EFFICIENT WINNING TICKETS DRAWING OVER FINE-GRAINED STRUCTURED SPARSITY

Anonymous authors
Paper under double-blind review

ABSTRACT

The fine-grained structured sparsity has been proposed as a middle-ground between unstructured sparsity, where weights are pruned independently, and coarse-grained structured sparsity, where entire blocks of weights are pruned. Specifically, N:M fine-grained structured sparsity allows for at most N nonzero weights across a group of M consecutive weights. A recent implementation of 2:4 sparsity (N=2 and M=4) in Sparse Tensor Cores of Nvidia A100 GPUs shows significant improvement in throughput compared to unstructured sparsity while maintaining similar performance (e.g., accuracy). However, despite its potential for superior computational performance, how to efficiently train DNNs with N:M fine-grained structured sparsity remains a challenging problem. In this work, we leverage the recent advance of *Lottery Ticket Hypothesis* (LTH) and propose an iterative pruning algorithm for N:M fine-grained structured sparsity. By leveraging the N:M sparsity constraint, we can identify the unimportant weights across each group of M weights at earlier stages of iterative pruning, which significantly lowers the cost of iterative training compared to conventional unstructured pruning.

1 INTRODUCTION

In recent years, there has been a proliferation of Deep Neural Network (DNN) architectures that have achieved state-of-the-art performances across a variety of domains (He et al., 2016; Redmon et al., 2016; Vaswani et al., 2017). However, the algorithmic superiority of DNNs comes at extremely high computation and memory costs that pose significant challenges to the hardware platforms executing them, which seriously limits their deployments on resource-limited devices such as mobile phones, IoT devices, etc. To alleviate the computation cost, tremendous research efforts have been made on efficient DNN implementation. Among these approaches, DNN pruning, which aims at removing less important parameters from DNNs, has emerged as one of the main techniques to improve DNN resource efficiency.

DNN pruning can be generally categorized into *unstructured pruning* and *structured pruning*. The pruning mechanism in unstructured pruning removes DNN parameters individually, leaving the resulting DNN with nonzero weights that are distributed in an irregular pattern. Although this approach enables a high degree of model sparsity without impacting the accuracy, the irregular positions of the nonzero weights make efficient hardware implementation difficult. In contrast, structured pruning enforces that entire groups of model parameters are eliminated, making it more amenable for efficient hardware implementations. However, coarse-grained structured sparsity usually leads to a suboptimal model accuracy relative to unstructured sparsity due to the restricted flexibility of the sparsity pattern. Recently, a new class of fine-grained structured called N:M sparsity (Kung et al., 2019; Nvi, 2020) has received an increasing amount of attention by the DNN community. As shown in Figure 1, N:M sparsity enforces the constraint that there are at most N nonzero weights within a group of consecutive M (N<M) weights. This N:M sparsity structure has been shown to obtain significant acceleration on commodity hardware platforms compared to the original dense DNN. For instance, the Nvidia A100 GPU is equipped with the Sparse Tensor Cores to support the efficient implementation of 2:4 sparsity, which can achieve an average of $1.5\times$ inference latency reduction for the BERT-large model (Nvi, 2020).

Pruning algorithms for unstructured and structured sparsity have been studied extensively by the literature. In particular, the recently-proposed *Lottery Ticket Hypothesis* (LTH) (Frankle & Carbin,

2018) conjectures that dense, randomly-initialized DNNs always contain small sub-networks which can match or even outperform the test accuracy of original DNNs when trained alone from scratch. The process to find these sub-networks (i.e., winning tickets) utilizes a procedure called iterative magnitude pruning (IMP), which requires repeated cycles of training and pruning until a target sparsity level has been obtained. Specifically, to achieve a target sparsity of p, at each pruning round r ($1 \leq r \leq R$), the current DNN is first trained to convergence. After that, $1-(1-p)^{\frac{1}{R}}$ smallest weights are pruned by ranking all weights with their magnitude. The remaining weights with largest magnitudes are then rewound back to their values at an earlier iteration of training. Despite the superior accuracy for a target sparsity achieved by IMP, it requires multiple round of training, making winning tickets computationally expensive to find. With this in mind, this paper tries to answer the following question:

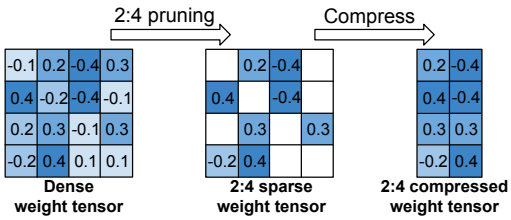

Figure 1: An example on 2:4 sparsity. Two weights are pruned for each row of four weights. The weights with larger magnitude are shown with darker color.

Does the N:M sparsity constraint enable winning tickets to be identified more efficiently?

In this work, we show that the answer is **yes** to the question above. In particular, we add additional constraints to IMP in order to enforce N:M sparsity. Importantly, we find that this constraint dramatically speeds up the procedure of iterative pruning without impacting model accuracy. The main contributions of the paper are:

- A novel *masked iterative magnitude pruning (mIMP)* algorithm that can learn N:M sparse DNN with high accuracy. At each pruning round, mIMP adds the smallest $M-N$ weights from each group into a *losing ticket pool*. mIMP progressively prune the weights in the losing ticket pool until the desired sparsity is reached.
- We leverage the structured nature of N:M sparsity with *Proactive Local Pruning (PLP)* which identifies and eliminates less important weights at earlier pruning rounds compared to mIMP. This substantially reduces the number of pruning rounds (and therefore training time) needed to reach a target sparsity.
- Finally, we show that the Early-Bird (EB) tickets (You et al., 2019) also appear consistently under the N:M sparsity constraint. We design an approach to identify the EB tickets at an earlier stage of training, which enables further reduction to the training cost by stopping each pruning round early. By integrating PLP and EB detection with mIMP, the result solution, termed *L-mIMP*, can reduce the training cost of mIMP by more than $20\times$ without impacting the accuracy of the final model.

2 Background and Related Works

2.1 Unstructured and Structured Pruning

DNN pruning has been studied extensively in the previous literature (Han et al., 2015; Molchanov et al., 2016; Sanh et al., 2020; He et al., 2017; Molchanov et al., 2016; Li et al., 2016; Frankle & Carbin, 2018; Liu et al., 2018; Zhuang et al., 2018; He et al., 2019). The tolerance of DNNs to small quantities of noise grants the possibility for DNNs to be compressed significantly without degrading the accuracy (e.g., $49\times$ for VGG-16 (Han et al., 2015)). Multiple criteria have been proposed for evaluating the importance of the weights, including magnitude-based pruning (Han et al., 2015), Hessian based heuristics (LeCun et al., 1990), gradient-based pruning (Molchanov et al., 2016; Sanh et al., 2020), etc. As discussed earlier, DNN pruning can be further divided into unstructured pruning and structured pruning. In unstructured pruning, each parameter is removed individually, giving rise to an irregular distribution of the nonzero weights. Although the resulting model has a high sparsity ratio, the irregular sparsity pattern makes the sparse DNN difficult to implement efficiently. This inefficiency leads to the proposal of structured pruning (Wen et al., 2016), which removes the DNN weights at a higher granularity (e.g., channel-wise pruning (Zhuang et al., 2018; Ye et al., 2018)

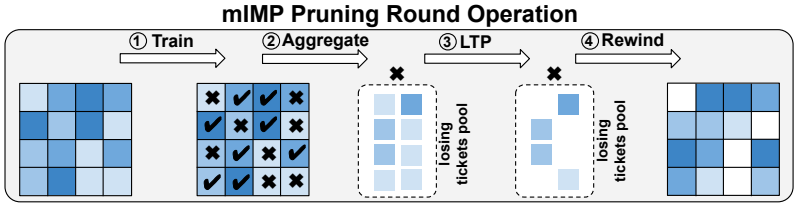

Figure 2: A single pruning round of mIMP with 2:4 sparsity. The step numbers are shown in circles.

and filter-wise pruning (He et al., 2019; Luo & Wu, 2020)). Structured pruning usually leads to a higher hardware utilization rate at the price of suboptimal model accuracy. In this work, we focus on N:M structured sparsity, which has not been extensively studied in the prior literature. In Zhou et al. (2021), the authors proposes SR-STE to train a N:M sparse network from the outset of training. In Hubara et al. (2021), the authors apply N:M sparsity to accelerate the DNN training.

2.2 Lottery Ticket Hypothesis

The *Lottery Ticket Hypothesis* (LTH) (Frankle & Carbin, 2018) conjectures that every dense, randomly-initialized DNNs always contains sparse sub-networks which can match or even outperform the test accuracy of the original network. These sparse sub-networks (i.e., winning tickets) can be found with Iterative Magnitude Pruning (IMP). At each pruning round of IMP, the current DNN is first trained until convergence. Then, a percentage of the weights are pruned based on their importance (i.e., magnitude), and the remaining weights are rewound to their values at some prior iteration k, where k is usually small (Frankle et al., 2019; 2020a; Renda et al., 2020). This iterative procedure is repeated until the target sparsity level is achieved. Despite its superior performance, IMP usually incurs a large training cost. Multiple following works have been conducted to mitigate this training overhead. For example, it has been shown that the resulting sparsity pattern of winning tickets can be transferred across different datasets (Morcos et al., 2019; Chen et al., 2020) and optimizers (Morcos et al., 2019) without significant accuracy loss. In You et al. (2019), the authors demonstrate that the winning tickets can be identified at a very early training stage, which reduces the training cost of IMP. While these studies have achieved significant saving on general sparsity pattern, none of them has been shown to work for N:M fine-grained structured sparsity.

3 Methodology

In this section, we first describe the *masked Iterative Magnitude Pruning* (mIMP) for obtaining N:M fine-grained structured sparsity. mIMP requires many rounds of training and further incurs a large training cost. To mitigate this cost, we leverage the structured nature in the N:M sparsity, and propose two additional methods to relieve the training cost while maintaining the high accuracy.

3.1 Masked Iterative Magnitude Pruning

In this section, we illustrate the procedure of mIMP for generating DNNs with N:M sparsity. An example of mIMP with 2:4 sparsity is given in Figure 2. For illustration purposes, the weight tensor has been reshaped into a size of 4×4 so that it is organized by groups of length M=4 across the rows. At the beginning of a pruning round, the DNN is first trained until convergence (step 1 in Figure 2). Within each row, the top N=2 weights with the largest magnitude, called *local winning tickets* (highlighted by ✓ in Figure 2), are separated from the rest weights, called *local losing tickets* (✗ in Figure 2). All the local losing tickets are then sent into the *losing tickets pool* (step 2). The losing tickets pool contains the local losing tickets from all the groups in the all the weight matrices across the entire DNN. After that, $p\%$ of the smallest weights are pruned from the losing ticket pool, where $p\%$ is a hyperparameter that specifies the percentage of removed weights at each pruning round (step 3). Note that no local winning tickets are pruned even if they have smaller magnitudes than losing tickets in other groups. We call this step *losing tickets pruning (LTP)*. Finally, the remaining weights in the weight matrix are rewound (step 4). This iterative process will repeat for multiple rounds. Between each round, the weights selected in the losing ticket pool may be different, but there

Algorithm 1: Masked Iterative Magnitude Pruning at Round r

Input: W_l^r is the DNN weight matrix of layer $l \in L$ at pruning round r. p is the percentage of weights pruned at each round. N weights are kept for every M weights. I is the total number of iterations within each pruning round. U_{lose} are sets that contain the local losing tickets. ϵ is the threshold for removing all the weights in U_{lose}.

Train $W_l^r, l \in L$ with I iterations until convergence.

for $l \leftarrow 1$ *to* L **do**

  for *every M weights in* W_l^r **do**

    Find the top N weights with the largest magnitude, collect the rest weights to U_{lose}.

Remove p percent of the weights from U_{lose} with the smallest magnitudes by setting their values to zero.

if $\frac{|U_{lose}|}{\sum_{l \in L} |W_l^r|} \leq \epsilon$ **then**

  Remove all the weights in U_{lose}. // **Clear the losing ticket pool if its size is small.**

Rewind the remaining weights.

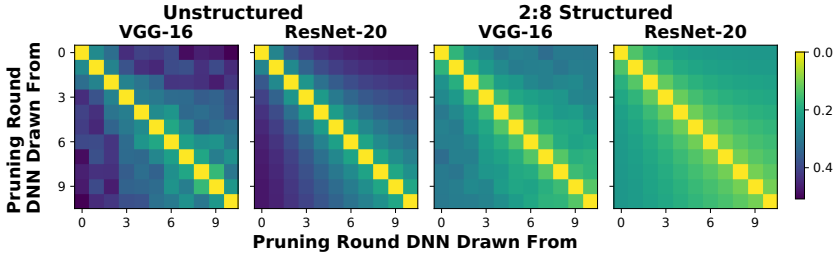

Figure 3: Average hamming distance between every pair of binary masks for unstructured and 2:8 structured sparsity on VGG-16 and ResNet-20.

are always N winning tickets remained for each group. This iterative process will continue until the losing ticket pool is exhausted and the target 2:4 sparsity will be reached. The completed mIMP algorithm is described by Algorithm 1.

3.2 Regularity of the Winning Tickets Distribution

The structured nature of the N:M sparsity enforces that for every group of M weights, only N of them will remain at the end of mIMP. Compared to the traditional unstructured pruning where the winning tickets can appear anywhere in the weight matrix, N:M sparsity places a strong restriction on the winning ticket distribution. To illustrate this, we perform ablation studies using two representative DNNs: ResNet-20 (He et al., 2016) and VGG-16 (Simonyan & Zisserman, 2014) on CIFAR-10. We follow the training details of Frankle et al. (2019). The batch size is set to 128 for all the DNNs. During each pruning round, we train the DNNs with 30K for ResNet-20 and 63K iterations for VGG-16. For ResNet-20, the learning rate is set to 0.1, 0.01 and 0.001 at the iterations 1, 20K and 25K, and the result weights at each pruning round is rewound to the values at the k=500th iteration. For VGG-16, we set the learning rate to 0.1, 0.01 and 0.001 at the iterations of 1, 32K and 48K, respectively. We rewind the result weights to their values at the k=1000th iteration. We apply SGD with a momentum of 0.9 and weight decay of 10 for DNN training. At each round, $p = 20\%$ of the nonzero weights with the smallest magnitude are removed using mIMP until 2:8 sparsity is achieved. For comparison, we apply the unstructured magnitude pruning on the same weight matrix, with the amount of pruned weights the same as that of mIMP for each pruning round. Define the binary weight mask produced at the end of the pruning round as $B \in \{0, 1\}^{|W|}$, where W is the weight matrix and $|W|$ is the total number of parameters in W, '1' means the corresponding weight remains, and vice versa. We record the binary masks of mIMP and unstructured pruning at the end of every pruning round and compute the average hamming distances between every pair of masks across the pruning round. From the 2D heatmap shown in Figure 3, we observe that the 2:8 structured sparsity incurs much smaller hamming distances between the masks than the unstructured sparsity. This indicates that N:M sparsity enables a much more regular winning ticket distribution over the unstructured sparsity, offering a great potential for us to detect and eliminate more unimportant weights at each pruning round. Therefore the mIMP process can terminate much earlier with a reduced training cost.

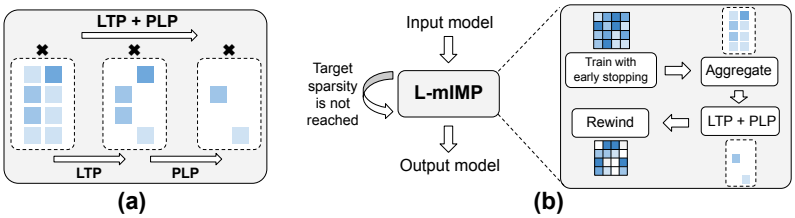

Figure 5: (a) Proactive local pruning (PLP) can be applied in addition to losing tickets pruning (LTP) for further reducing the weights in the losing ticket pool. (b) The steps of L-mIMP.

3.3 Proactive Local Pruning for N:M Sparsity

Given the regular mask patterns shown in Figure 3, one natural question is can we leverage this fact to early predict more unimportant weights that will be pruned by mIMP at later pruning rounds? For example, assume a weight will be pruned by mIMP at the fourth round, can we detect and prune this weight at early rounds? Given the fact that only N weights will be kept within every M weights at the end of mIMP, one approach is to proactively remove the weights that are much smaller than the rest weights within the same group after a period of training, because it is unlikely for the relatively smaller weights to surpass other relatively larger weights within the group over the remaining training process. This local comparison scheme allows us to early detect and eliminate more weights during each pruning round, which further accelerates the mIMP. To measure the relative magnitudes of the weights within a group, we first apply group-based normalization across each group of M weights. Specifically, denote $w^l_{j,k}$ the jth nonzero weights in the kth group in W_l, and denote $v^l_{k,j}$ the normalized version of $w^l_{k,j}$. $w^l_{k,j}$ will be removed if $|v^l_{k,j}| < \alpha$, where α is a hyperparameter that specifies the threshold. We name this pruning scheme *Proactive Local Pruning* (PLP). Figure 5 (a) shows how PLP can be integrated with mIMP to eliminate more weights in the losing ticket pool and further accelerate IMP. Algorithm 3 in the Appendix describes the PLP in detail.

To evaluate the performance of PLP, we train ResNet-20 and VGG-16 on CIFAR-10 using the same setting as we used in Section 3.2. At the end of each pruning round, we first apply the LTP in the losing ticket pool (step 3 of Figure 2). Among the remaining weights in the losing ticket pool, we compute their normalized values $v^l_{k,j}$ and record weights that would be pruned by PLP (i.e., $|v^l_{k,j}| < \alpha$). Using this, we can determine the percentage of weights that are predicted mistakenly (i.e., percentage of the weights that would have been pruned under PLP but are actually kept under mIMP without PLP). We evaluate PLP by setting $\alpha = -0.5, -0.3, -0.1$ on 2:8 sparsity pattern. The results are given in Figure 4 (a). We have the following observations: First, the misprediction rate decreases with α. This is because the weights pruned by a smaller α have a smaller relative magnitudes, and are therefore more likely to be pruned with magnitude pruning. Second, the misprediction rate decreases rapidly after the initial pruning rounds (e.g., first two rounds) and becomes stable in the later rounds. For example, with $\alpha = -0.5$, the misprediction rate decreases from 20% to less than 6% at round 3, and results in a misprediction rate of 1.8% at pruning round 6 on ResNet-20. This suggests that PLP can be applied more successfully after the initial pruning rounds (e.g., two rounds). We have performed a similar study for ResNet-20 and VGG-16 on ImageNet (Figure 4(b)) and results show a similar trend.

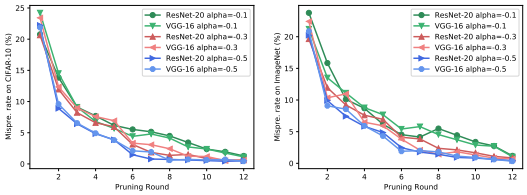

Figure 4: Misprediction rate by PLP after each pruning round on CIFAR-10 (left) and ImageNet (right).

3.4 Early-bird Tickets in N:M Sparsity

You et al. (2019) showed that winning tickets can be identified at earlier iterations in the training process. For each pruning round, instead of training the DNN model for a full I iteration, we can stop the training process at $I'(I' < I)$ iteration and use the intermediate weights to identify the winning

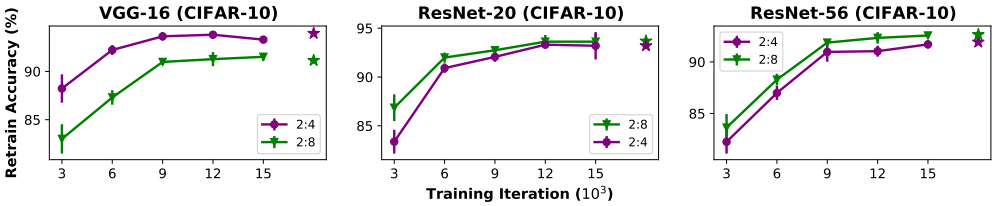

Figure 6: The performance of the EB tickets drawn from different iterations with 2:4 and 2:8 sparsity over VGG-16, ResNet-20 and ResNet-56. The vertical bars show the 95% confidence interval. The star signs indicate the accuracies acquired by training the DNNs with full I iterations.

tickets. The result winning tickets, termed as *Early-Bird (EB) tickets*, can obtain a similar and even higher accuracy than the winning tickets drawn using the fully trained DNN.

We first check the existence of EB tickets on the N:M sparsity. To test this, we apply mIMP to train ResNet-18, ResNet-50 and VGG-16 on CIFAR-10 and ImageNet. At each pruning round, instead of training the DNN for the full I iterations, we stop the training process at an earlier iteration $I'(I' < I)$. The rest settings are the same as described Section 3.2. Figure 6 shows the best test accuracies achieved by mIMP with the intermediate weights drawn from different iterations. For CIFAR-10, the EB tickets generally exist as early as $I' = 6K - 9K$ iterations (with a total of $I = 30K$ iterations for ResNet-18 and ResNet-56 and $I = 63K$ iterations for VGG-16).

The consistent observations above indicate that we can apply the EB tickets in the N:M sparsity to further reduce the training cost. To detect the EB tickets, we adopt a similar approach as You et al. (2019) by computing the hamming distances between the binary masks generated by the intermediate weights at the different iterations. If the hamming distances between the consecutive masks is less than a predefined threshold β, the current pruning round will be stopped early.

3.5 Efficient mIMP Training

PLP and early stopping mechanism described in Section 3.3 and Section 3.4 can be leveraged and integrated with mIMP to greatly reduce the training cost. This result algorithm, termed as *Light mIMP (L-mIMP)*, is described in Figure 5(b). The detailed algorithm is presented by Algorithm 2 in the Appendix.

4 Experiments

In this section, we evaluate the performance of mIMP and L-mIMP on different applications including multiple CNNs (ResNet-18 (He et al., 2016), ResNet-50 (He et al., 2016) and VGG-16 (Simonyan & Zisserman, 2014)) on CIFAR-10 and ImageNet (Deng et al., 2009), and pretrained BERT model (Devlin et al., 2018) on GLUE tasks (Wang et al., 2018). We first show the performances of mIMP and L-mIMP on CIFAR-10 and ImageNet in Section 4.1. Then we compare L-mIMP with the other early pruning approaches in Section 4.2. In Section 4.3, we conduct the ablation studies by changing the threshold parameters α and β of L-mIMP and show its corresponding impacts on the performance. Section 4.4 depicts the evaluation of the sparse BERT model on the GLUE benchmarks.

4.1 Performance Evaluation on CIFAR and ImageNet

In this section, we evaluate the performance of mIMP and L-mIMP on CIFAR-10 and ImageNet. For mIMP and L-mIMP, we consider three types of sparsities: 2:4, 1:4 and 2:8. The sparsity level of 2:4 is 50%, and the sparsity level for 1:4 and 2:8 are both 75%. The implementation details for these experiments are listed in Section A.3 in the Appendix. We consider multiple recent iterative pruning algorithms for comparison, including Nvidia ASP (Nvi, 2020), EB (You et al., 2019). In ASP, the dense DNN is first trained until converge, the result DNN is then pruned with N:M sparsity using magnitude-based single-shot pruning. Finally the sparse DNN is trained for one more round to recover the accuracy. The EB method further reduces the training cost of ASP by early stopping the first round of the DNN training, and the intermediate weights are used for magnitude pruning to

	Methods	ResNet-20		ResNet-56		VGG-16	
		Acc.(%)	FLOPs (10^{15})	Acc.(%)	FLOPs (10^{15})	Acc.(%)	FLOPs (10^{15})
Sparsity Ratio 2:4	mIMP	91.29	6.40	93.42	12.79	93.18	35.92
	L-mIMP	91.22	0.42	93.25	0.72	93.09	3.70
	mIMP-high	90.83	4.32	92.83	8.18	92.74	27.30
	mIMP-rand	90.40	6.40	92.33	12.79	92.43	35.92
	ASP (Nvi, 2020)	90.69	0.47	92.67	0.86	92.67	4.22
	EB (You et al., 2019)	90.51	0.39	92.63	0.72	92.46	3.48
	L-mIMP Improvement	**+0.40**	**8.6**×	**+0.41**	**9.8**×	**+0.33**	**6.1**×
Sparsity Ratio 1:4	mIMP	91.01	8.66	93.12	17.30	92.90	57.76
	L-mIMP	90.82	0.45	92.94	0.80	92.76	4.02
	mIMP-high	90.38	6.11	92.36	11.13	92.47	41.35
	mIMP-rand	89.98	8.66	91.75	17.30	92.24	57.76
	ASP (Nvi, 2020)	90.13	0.47	92.21	0.86	92.31	4.22
	EB (You et al., 2019)	90.11	0.41	92.22	0.70	92.23	3.65
	L-mIMP Improvement	**+0.42**	**10.8**×	**+0.51**	**11.8**×	**+0.38**	**8.2**×
Sparsity Ratio 2:8	mIMP	91.13	8.66	93.26	17.30	93.02	57.76
	L-mIMP	90.90	0.44	93.02	0.81	92.89	4.00
	mIMP-high	90.46	6.22	92.53	11.80	92.55	44.35
	mIMP-rand	90.11	8.66	92.03	17.30	92.28	57.76
	ASP (Nvi, 2020)	90.29	0.47	92.41	0.86	92.36	4.22
	EB (You et al., 2019)	90.37	0.41	92.38	0.71	92.30	3.68
	L-mIMP Improvement	**+0.37**	**8.9**×	**+0.41**	**11.8**×	**+0.36**	**8.4**×
Dense	-	91.27	-	93.45	-	93.21	-

Table 1: Performance on CIFAR-10 on ResNet-20, ResNet-56 and VGG-16 over different methods. The **L-mIMP Improvement** is the average improvement of L-mIMP compared to the other methods. **Dense** is the accuracy of the dense DNN.

achieve N:M sparsity. We adopt the default settings used in You et al. (2019) for EB method, where the queue length and the threshold on the mask distance is set to 5 and 0.1, respectively. Additionally, we consider two variations on the mIMP. The first variation, *mIMP-high*, achieves the training cost reduction by increasing the pruning ratio p from 20% to 40%, so that more weights will be pruned from the losing ticket pool for each pruning round. The remaining settings for mIMP-high are the same as mIMP. The second variation, *mIMP-rand* is the same as mIMP, except that the weights are initialized using random numbers generated by the weight initialization function rather than the early values during each pruning round. The purpose of mIMP-rand is to investigate the effectiveness of weight rewinding proposed in Frankle et al. (2019) on N:M sparsity.

Table 1 shows the test accuracies of the algorithms and their corresponding training costs (in terms of FLOPs) under 2:4, 1:4 and 2:8 sparsity levels for different DNNs on CIFAR-10. We also show the test accuracies for the dense DNN as well as the improvement of L-mIMP by comparing against the average accuracy and training cost (denoted as L-mIMP Improvement). First, we see that mIMP generally outperforms the other algorithms on test accuracy across all DNNs and sparsity ratios. It also achieves a comparable accuracy to the dense DNN. Second, L-mIMP achieves a slightly lower test accuracies than mIMP with a $11\times$ lower training FLOPs on average. L-mIMP also consumes comparable training FLOPs than mIMP-high, ASB and EB, while consistently achieving a $1-1.5\%$ higher accuracy. Finally, mIMP-rand performs poorly across all the DNNs, which indicates that weight rewinding is necessary for maintaining the accuracy under N:M sparsity. Table 2 depicts the evaluation results of ResNet-18, ResNet-50 and VGG-16 on ImageNet. All the implementation details are given in Section A.3. From table 2, we observe a similar trend as CIFAR-10. mIMP achieves the best accuracy performance over the other algorithms, while L-mIMP obtains a slightly lower accuracy with a much lower training cost.

4.2 Comparison With the Early Pruning Methods

Next, we compare L-mIMP with several *early pruning* approaches. In these methods, each weight are first assigned with an importance score using some sample training data. For example, SNIP (Lee et al., 2018) uses the magnitude of the product between the weight gradient and weight value as the importance score, while GraSP (Wang et al., 2020) utilizes the hessian-gradient products to estimate the importance score. In addition, we also consider another baseline, named *Early Magnitude Pruning (EMP)*, which adopts the initial weight magnitude as the importance score. Given the

	Methods	ResNet-18		ResNet-50		VGG-16	
		Acc.(%)	FLOPs (10^{18})	Acc.(%)	FLOPs (10^{18})	Acc.(%)	FLOPs (10^{18})
Sparsity Ratio 2:4	mIMP	70.35	5.28	76.32	10.50	71.00	29.3
	L-mIMP	70.18	0.32	76.13	0.55	70.87	1.97
	mIMP-high	69.73	3.68	75.52	6.98	70.30	21.67
	mIMP-rand	68.90	5.28	74.94	10.50	69.42	29.3
	ASP (Nvi, 2020)	69.33	0.37	75.58	0.61	69.86	2.57
	EB (You et al., 2019)	68.85	0.28	75.33	0.51	69.58	1.68
	L-mIMP Improvement	**+0.63**	**9.4**×	**+0.49**	**10.6**×	**+0.69**	**8.6**×
Sparsity Ratio 1:4	mIMP	69.94	7.13	75.73	14.18	70.66	47.17
	L-mIMP	69.82	0.35	75.55	0.59	70.50	2.35
	mIMP-high	69.24	4.98	75.13	9.39	69.72	29.05
	mIMP-rand	68.60	7.13	74.30	14.18	69.07	47.17
	ASP (Nvi, 2020)	68.98	0.38	74.51	0.61	69.23	2.57
	EB (You et al., 2019)	68.34	0.36	74.30	0.56	69.20	2.10
	L-mIMP Improvement	**+0.67**	**11.4**×	**+0.64**	**13.2**×	**+0.77**	**10.9**×
Sparsity Ratio 2:8	mIMP	70.11	7.01	76.02	14.18	70.78	47.16
	L-mIMP	69.92	0.36	75.72	0.58	70.63	2.41
	mIMP-high	69.48	5.17	75.21	9.50	69.77	29.1
	mIMP-rand	68.69	7.01	74.59	14.18	69.36	47.16
	ASP (Nvi, 2020)	69.11	0.38	74.78	0.61	69.64	2.57
	EB (You et al., 2019)	68.76	0.34	74.86	0.52	69.29	2.21
	L-mIMP Improvement	**+0.58**	**11.1**×	**+0.53**	**13.5**×	**+0.72**	**10.6**×
Dense	-	70.53	-	76.50	-	71.12	-

Table 2: Performance on ImageNet for ResNet-18, ResNet-50 and VGG-16 over different methods.

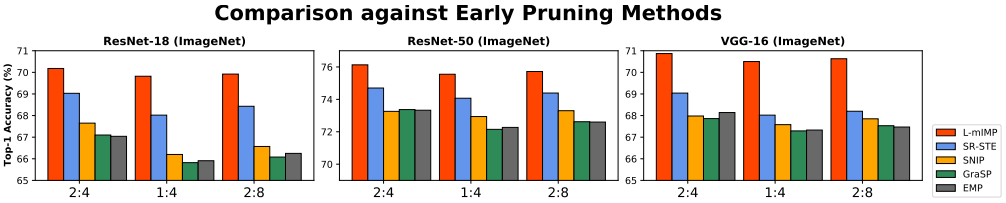

Figure 7: Accuracy comparison between L-mIMP and the early pruning methods. All the early pruning methods have a training FLOPs of $0.19E(10^{18})$, $0.33E$ and $1.32E$ on ResNet-18, ResNet-50 and VGG-16, respectively. The training FLOPs of L-mIMP are shown in Table 3.

importance scores, the weights are then pruned at one-shot based on their scores to reach the target sparsity level. In SR-STE (Zhou et al., 2021), the N:M sparse DNN is trained from scratch using sparse-refined straight through estimator (SR-STE). Ignoring the overhead of the score computation, the above approaches only requires to train the DNN once, which significantly reduces the training cost. However, the resulting sparse DNN accuracies are usually suboptimal to those obtained via IMP (Frankle et al., 2020b).

We compare the test accuracies of L-mIMP ($\alpha = -0.1, \beta = 0.1$) with these approaches on ImageNet under different sparsity levels. We train ResNet-20, ResNet-50 and VGG-16 with each of the early pruning method using a batch size of 128 for 90 epochs. The remaining training details (e.g., learning rate, optimizer) are the same as described in Section A.3 in the Appendix. Figure 7 shows the accuracies of different methods. We notice that L-mIMP outperforms the other approaches by $2-4\%$ on average, at a price of $1.8\times$ higher training FLOPs on average.

4.3 Ablation Study on Threshold Parameters

In this section, we evaluate the impact of the threshold parameters α and β on the test accuracy and training costs of L-mIMP. α controls the criteria of weight removal within the group for PLP, while β specifies the average hamming distances between the consecutive binary masks for EB detection. Table 3 show the changes to test accuracies and training FLOPs under different α and β on ImageNet. We can see that both the accuracies and training FLOPs decrease as the α and β grow. In particular, compared with the setting of $\alpha = -0.1, \beta = 0.1$, smaller threshold values ($\alpha = -0.3, \beta = 0.05$) can obtain almost identical accuracy performance as mIMP with 4-6$\times$ lower training cost. This indicates

Methods	Sparsity ratio	ResNet-18		ResNet-50		VGG-16	
		Acc.(%)	FLOPs (10^{18})	Acc. (%)	FLOPs (10^{18})	Acc. (%)	FLOPs (10^{18})
L-mIMP	2:4	70.29	1.43	76.26	3.44	70.93	9.12
(α=-0.3,	1:4	69.89	1.67	75.66	4.49	70.64	13.88
β=0.05)	2:8	70.06	1.70	75.94	4.40	70.72	13.75
L-mIMP	2:4	70.23	1.06	76.18	1.93	70.90	4.13
(α=-0.1,	1:4	69.85	1.31	75.60	2.40	70.54	6.06
β=0.05)	2:8	70.01	1.30	75.89	2.51	70.67	6.08
L-mIMP	2:4	70.18	0.32	76.13	0.55	70.87	1.97
(α=-0.1,	1:4	69.82	0.35	75.55	0.59	70.50	2.35
β=0.1)	2:8	69.92	0.36	75.72	0.58	70.63	2.41

Table 3: Performance under different α and β on ImageNet.

	Methods	MRPC		SST-2		MNLI	
		Acc.	FLOPs (10^{15})	Acc.	FLOPs (10^{15})	Acc.	FLOPs (10^{15})
Sparsity Ratio 2:4	mIMP	84.40	4.17	92.11	74.40	82.60	440.88
	L-mIMP	84.18	0.48	91.90	8.72	82.46	49.90
	mIMP-high	83.66	3.26	91.41	60.21	81.90	386.71
	ASP (Nvi, 2020)	83.06	0.51	91.16	9.22	81.57	54.43
	EB (You et al., 2019)	82.79	0.45	91.21	7.99	81.48	43.39
Sparsity Ratio 2:8	mIMP	84.08	5.86	91.73	88.94	82.37	533.61
	L-mIMP	83.80	0.51	91.61	9.87	82.11	50.85
	mIMP-high	83.12	4.70	91.04	69.71	81.65	410.84
	ASP (Nvi, 2020)	82.45	0.51	90.60	9.22	81.27	54.43
	EB (You et al., 2019)	82.28	0.46	90.57	8.03	81.19	44.96
Dense	-	84.44	-	92.12	-	82.66	-

Table 4: Performance comparison of BERT on MRPC, SST-2 and MNLI.

that α and β can be adjusted to obtain a trade-off between accuracy and training cost. The results for CIFAR-10 is given by Table 5 in the Appendix.

4.4 Performance Evaluation on BERT

In this section, we apply mIMP and L-mIMP to prune the pretrained BERT model (Devlin et al., 2018) with the GLUE benchmark Wang et al. (2018). GLUE consists of nine downstream tasks which further includes two single sentence task (CoLA, SST-2), three similarity and paraphrase tasks (MRPC, QQP, STS-B) and four inference tasks (MNLI, QNLI, RTE, WNLI). We adopt the 12-layer pretrained $BERT_{BASE}$ model from Huggingface repository (Hug) and attach a separate classification layer on the BERT output for each GLUE task. During each pruning round, we finetune the BERT model for three epochs with a batch size of 32 for each task. We adopt the AdamW optimizer (Loshchilov & Hutter, 2017) with an initial learning rate of 2×10^{-5}. For mIMP and L-mIMP, we set $R_t = 2$, $\epsilon = 0.01$, $p = 20\%$, $\alpha = -0.1$ and $\beta = 0.1$ as before.

Table 4 shows the test accuracies and the corresponding training FLOPs on MRPC, SST-2 and MNLI tasks from the GLUE. The performances on the remaining tasks are given in the Appendix. We can see that the superior performance of mIMP and L-mIMP also translates to the BERT model. Specifically, L-IMP can achieve almost the same performance as mIMP with $10\times$ lower training FLOPs. In addition, L-mIMP can achieve $1\% - 2\%$ higher accuracy than the other baseline algorithms on average.

5 Conclusion

In this paper, we propose mIMP and L-mIMP for efficient learning of N:M sparsity across multiple DNN models with negligible accuracy loss. We show that, compared to unstructured sparsity, the N:M sparsity offers a better opportunity to identify the unimportant weights at earlier stages of iterative pruning, which significantly lowers the cost of iterative training. Evaluation over multiple DNN models and datasets indicates that we can achieve superior accuracy performance with extremely low training cost.

REFERENCES

Huggingface official repository. https://huggingface.co/transformers/model_doc/bert.html.

Nvidia a100 tensor core architecture, 2020. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks. *arXiv preprint arXiv:2007.12223*, 2020.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*, 2019.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pp. 3259–3269. PMLR, 2020a.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576*, 2020b.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4340–4349, 2019.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, 2017.

Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Seffi Naor, and Daniel Soudry. Accelerated sparse neural training: A provable and efficient method to find n: m transposable masks. *arXiv preprint arXiv:2102.08124*, 2021.

H. T. Kung, Bradley McDanel, and Sai Qian Zhang. Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization. *24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

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

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *International Conference on Learning Representations*, 2016.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *Pattern Recognition*, 107:107461, 2020.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Ari S Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. *arXiv preprint arXiv:1906.02773*, 2019.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.

Victor Sanh, Thomas Wolf, and Alexander M Rush. Movement pruning: Adaptive sparsity by fine-tuning. *arXiv preprint arXiv:2005.07683*, 2020.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.

Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018.

Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G Baraniuk, Zhangyang Wang, and Yingyan Lin. Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*, 2019.

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*, 2021.

Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. *arXiv preprint arXiv:1810.11809*, 2018.

A APPENDIX

A.1 DETAILED IMPLEMENTATION OF L-MIMP

The detailed implementation of the L-mIMP is described in Algorithm 2.

Algorithm 2: L-mIMP Algorithm

Input: W_l^r is the weight matrix of layer $l \in L$ at pruning round $r \in R$. $|W_l^r|$ is number of weights in W_l^r.
p is the percentage of weights pruned in the local losing tick pool at each round.
N weights are kept for every M weights.
I is the total number of iterations at each pruning round.
I_{eb} is period for detecting the EB tickets.
B_l and B_l^{last} are the current and last binary masks at layer l.
R_t is the pruning round to start the proactive local pruning.
K is the number of weights groups, $K = \frac{|W_l^r|}{M}$.
U_{lose} is set that contains the local losing tickets. $|U_{lose}|$ is the size of U_{lose}.
α, β are the thresholds for proactive local pruning and EB detection.
ϵ is the threshold for removing all the weights in U_{lose}.

for $r \leftarrow 1$ *to* R **do**
  for $i \leftarrow 1$ *to* I **do**
    Train $W_l^r, \forall l \in L$ for one iteration with the training set.
    if *i is a multiple of* I_{eb} **then**
      for $l \leftarrow 1$ *to* L **do**
        for $k \leftarrow 1$ *to* K **do**
          Find the top N weights with the largest magnitude and put the rest nonzero weights to U_{lose}.
          Aggregate all the nonzero weights $w_{k,j}^l$ in group k, compute the normalized weights $v_{k,j}^l$.
      Prune the smallest p percent of weights from U_{lose}. // **Losing Tickets Pruning (LTP)**
      for $l \leftarrow 1$ *to* L **do**
        for $k \leftarrow 1$ *to* K **do**
          if $r > R_t$ *and* $v_{k,j}^l \leq \alpha$ *and* $w_{k,j}^l \in U_{lose}$ **then**
            Remove it from U_{lose}. // **Proactive Local Pruning (PLP)**
      if $\frac{|U_{lose}|}{\sum_{l \in L} |W_l^r|} \leq \epsilon$ **then**
        Remove all the weights in U_{lose}. Terminate the L-mIMP process.
      Record the result binary mask B_l for each layer $l \in L$.
      Compute the average hamming distance between B_l^{last} and B_l for each layer $l \in L$.
      Set $B_l^{last} = B_l$ for each layer $l \in L$.
      if *Average hamming distance between* $B_{l \in L}$ *and* $B_{l \in L}^{last}$ *is less than* β **then**
        Break the inner for loop. // **EB is detected, current round r can be stopped early.**
  Rewind the remaining weights.

L-mIMP integrates PLP (Section 3.3) and early stopping (Section 3.4) with mIMP (Section 3.1), which greatly reduces the training cost by reducing the number of pruning rounds and the number of training iterations per pruning round.

A.2 DETAILED IMPLEMENTATION OF PLP

Algorithm 3 depicts the detailed operations of PLP. All the nonzero weights in the group will be first aggregated and normalized. If a weight is in the losing ticket pool, and its normalized weights value is less than the threshold α, then it will be pruned by PLP.

A.3 PARAMETER SETTINGS OF SECTION 4.1

For the hyperparameter in Algorithm 2, we set the threshold parameter ϵ and percentage of pruned weight p to 0.01 and 20%, respectively. We set the threshold parameters α and β to -0.1 and 0.1,

Algorithm 3: Proactive Local Pruning (PLP)

Input: W_l^r is the DNN weights at layer $l \in L$.
α is the threshold for detection.
M weights are kept for every N weights.
K is the number of weights groups, $K = \frac{|W_l^r|}{N}$.
U_{lose} are the local losing tickets pool.

for $l \leftarrow 1$ *to* L **do**
  for $k \leftarrow 1$ *to* K **do**
    Aggregate all the nonzero weights $w_{k,j}^l$ in the group k, find the normalized weights $v_{k,j}^l$.
  if $v_{k,j}^l \leq \alpha$ *and* $w_{k,j}^l \in U_{lose}$ **then**
    Remove it from U_{lose}.

Methods	Sparsity ratio	ResNet-20		ResNet-56		VGG-16	
		Acc.(%)	FLOPs (10^{15})	Acc.(%)	FLOPs (10^{15})	Acc.(%)	FLOPs (10^{15})
L-mIMP	2:4	91.27	2.11	93.33	3.96	93.17	10.86
(α=-0.3,	1:4	90.94	2.81	93.05	4.87	92.87	14.08
β=0.05)	2:8	91.05	2.79	93.13	4.80	92.95	14.11
L-mIMP	2:4	91.22	1.04	93.31	1.39	93.13	4.44
(α=-0.1,	1:4	90.91	1.55	92.97	2.01	92.80	6.78
β=0.05)	2:8	90.96	1.56	93.09	2.08	92.94	6.80
L-mIMP	2:4	91.22	0.42	93.25	0.72	93.09	3.70
(α=-0.1,	1:4	90.82	0.45	92.94	0.80	92.76	4.02
β=0.1)	2:8	90.90	0.44	93.02	0.81	92.88	4.00

Table 5: Performance under different α and β on CIFAR-10.

respectively. The EB tickets are checked at every $I_{eb} = 3000$ iterations starting after $R_t = 2$ pruning rounds.

For the training on CIFAR-10, the batch size is set to 128 for all the DNNs. We train the DNNs with 30K for ResNet-20 and 63K iterations for VGG-16 for each pruning round except for the EB and L-mIMP, which can stop the training process at an early stage. For ResNet-20 and ResNet-56, the learning rate is set to 0.1, 0.01 and 0.001 at the iterations 1, 20K and 25K, respectively. The result weights at each pruning round is rewound to the values at the k=500th iteration. For VGG-16, the learning rate is set to 0.1, 0.01 and 0.001 at the iterations of 1, 32K and 48K, respectively. We rewind the result weights to their values at the k=1000th iteration. We apply SGD with a momentum of 0.9 and weight decay of 10 for DNN training.

For ImageNet, all the DNNs are trained with 90 epochs with a batch size of 128 at each pruning round, the initial learning rate is set to 0.1, and is decreased by $10\times$ at the epoch 30, 60, 80, respectively. The result weights at each pruning round are rewound to the values at the fourth epochs at the start of each pruning round (Frankle et al., 2020a).

A.4 Ablation Study on CIFAR-10

Table 5 describes the ablation study on CIFAR-10 by changing the hyperparameter α and β of L-mIMP.

A.5 Performance on the Rest GLUE Tasks

Table 6 and Table 7 show the performance of mIMP and L-mIMP on the rest GLUE tasks including CoLA, QQP, QNLI, WNLI, RTE and STS-B. They indicate a similar results as Table 4.

	Methods	CoLA		QQP		QNLI	
		Acc.	FLOPs (10^{15})	Acc.	FLOPs (10^{15})	Acc.	FLOPs (10^{15})
Sparsity Ratio 2:4	mIMP	53.54	9.24	89.91	416.23	88.27	122.25
	L-mIMP	53.39	0.98	89.80	45.31	88.10	14.72
	mIMP-high	52.79	7.18	89.42	350.80	87.77	103.76
	ASP (Nvi, 2020)	52.20	1.12	88.76	49.93	87.24	15.41
	EB (You et al., 2019)	52.28	0.91	88.67	41.60	87.31	13.85
Sparsity Ratio 2:8	mIMP	53.41	10.76	89.77	487.04	87.83	143.96
	L-mIMP	53.33	1.05	89.61	48.92	87.73	17.88
	mIMP-high	52.52	8.86	89.12	378.42	87.46	119.69
	ASP (Nvi, 2020)	52.10	1.12	88.58	49.93	87.10	15.41
	EB (You et al., 2019)	52.11	0.94	88.43	42.15	87.02	13.83
Dense	-	53.61	-	89.89	-	88.30	-

Table 6: Accuracy comparison of BERT on CoLA, QQP and QNLI.

	Methods	WNLI		RTE		STS-B	
		Acc.	FLOPs (10^{15})	Acc.	FLOPs (10^{15})	Acc.	FLOPs (10^{15})
Sparsity Ratio 2:4	mIMP	54.01	0.70	65.60	2.25	87.79	6.32
	L-mIMP	53.88	0.09	65.42	0.25	87.55	0.70
	mIMP-high	53.61	0.61	65.24	2.02	87.23	4.96
	ASP (Nvi, 2020)	53.05	0.10	64.87	0.29	86.89	0.78
	EB (You et al., 2019)	53.07	0.08	64.61	0.24	86.91	0.51
Sparsity Ratio 2:8	mIMP	53.45	0.88	65.18	2.70	87.62	8.19
	L-mIMP	53.28	0.10	65.04	0.28	87.48	0.74
	mIMP-high	52.97	0.71	64.70	2.39	87.17	6.65
	ASP (Nvi, 2020)	52.60	0.10	64.29	0.29	86.73	0.78
	EB (You et al., 2019)	52.36	0.08	64.13	0.26	86.69	0.54
Dense	-	54.09	-	65.66	-	87.81	-

Table 7: Accuracy comparison of BERT on WNLI, RTE and STS-B.