-training phase or a heavy dense-gradient computation. In this paper, we show that the N:M learning can be naturally characterized as a combinatorial problem that searches for the best combination candidate within a finite collection. Motivated by this characteristic, we solve N:M sparsity in an efficient divide-and-conquer manner. First, we divide the weight vector into $C_{\mathrm{M}}^{\mathrm{N}}$ combination subsets of a fixed size N. Then, we conquer the combinatorial problem by assigning each combination a learnable score that is jointly optimized with its associate weights. We prove that the introduced scoring mechanism can well model the relative importance between combination subsets. And by gradually removing low-scored subsets, N:M fine-grained sparsity can be efficiently optimized during the normal training phase. Comprehensive experiments demonstrate that our learning best combination (LBC) performs consistently better than off-the-shelf N:M sparsity methods across various networks. Our code is anonymously released at https://github.com/papercodesub/LBC.

1 Introduction

Recent years have witnessed the growing popularity of deep convolutional neural networks (CNNs) in a wide range of visual problems such as image classification [33, 12], object detection [11, 8] and semantic segmentation [9, 3]. However, the remarkable performance comes at the cost of huge computation burdens and memory footprint, resulting in great challenges for deploying CNNs in resource-limited devices. With the popularization of mobile devices, model compression has received extensive attention from both academia and industries. By reducing the number of parameters in CNNs, network sparsity has become one of the most representative techniques to alleviate storage and computation burdens. According to the sparsity granularity, traditional methods can be divided into unstructured sparsity [18, 6, 10] and structured sparsity [20, 28, 26]. As shown in Fig. 1(a), unstructured sparsity removes arbitrary weights in CNNs. Dozens of earlier studies [22, 17, 7] have demonstrated that unstructured sparsity is able to reach negligible performance degradation under a large compression rate. Unfortunately, unstructured sparsity often results in an irregular sparse matrix, which causes heavy index storage and very little acceleration. On the contrary, structured sparsity removes all weights in a filter as illustrated in Fig. 1 (b). As a result, the sparse weights are still in a hardware-friendly format, leading to notable speedups. Nevertheless, the removal of the whole filters also severely damages the accuracy performance. Thus, future progress is expected to find out a new sparsity pattern to tradeoff the performance and acceleration.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

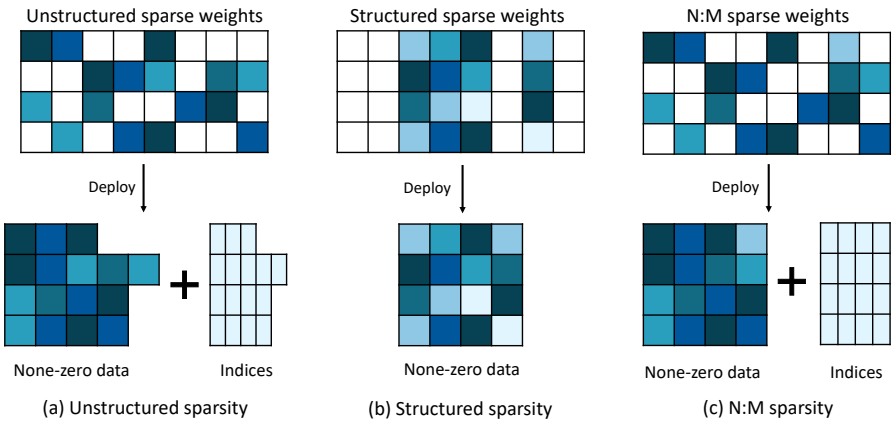

Figure 1: Mainstream sparsity granularity in the literature. (a) Unstructured sparsity removes individual weights at arbitrary locations. (b) Structured sparsity removes entire convolutional filters. (c) N:M sparsity (2:4 case) requires at most N out of M consecutive weights to be non-zero.

More recently, the N:M fine-grained sparsity has shown a promising power for the research community. As depicted in Fig. 1 (c), N:M sparsity divides the weights in a network into several blocks of size M. By preserving at most N out of M weights in each block, it not only achieves a high sparsity rate but also retains a regular sparse matrix structure for an effective acceleration supported by the NVIDIA Ampere Sparse Tensor Core [32]. The pioneering ASP (apex's automatic sparsity) [29] preserves two largest-magnitude weights in every four-element block and then re-trains the sparse network on the observed dataset. However, the time-consuming pre-training phase greatly prohibits the deployment of N:M sparsity in resource-constrained scenarios. To avert the pre-training burden, Zhou *et al.* [39] proposed to learn N:M sparsity from scratch by leveraging the straight-through estimator (STE) [1] to approximate the gradients of removed weights. Although an additional regularization is proposed to alleviate the weight error accumulation, dense gradients are required to be computed in each training iteration to update both the removed and preserved weights. Consequently, a heavy training burden is unavoidable, barricading the generalization of N:M sparsity to gradient and activation as analyzed by Chmiel *et al.* [2]. Therefore, how to realize an efficient N:M sparsity remains unexplored so far.

In this paper, we propose to learn the best combination in order to reach an efficient N:M sparsity (LBC). Our LBC is motivated by the fact that the N:M learning can be modeled as a combinatorial problem which consists in finding, among a finite collection of combinations, one that satisfies all the given conditions. Considering a single block, the purpose of N:M sparsity is to select N non-repeated elements as a combination from the M continuous weights. Thus, N:M sparsity can be simplified as finding the best combination candidate from a total of $C_{\mathrm{M}}^{\mathrm{N}}$ combinations on condition that the training loss can be minimized on the observed dataset. Given this analysis, our LBC solves the N:M sparsity from the perspective of combinatorial problems in an efficient divide-and conquer manner. Specifically, we first divide the weight block into a collection of $C_{\mathrm{M}}^{\mathrm{N}}$ combinations with a fixed size N. And then, our LBC aims to learn the best combination from this collection. To conquer this issue, we introduce a learnable score variable, which, as we formally prove in Sec. 3.4, can well tell the relative importance between different combinations. Consequently, the best combination can be located by gradually removing the low-scored combinations during a normal training process without the necessity of a dense-gradient computation.

Extensive experiments on the large ImageNet dataset demonstrate the efficacy of our LBC. For instance, our LBC achieves 77.2% in the top-1 accuracy for training a 2:4 sparse ResNet-50 on ImageNet, which takes the leading position in comparison with existing N:M sparsity methods [29, 39]. It is also worth mentioning that our LBC even surpasses some state-of-the-art unstructured sparsity methods that fail to achieve speedups. For example, under an extremely high sparsity rate of around 95%, our LBC still achieves a top-1 accuracy of 71.8% with a 1:16 sparse pattern, surpassing the recent competitor STR [17] by 1.2%.

2 Related Work

2.1 Network Sparsity

Network sparsity has become an active research topic ever since the last decade [18, 10, 24]. Traditional methods can be roughly categorized into unstructured sparsity and structured sparsity. The former category removes individual weights at any position of the network. Prevailing studies remove unimportant weights using heuristic criteria such as gradient [19], momentum [6], and magnitude [10]. Recent advances implement unstructured sparsity in a training adaptive manner. RigL [7] alternately removes and revives weights according to their magnitudes and dense gradients. Ding *et al.* [6] gradually zeroed out the redundant weights by categorizing CNN weights into two parts at each training iteration and updating them using different rules. Despite the progress, the irregular sparse weight tensors rarely support speedups on common hardware unless the sparse rate increases to 95% or even higher [36, 32]. On the other hand, structured sparsity gains noticeable speedups thanks to its coarse-grained removal of the entire filters. Two issues are widely discussed in existing studies including the pruned network structure and the filter importance measurement. Typical works solve the former issue by rule-of-thumb designs [13, 5] or layer-wise sparse ratio searching based on evolutionary algorithms [21, 23]. As for the second issue, most works resort to devising a certain criterion to estimate the filter importance including output activation [38], ranks of feature map [20], geometric median of filters [14], *etc*.

2.2 N:M Sparsity

The N:M fine-grained sparsity [29, 39, 34] advocates N-out-of-M non-zero sparse tensors in the input channel dimension. Supported by the NVIDIA Ampere Core [32], N:M sparsity leads to attractive storage and computation efficiency. For example, N:M sparsity can achieve 2x speedups on the NVIDIA A100 GPU in the 2:4 case, while unstructured sparsity even decreases the inference speed under a similar sparse rate. NVIDIA [29] follows a traditional three-step pipeline to implement N:M sparsity, unfolded as pre-training, pruning and fine-tuning. Pool *et al.* [31] further leveraged channel permutations to maximize the accuracy of N:M sparse networks. Sun *et al.* [34] proposed a layerwise fine-grained N:M scheme to achieve higher accuracy than the uniform N:M sparsity. Recently, there are also efforts in exploring the training efficiency of N:M fine-grained sparsity. Hubara *et al.* [15] devised transposable masks to accelerate the backward phase of N:M sparsity training. Chmiel *et al.* [2] further extended N:M sparsity to activation, which further boosts the training efficiency. Despite the well-retained performance [29], the expensive pre-training process still hinders the deployment of N:M sparsity. Sparse-refined straight-through estimator (SR-STE) [39] learns the N:M sparsity from scratch. Detailedly, the removed weights are also revived during the backward phase using the straight-through estimator [1] and a specially-designed sparse penalty item. Therefore, by choosing N-out-of-M weights with the largest magnitudes in each forward propagation, the N:M sparsity can be obtained in a dynamic and end-to-end training manner. However, it requires dense gradients in order to revive the weights, thus can not be combined with these studies on training efficiency [15, 2].

3 Methodology

3.1 Preliminaries

Denoting the parameters of an L-layer convolutional network as $\mathbf{W} = \{\mathbf{W}^0, \mathbf{W}^1, ..., \mathbf{W}^{\mathrm{L}}\}$ where $\mathbf{W}^l$ is a K^l-dimension vector[1], N:M fine-grained sparsity divides parameters in each layer into groups along the input channel dimension, leading to each group containing M consecutive weights. It saves computation and retains performance by requiring at most N out of M consecutive parameters to be non-zero values. To this end, the $\mathbf{W}^l$ is firstly divided into $\mathrm{G}^l = \frac{\mathrm{K}^l}{\mathrm{M}}$ groups. For ease of the following representation, we reorganize $\mathbf{W}^l$ as $\mathbf{W}^l \in \mathbb{R}^{\mathrm{G}^l \times \mathrm{M}}$. And then, the optimization of N:M sparsity can

[1]Batch norm layers and bias items are omitted here for simplicity.

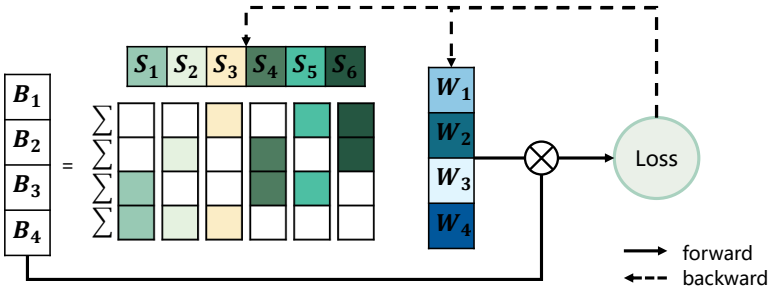

Figure 2: The training process for learning the best combination candidate of N:M sparsity (2:4 case).

be formulated as:

$$\min_{\mathbf{W}} \mathcal{L}(\mathbf{W};\ \mathcal{D}) \quad s.t. \quad \|\mathbf{W}^l_{g,:}\|_0 = \mathrm{N}, \tag{1}$$

where $l = 1, 2, ..., \mathrm{L}$ and $g = 1, 2, ..., \mathrm{G}^l$. The $\mathcal{L}(\cdot)$ denotes training loss function and $\mathcal{D}$ represents the observed dataset. An illustrative case of 2:4 sparsity is given in Fig. 1(c).

Previous N:M methods [29, 39] follow the traditional unstructured sparsity [10] to remove weights in a global manner by looking at the weight magnitude. However, they all require huge training costs either from a pre-training phase [29] or dense-gradient calculation [39] to select the most important weights from the large number of parameters in deep networks. Hence, how to achieve N:M sparsity in a more efficient manner remains an open issue so far.

3.2 Combinatorial Problem in N:M Sparsity

Fortunately, by making full use of the intrinsic particularity in N:M sparsity, the solution space can be well reduced to pursue an efficient N:M sparsity. Concretely, given a weight group $\mathbf{W}^l_{g,:}$, the target of N:M sparsity is to select N out of the M weights in $\mathbf{W}^l_{g,:} \in \mathbb{R}^{\mathrm{M}}$ and discard the remaining M$-$N ones. This leads to a combination collection $\Theta^l_g = \{\Theta^l_{g,i}\}_{i=1}^{C_{\mathrm{M}}^{\mathrm{N}}}$ where each combination $\Theta^l_{g,i}$ is a subset of $\mathbf{W}^l_{g,:}$ and consists of N non-repeated elements from $\mathbf{W}^l_{g,:}$. It is intuitive to derive the collection size $|\Theta^l_g| = C_{\mathrm{M}}^{\mathrm{N}}$. As a toy example, we have $C_4^2 = 6$ considering the 2:4 case. Also, we have:

$$\Theta^l_g = \{ \underbrace{\{\mathbf{W}^l_{g,0}, \mathbf{W}^l_{g,1}\}}_{\Theta^l_{g,1}}, \underbrace{\{\{\mathbf{W}^l_{g,0}, \mathbf{W}^l_{g,2}\}}_{\Theta^l_{g,2}}, \underbrace{\{\mathbf{W}^l_{g,0}, \mathbf{W}^l_{g,3}\}}_{\Theta^l_{g,3}}, \underbrace{\{\mathbf{W}^l_{g,1}, \mathbf{W}^l_{g,2}\}}_{\Theta^l_{g,4}}, \underbrace{\{\mathbf{W}^l_{g,1}, \mathbf{W}^l_{g,3}\}}_{\Theta^l_{g,5}}, \underbrace{\{\mathbf{W}^l_{g,2}, \mathbf{W}^l_{g,3}\}}_{\Theta^l_{g,6}} \}. \tag{2}$$

According to the above analyses, we realize that the N:M sparsity can be naturally characterized as a *combinatorial problem*, which aims to, given a finite collection of combinations Θ^l_g, search for the best candidate within this collection. Herein, we reformulate the realization of N:M sparsity from the perspective of a combinatorial problem as:

$$\underset{\Theta_{g,i} \in \mathbf{W}^l_{g,:}}{\arg\min}\ \mathcal{L}(\mathbf{W}^l_{g,:};\ \mathcal{D}),\ s.t.\ |\Theta_{g,i}| = \mathrm{N} \tag{3}$$

where $l = 1, 2, ..., \mathrm{L}$, $g = 1, 2, ..., \mathrm{G}^l$ and $i = 1, 2, ..., C_{\mathrm{M}}^{\mathrm{N}}$.

Discussions. Eq. (1) implements N:M sparsity by globally sparsifying a weight vector $W^l_{g,:}$. In contrast to earlier implementations, our redefined Eq. (3) solves N:M sparsity in a divide-and-conquer manner. We first divide the weight vector into $C_{\mathrm{M}}^{\mathrm{N}}$ subsets of a fixed size N. Then, we conquer the combinatorial problem by scoring each subset and finally picking up the highest-scored one, details of which are given in the following contents.

3.3 Learning Best Combination

Based on the above analyses, we further propose to learn the best combination for each weight vector $\mathbf{W}^l_{g,:}$. Following [39], we consider learning the N:M sparsity from scratch to avoid the heavy

dependency on an expensive pre-trained model. A learning example of the 2:4 case is shown in Fig. 2. Specifically, denoting the input feature map of the l-th convolution layer as $\mathbf{Z}^{l-1}$, the output feature map $\mathbf{Z}^l$ is calculated as:

$$\mathbf{Z}^l = \mathbf{Z}^{l-1} \otimes \mathbf{W}^l, \tag{4}$$

where $\otimes$ denotes the convolutional operation. The earlier study [39] chooses to zero out low-magnitude N weights for every weight vector $\mathbf{W}^l_{g,:}$ in the forward propagation. The straight-through estimator (STE) is used to dynamically remove and revive weights, which however leads to error accumulation on the weights. Besides, dense-gradient computation is of heavy cost since both removed and preserved weights are required to update. Given the combinatorial property, we are entitled to the opportunity to reach N:M sparsity by gradually removing the low-scored candidate subsets in the combination collection Θ^l_g during the training procedure until the one with the smallest loss can be found. The gradual sparsity is demonstrated to perform better in many traditional network sparsity [41, 40, 22]. To that effect, we consider the gradual pruning schedule proposed by Zhu *et al*. [41] to remove candidate subsets considered unimportant, and the remaining collection at the t-th training epoch is:

$$\Theta^l_{g,t} = \Theta^l_{g,t-1} - \bar{\Theta}^l_{g,t}, \tag{5}$$

where $\bar{\Theta}^l_{g,t} \in \Theta^l_{g,t-1}$ contains the candidates considered unimportant at the t-th training epoch and its size is defined as:

$$|\bar{\Theta}^l_{g,t}| = \begin{cases} 0, \text{ if } t = t_i, \\ \left\lceil (C^{\mathrm{N}}_{\mathrm{M}} - 1)\big(1 - (1 - \frac{t-t_i}{t_f - t_i})^3\big) \right\rceil - |\bar{\Theta}^l_{g,t-1}|, \text{ otherwise,} \end{cases} \tag{6}$$

where $t \in [t_i, t_f]$, t_i and t_f denote the beginning and ending of the training epochs in the gradual pruning schedule [41]. It is easy to know that when $t = t_f$, we remove a total of $C^{\mathrm{N}}_{\mathrm{M}} - 1$ candidate subsets. In other words, only one combination, which is considered to be the best candidate, is preserved in the end.

To measure the candidate's importance, we invent a learnable score matrix $\mathbf{S}^l \in \mathbb{R}^{\mathrm{G}^l \times C^{\mathrm{N}}_{\mathrm{M}}}$. In our design, each element $\mathbf{S}^l_{g,i}$ reflects relative importance of candidate $\Theta^l_{g,i}$ in comparison to others in Θ^l_g and we reduce the size of collection Θ^l_g by removing the low-scored candidates. In the forward propagation, an individual weight $\mathbf{W}^l_{g,i}$ often exists in multiple candidate subsets as shown in Eq. (2). Given this, we impose a binary mask $\mathbf{B}^l \in \{0,1\}^{\mathrm{G}^l \times \mathrm{M}}$ upon the weight $\mathbf{W}^l$ and the output in Eq. (4) becomes:

$$\mathbf{Z}^l = \mathbf{Z}^{l-1} \otimes (\mathbf{B}^l \odot \mathbf{W}^l), \tag{7}$$

where $\odot$ represents the element-wise multiplication. The zero elements in $\mathbf{B}^l$ indicate the removal of corresponding weights that contribute least to the network. In our design, the binary mask $\mathbf{B}^l$ is derived as:

$$\mathbf{B}^l_{g,i} = \mathbb{I}\Big(\big(\sum_j \mathbf{S}^l_{g,j} \cdot \mathbb{I}(\mathbf{W}^l_{g,j} \in \Theta^l_{g,i})\big) \neq 0\Big), \tag{8}$$

where $\mathbb{I}(\cdot)$ returns 1 if its input is true, and 0 otherwise. That is to say, we choose to preserve weights within any preserved candidate subset. Note that if $\mathbf{W}^l_{g,i}$ does not exist in any preserved candidate subset, it will be automatically removed since $\mathbf{B}^l_{g,i} = 0$ according to Eq. (8).

The use of a binary mask originated from many traditional unstructured sparsity methods [7, 16]. However, to our best knowledge, it is the first time to be introduced in N:M sparsity. Moreover, very different from these studies that directly optimize the binary mask, our mask is obtained according to the candidate subsets the corresponding weight falls into. Finally, since the derivative of the $\mathbb{I}(\cdot)$ function is zero almost everywhere, we adopt the STE [1] to update $\mathbf{S}^l_{g,j}$ as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{S}^l_{g,j}} = \frac{\partial \mathcal{L}}{\partial \mathbf{B}^l_{g,i}} \frac{\partial \mathbf{B}^l_{g,i}}{\partial \mathbf{S}^l_{g,j}} \approx \frac{\partial \mathcal{L}}{\partial \mathbf{B}^l_{g,i}}. \tag{9}$$

Consequently, the STE approximation of weight gradient [39] becomes the approximation of the gradient of the introduced score matrix, avoiding the error accumulation on the weights. Moreover,

Algorithm 1: Learning Best Candidate (LBC).

Require : Observed data $\mathcal{D}$; Randomly initialized weights $\mathbf{W}$; Loss function $\mathcal{L}$; Training epochs T; Initial and final epoch for removing combinations t_i, t_f; Candidate collection Θ.

Output : Trained weights $\mathbf{W}$; Binary mask $\mathbf{B}$

```
S^l_{g,i} ← 1 ∀l, ∀g, ∀i // Learnable Importance Score
for t ∈ [1, ..., T] do
    if |Θ̄^l_{g,t}| ≠ 1 then
        Calculate |Θ̄^l_{g,t}| via Eq. (6)
        for l ∈ [1, ..., L] do
            for g ∈ [1, ..., G^l] do
                Θ̄^l_{g,t} = Top-|Θ̄^l_{g,t}| smallest S^l_g in Θ^l_{g,t-1}
                Θ^l_{g,t} = Θ^l_{g,t-1} − Θ̄^l_{g,t} // Remove Unimportant Candidate Subsets
            end
        end
        Get B via Eq. (8)
    end
    Forward via Eq. (7)
    Update via the SGD optimizer
end
```

as verified in Sec. 3.4, the STE approximation endows our score matrix with the ability to measure the relative importance between different candidate subsets.

Our proposed algorithm for learning the best candidate, dubbed as LBC, is outlined in Alg. 1. Different from traditional methods [38, 25] that conduct network sparsity in a two-step manner including a mask learning and a weight tuning, LBC enables the optimization of $\mathbf{W}$ and $\mathbf{S}$ in a joint framework, leading to a well-optimized N:M sparsity during a normal training phase. It is worthy to highlight the efficiency of LBC since it casts off the dense gradient calculation in [39]. Also, LBC can seamlessly cooperate with techniques that extend N:M sparsity to gradient and activation to further boost the training efficiency [2, 15].

3.4 Tractability of Optimization

In this section, we take the candidate $(\mathbf{W}^l_{g,0}, \mathbf{W}^l_{g,1})$ in the 2:4 sparsity of Eq. (2) as an example, to show that our introduced mask score can well reflect the relative importance of each candidate subset. Our analysis can be extended to other N:M patterns as well. The gradient of the candidate score $\mathbf{S}^l_{g,1}$ is derived using STE [1] as:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{S}^l_{g,1}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{B}^l_{g,0}}\frac{\partial \mathbf{B}^l_{g,0}}{\partial \mathbf{S}^l_{g,1}} + \frac{\partial \mathcal{L}}{\partial \mathbf{B}^l_{g,1}}\frac{\partial \mathbf{B}^l_{g,1}}{\partial \mathbf{S}^l_{g,1}} \\ &= \mathbf{W}^l_{g,0}\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l_{g,0}}\frac{\partial \mathbf{B}^l_{g,0}}{\partial \mathbf{S}^l_{g,1}} + \mathbf{W}^l_{g,0}\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l_{g,0}}\frac{\partial \mathbf{B}^l_{g,1}}{\partial \mathbf{S}^l_{g,1}} \\ &\approx \mathbf{W}^l_{g,0}\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l_{g,0}} + \mathbf{W}^l_{g,1}\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l_{g,1}}.\end{aligned} \tag{10}$$

Here, $\mathbf{W}^l_{g,0}\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l_{g,0}}$ meets the well-known first-order based network pruning criteria [27] that estimates the loss changes after removing $\mathbf{W}^l_{g,0}$ as:

$$\begin{aligned}\Delta\mathcal{L}(\mathbf{W}^l_{g,0}) &= \mathcal{L}(\mathbf{W}^l_{g,0} = 0) - \mathcal{L}(\mathbf{W}^l_{g,0}) \\ &\approx \mathcal{L}(\mathbf{W}^l_{g,0}) - \mathbf{W}^l_{g,0}\frac{\partial \mathcal{L}}{\partial(\mathbf{W}^l_{g,0} = 0)} + R_1(\mathbf{W}^l_{g,0} = 0) - \mathcal{L}(\mathbf{W}^l_{g,0}) \\ &\approx -\mathbf{W}^l_{g,0}\frac{\partial \mathcal{L}}{\mathbf{W}^l_{g,0}}.\end{aligned} \tag{11}$$

Table 1: Experimental results of different methods for training the N:M sparse ResNet-18, ResNet-50 and DeiT-small. * implies our reproduced results.

Model	Method	N:M Pattern	Top-1 Accuracy (%)	Epochs (Train)	FLOPs (Train)
ResNet-18	Baseline	-	70.9	120	1x(1.4e18)
ResNet-18	ASP	2:4	69.9	200	1.24x
ResNet-18	SR-STE	2:4	71.2	120	0.83x
ResNet-18	**LBC**	2:4	**71.5**	120	**0.70x**
ResNet-50	Baseline	-	77.1	120	1x(3.2e18)
ResNet-50	ASP	2:4	76.8	200	1.24x
ResNet-50	SR-STE	2:4	77.0	120	0.83x
ResNet-50	**LBC**	2:4	**77.2**	120	**0.72x**
ResNet-50	SR-STE	1:4	75.3	120	0.74x
ResNet-50	**LBC**	1:4	**75.9**	120	**0.39x**
ResNet-50	SR-STE	2:8	76.2	120	0.74x
ResNet-50	**LBC**	2:8	**76.5**	120	**0.41x**
ResNet-50	SR-STE	1:16	71.5	120	0.69x
ResNet-50	**LBC**	1:16	**71.8**	120	**0.28x**
DeiT-small	Baseline	-	79.8	300	1x(8.9e18)
DeiT-small	SR-STE*	2:4	79.6	300	0.83x
DeiT-small	**LBC**	2:4	**80.1**	300	**0.71x**

Consequently, Eq. (10) becomes:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{S}_{g,1}^{l}} \;=\; -(\Delta\mathcal{L}(\mathbf{W}_{g,0}^{l}) + \Delta\mathcal{L}(\mathbf{W}_{g,1}^{l})). \tag{12}$$

We use the stochastic gradient descent (SGD) optimizer for training and the updating rule of $\mathbf{S}_{g,j}^{l}$ can be finally represented as:

$$\mathbf{S}_{g,1}^{l} \;=\; \mathbf{S}_{g,1}^{l} + (\Delta\mathcal{L}(\mathbf{W}_{g,0}^{l}) + \Delta\mathcal{L}(\mathbf{W}_{g,1}^{l})), \tag{13}$$

where learning rate and momentum are ignored here for simplicity. As can be seen, the candidate score $\mathbf{S}_{g,1}^{l}$ is the accumulation of loss change for removing weights in the corresponding candidate. A small score denotes less loss increase thus the corresponding candidate can be safely removed. Therefore, our introduced score matrix can be used to measure the relative importance between candidate subsets, and preserving the highest-scored combination well solves the combinatorial problem of Eq. (3) in N:M sparsity.

4 Experiments

4.1 Experiment Settings and Implementation Details

In this section, we evaluate our LBC on the challenging ImageNet [4] using the representative networks including ResNet-18, ResNet-50 [12] and DeiT [35]. We implement LBC using the PyTorch [30] framework across 2 NVIDIA Tesla A100 GPUs. Following Zhou *et al.* [39], we train the ResNets for 120 epochs with an initial learning rate of 0, which is linearly increased to 0.1 during the first 5 epochs and then decayed to 0 scheduled by the cosine annealing. The SGD optimizer is adopted for updating parameters with the weight decay and momentum setting as 0.0005 and 0.9, respectively. Besides, we adopt the Timm framework [37] to train DeiT with 300 epochs. For the implementation of the proposed LBC, we set t_i to 0 and t_f to 1/2 of the total training epochs. Top-1 classification accuracy, training epochs and training FLOPs are reported for all methods under different sparse patterns.

This section is organized as follows. In Sec. 4.2, we compare our LBC with existing N:M sparsity methods including ASP [32] and SR-STE [39] under different N:M patterns. In Sec. 4.3, we further

Table 2: Results of the N:M and unstructured sparsity methods for sparsifying ResNet-50.

Method	Top-1 Accuracy (%)	Sparsity (%)	Params (Test)	FLOPs (Test)	Structured
Baseline	77.3	0.0	25.6M	1x (4.1G)	✓
DNW	68.3	95	1.28M	0.20G	✗
RigL	70.0	95	1.28M	0.49G	✗
GMP	70.6	95	1.28M	0.20G	✗
STR	70.4	95	1.27M	0.16G	✗
SR-STE	71.5	94 (1:16)	3.52M	0.44G	✓
LBC	**71.8**	94 (1:16)	3.52M	0.44G	✓

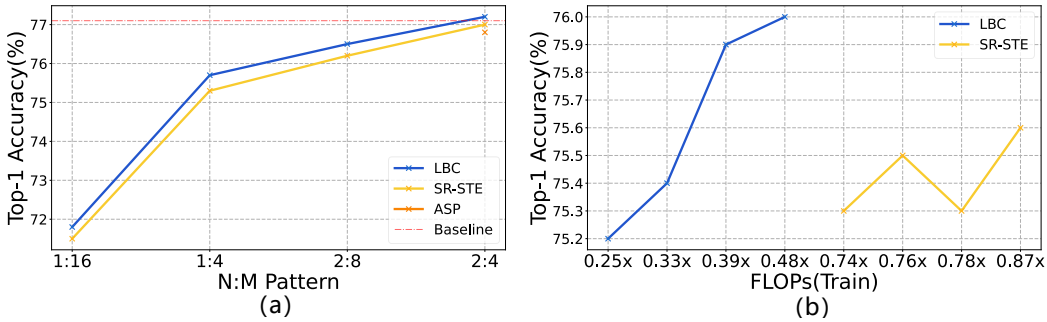

Figure 3: (a) Comparison results under different N:M pattern for sparsifying ResNet-50. (b) Top-1 accuracy *v.s.* training FLOPs for SR-STE and our proposed LBC at 1:4 case.

show the performance of LBC and traditional unstructured sparsity methods including RigL [7], GMP [41], and STR [17] under similar total compression rates. At last, we give efficiency analyses of LBC with respect to its components in Sec. 4.4.

4.2 Comparison with N:M sparsity methods

ResNet. We first apply our LBC to train N:M sparse ResNet [12] with different depths of 18 and 50. As can be referred to Tab. 1, the proposed LBC takes the lead in all N:M patterns and networks with even less training burden. For ResNet-18, LBC outperforms SR-STE [39] at 2:4 pattern at the Top-1 classification accuracy by 0.3% (71.2% for SR-STE and 71.5% for LBC). For ResNet-50, LBC again demonstrates its superiority against both ASP [29] and SR-STE for all sparse patterns. Specifically, ASP achieves a Top-1 accuracy of 76.8% for training 2:4 sparse ResNet-50 using 200 training epochs that contain pre-training and fine-tuning phase, while LBC obtains a significantly higher accuracy of 77.2% with a fewer epoch of 120. When it comes to 1:4 sparse case, LBC surpasses SR-STE [39] by a large margin (75.9% *v.s.* 75.3% in Top-1 accuracy and 0.39x *v.s.* 0.74x in training FLOPs). Although SR-STE also alleviates the pre-training process, the dense backward propagation greatly decreases its training efficiency in comparison with LBC. As can be referred from Fig.3 (a), LBC is still advantageous in all aspects when compared with SR-STE in the 1:4 case. Hence, the superiority of the proposed idea for learning the best combination in N:M sparsity is evident.

DeiT-small. We further conduct experiments on the recent advanced Vision Transformer (ViT) model DeiT [35] to validate the efficacy of LBC. Similar to the strategy for compressing CNNs, we divide all parameters including the multi-head attention and feed-forward module in DeiT-small and train the N:M sparse networks from scratch. Tab. 1 shows the results of LBC and our re-produced SR-STE for training 2:4 sparse DeiT on ImageNet. As can be seen, LBC again surpasses SR-STE by a noticeable margin of 0.5% (80.1% for LBC and 79.6% for SR-STE) using fewer training FLOPs (0.71x for LBC and 0.83x for SR-STE). Therefore, the ability of LBC is well validated for compressing ViT models.

4.3 Comparison with unstructured sparsity methods

We also compare the performance of LBC with state-of-the-art unstructured sparsity methods in Tab. 2. By training 1:16 sparse ResNet-50 on ImageNet, LBC achieves a Top-1 accuracy of 71.8%,

with 1.4% and 1.2% improvements over STR and GMP, respectively. Furthermore, the compressed models generated by unstructured sparsity remain an irregular format, which brings great challenges for the real-application deployment. Nevertheless, under a similar sparse rate, our LBC achieves practical compression and acceleration thanks to the NVIDIA Ampere Core while still bringing higher accuracy performance, which validates the effectiveness and advantages of exploring N:M sparsity.

4.4 Efficiency analyses

In this part, we investigate the property of LBC for improving the N:M sparse training efficiency. All the experimental results are conducted on ImageNet by training 2:4 sparse ResNet-50. We first point out that the total training FLOPs of LBC are proportional to the initial epoch t_i and final epoch t_f. To explain, larger t_i and t_f indicate a more smooth curve for removing combinations and more weights can be sufficiently trained before the epoch reaches t_f. For a fair comparison, we also apply the same gradual sparsity schedule to SR-STE [39]. As illustrated in Tab. 3, more training FLOPs consistently improve the performance of LBC as the importance score can be jointly trained with weights more sufficiently. Fig. 3 (b) further shows that our LBC takes the lead in the trade-off between training cost and Top-1 accuracy performance when compared with SR-STE [29, 39]. It is worthwhile to note that, by selecting different t_i and t_f, LBC can provide a satisfying solution between training expenditure and performance under different resource constraints. Thus, the efficiency and applicability of LBC are significant.

Figure 4: Experimental results for SR-STE and LBC at the same gradual sparsity schedule (1:4 case, ResNet-50).

Method	t_i	t_f	Top-1 Accuracy (%)	FLOPs (Train)
SR-STE	0	0	75.3	0.74x
LBC	0	0	75.2	0.25x
SR-STE	0	30	75.5	0.76x
LBC	0	30	75.4	0.33x
SR-STE	0	60	75.3	0.78x
LBC	0	60	75.9	0.39x
SR-STE	30	60	75.6	0.87x
LBC	30	60	76.0	0.48x

5 Limitation

Firstly, LBC still needs relatively dense computation in the early stage of N:M training as there are still plenty of left candidate subsets. Thus, despite the advantages over approaches that rely on pre-trained models or dense-gradient calculation over the entire training phase, there are still room for improving the efficiency of LBC. Secondly, this paper only explores N:M sparsity on image classfication task. We expect to show more results on other tasks such as natural language processing (NLP) and object detection in our future work. Finally, we mainly focus on N:M sparsity on weights, while adding N:M sparsity for gradient [15] and activation [2] in the workflow of LBC remains to be excavated in the near future.

6 Conclusion

N:M fine-grained sparsity is an important technique that allows fast inference and training on the next-generation hardware and platforms. In this paper, we have presented LBC towards efficient N:M sparsity from scratch. We first transfer the N:M learning into a combinatorial problem, which motivates us to select the best weights combination from C_M^N candidates. Further, we propose to assign learnable importance scores for each combination, which is jointly optimized with the associate weights. We perform an in-depth analysis that the best combination can be identified by aggressively removing less important combinations. Extensive experiments have demonstrated the efficacy of LBC in reducing the N:M training cost, and its superiority over several SOTAs. Our LBC is characterized by end-to-end implementation, efficient training, the tractability of optimization, and state-of-the-art performance in sparsifying modern networks.

References

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

[2] Brian Chmiel, Itay Hubara, Ron Banner, and Daniel Soudry. Optimal fine-grained n: M sparsity for activations and neural gradients. *arXiv preprint arXiv:2203.10991*, 2022.

[3] Ioana Croitoru, Simion-Vlad Bogolin, and Marius Leordeanu. Unsupervised learning of foreground object segmentation. *International Journal of Computer Vision (IJCV)*, 127(9):1279–1302, 2019.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[5] Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan. Approximated oracle filter pruning for destructive cnn width optimization. In *International Conference on Machine Learning (ICML)*, pages 1607–1616. PMLR, 2019.

[6] Xiaohan Ding, Xiangxin Zhou, Yuchen Guo, Jungong Han, Ji Liu, et al. Global sparse momentum sgd for pruning very deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6382–6394, 2019.

[7] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning (ICML)*, pages 2943–2952, 2020.

[8] Ross Girshick. Fast r-cnn. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[9] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 580–587, 2014.

[10] 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 (NeurIPS)*, pages 1135–1143, 2015.

[11] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[13] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2234–2240, 2018.

[14] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4340–4349, 2019.

[15] Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Joseph Naor, and Daniel Soudry. Accelerated sparse neural training: A provable and efficient method to find n: m transposable masks. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021.

[16] Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 20744–20754, 2020.

[17] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning (ICML)*, pages 5544–5555, 2020.

[18] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 598–605, 1989.

[19] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations (ICLR)*, 2019.

[20] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1529–1538, 2020.

[21] Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, and Yonghong Tian. Channel pruning via automatic structure search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 673–679, 2020.

[22] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Zahra Atashgahi, Lu Yin, Huanyu Kou, Li Shen, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. Sparse training via boosting pruning plasticity with neuroregeneration. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[23] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Tim Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *IEEE International Conference on Computer Vision (ICCV)*, pages 3296–3305, 2019.

[24] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations (ICLR)*, 2017.

[25] Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *Pattern Recognition (PR)*, page 107461, 2020.

[26] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *IEEE International Conference on Computer Vision (ICCV)*, pages 5058–5066, 2017.

[27] 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.

[28] Xuefei Ning, Tianchen Zhao, Wenshuo Li, Peng Lei, Yu Wang, and Huazhong Yang. Dsa: More efficient budgeted pruning via differentiable sparsity allocation. In *European Conference on Computer Vision (ECCV)*, pages 592–607. Springer, 2020.

[29] Nvidia. Nvidia a100 tensor core gpu architecture. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf, 2020.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8026–8037, 2019.

[31] Jeff Pool and Chong Yu. Channel permutations for n: M sparsity. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021.

[32] Olivier Giroux et al. Ronny Krashinsky. Nvidia ampere sparse tensor core. https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/, 2020.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[34] Wei Sun, Aojun Zhou, Sander Stuijk, Rob Wijnhoven, Andrew O Nelson, Henk Corporaal, et al. Dominosearch: Find layer-wise fine-grained n: M sparse schemes from dense neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021.

[35] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning (ICML)*, pages 10347–10357, 2021.

[36] Ziheng Wang. Sparsert: Accelerating unstructured sparsity on gpus for deep learning inference. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques (ICPACT)*, pages 31–42.

[37] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[38] Yuxin Zhang, Mingbao Lin, Chia-Wen Lin, Jie Chen, Yongjian Wu, Yonghong Tian, and Rongrong Ji. Carrying out cnn channel pruning in a white box. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 2022.

[39] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n: M fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations (ICLR)*, 2021.

[40] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. In *International Conference on Learning Representations (ICLR)*, 2017.

[41] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. In *International Conference on Learning Representations Workshop (ICLRW)*, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]