Dynamic Sparse Training: Find Efficient Sparse Network from scratch with Trainable Masked Layers

Anonymous authors
Paper under double-blind review

Abstract

We present a novel network pruning algorithm called Dynamic Sparse Training that can jointly find the optimal network parameters and sparse network structure in a unified optimization process with trainable pruning thresholds. These thresholds can have fine-grained layer-wise adjustments dynamically via backpropagation. We demonstrate that our dynamic sparse training algorithm can easily train very sparse neural network models with little performance loss using the same training epochs as dense models. Dynamic Sparse Training achieves prior art performance compared with other sparse training algorithms on various network architectures. Additionally, we have several surprising observations that provide strong evidence to the effectiveness and efficiency of our algorithm. These observations reveal the underlying problems of traditional three-stage pruning algorithms and present the potential guidance provided by our algorithm to the design of more compact network architectures.

1 Introduction

Despite the impressive success that deep neural networks have achieved in a wide range of challenging tasks, the inference of deep neural network is highly memory-intensive and computation-intensive due to the over-parameterization of deep neural networks. Network pruning (LeCun et al. (1990); Han et al. (2015); Molchanov et al. (2017)) has been recognized as an effective approach to improving the inference efficiency in resource-limited scenarios. Typical pruning methods consist of dense network training followed with expensive pruning and fine-tuning iterations, where many non-trivial hyperparameters need to be set properly.

In terms of the granularity of sparsity, current network pruning methods can be divided into unified (Li et al. (2016); He et al. (2017); Luo et al. (2017)) or dynamic strategy (Suau et al. (2018); Liu et al. (2018a); Molchanov et al. (2016); Huang & Wang (2018)). The layer-wise percentage of parameter pruned can either be unified or dynamically adjusted by the pruning algorithm. For example, if we want to prune $x\%$ of overall parameters in a neural network, the least important $x\%$ parameters in each layer will be pruned with a local pruning threshold for the unified strategy. For the dynamic strategy, a single global pruning threshold (Molchanov et al. (2019); Lin et al. (2018)) or layer-wise greedy algorithms (Luo et al. (2017); He et al. (2017)) are applied to achieve different layer-wise sparsity.

However, due to the high complexity of deep neural networks, different layers may carry different degrees of redundancy and relative importance. Simply pruning the same percentage of parameter in each layer may significantly damage the model capacity, which leads to performance loss. For example, the last layer of a deep neural network can directly affect the output of the network so usually it should be pruned more carefully with less percentage of parameters pruned. Thus, the dynamic pruning strategy tends to have advantages over the unified strategy. Despite the benefit of dynamic pruning, it is quite challenging to develop dynamic pruning algorithms. Using a single global pruning threshold is exceedingly difficult to assess the local parameter importance of individual layer, since each layer has a significantly different amount of parameter and contribution to the model performance. This makes pruning algorithms based on a single global threshold non-robust. The problem of layer-by-layer greedy pruning methods is that the unimportant neurons in an early layer can have a significant influence on the responses in later layers (Yu et al., 2018), which may result in propagation and amplification of the reconstruction error.

Meanwhile, most of the existing pruning algorithms conduct ”hard” pruning that irreversibly removes the network connectivity. If a network connection is pruned, the corresponding network parameter will be set to zero and will not be updated via back-propagation in the fine-tuning process afterward. However, due to the extremely complicated and convoluted interconnections among the huge amount of neurons in modern neural networks, the relative importance of connection may change dramatically in the pruning process. Therefore, there are two main problems of ”hard” pruning algorithm. Firstly, possible irretrievable network damages may be incurred, which leads to failure in finding the optimal sparse structure. Secondly, directly setting the non-zero parameter to zero has unknown side effects. He et al. (2018) propose ”soft” pruning algorithm that allows the pruned connection to be updated in later back-propagation, which only solves the first problem.

In this work, we develop a novel dynamic pruning algorithm called dynamic sparse training that achieves layer-wise dynamic pruning and solves the two problems of ”hard” pruning. The current unimportant parameters are masked instead of pruned by directly setting to zero, thus the historical information about the parameter importance is preserved and the masked parameter can be easily unmasked. We demonstrate that it is the collective effect of layer-wise dynamic pruning thresholds and ability to recover network connectivity that helps to find the optimal sparse structure. The proposed algorithm finds the sparse network during the training process directly from a randomly initialized network without the burdensome pruning and fine-tuning iterations. Our proposed algorithm has following promising properties:

Simplicity. The pruning process and the training process are combined in the proposed algorithm hence the expensive pruning and fine-tuning iterations are circumvented. Meanwhile, our method can control the final percentage of parameter remaining with a single hyperparameter. Thus it is easy to obtain well-performed model with different sparse levels.
Better performance. The layer-wise sparsity and pruning thresholds can be adjusted dynamically according to a predefined loss via back-propagation. Together with the ability to recover pruned connections, our method can find better sparse structure than other existing sparse learning methods.
Versatility. Since our method has no assumptions on the network architecture, it can be applied to various kinds of neural network layers including fully connected, convolutional and recurrent networks.

We evaluate our algorithm on MNIST and CIFAR-10 with various modern network architectures. On all the tested network architectures, our method can train highly sparse models directly with little performance loss compared with dense counterparts. Furthermore, we analyze two surprising observations about the change of layer-wise parameter sparsity during the dynamic sparse training process and the generated sparse pattern, which provides strong evidence to the effectiveness and efficiency of the proposed method.

2 Related work

Traditional Pruning Methods: LeCun et al. (1990) presented the early work about network pruning using second-order derivatives as the pruning criterion. The effective and popular training, pruning and fine-tuning pipeline was proposed by Han et al. (2015), which used the parameter magnitude as the pruning criterion. Narang et al. (2017) extended this pipeline to prune the recurrent neural networks with a complicated pruning strategy. Molchanov et al. (2016) introduced first-order Taylor term as the pruning criterion and conduct global pruning. Li et al. (2016) used ℓ_1 regularization to force the unimportant parameters to zero.

Sparse Neural Network Training: Recently, some works attempt to find the sparse network directly during the training process without the pruning and fine-tuning stage. Inspired by the growth and extinction of neural cell in biological neural networks, Mocanu et al. (2018) proposed a prune-regrowth procedure called Sparse Evolutionary Training (SET) that allows the pruned neurons and connections to revive randomly. However, the sparsity level need to be set manually and the random recover of network connections may provoke unexpected effects to the network. DEEP-R proposed by Bellec et al. (2017) used Bayesian sampling to decide the pruning and regrowth configuration, which is computationally expensive. Dynamic Sparse Reparameterization (Mostafa & Wang, 2019) used dynamic parameter reallocation to find the sparse structure. However, the pruning threshold can only get halved if the percentage of parameter pruned is too high or get doubled if that percentage is too low for a certain layer. This coarse-grained adjustment of pruning threshold significantly limits the ability of Dynamic Sparse Reparameterization. Additionally, a predefined pruning ratio and fractional tolerance are required. Dynamic Network Surgery (Guo et al., 2016) proposed pruning and splicing procedure that can prune or recover network connections according to the parameter magnitude but it requires manually determined thresholds that are fixed during the sparse learning process. This layer-wise thresholds are extremely hard to manually set. Meanwhile, Fixing the thresholds makes it hard to adapt to the rapid change of parameter importance. Dettmers & Zettlemoyer (2019) proposed sparse momentum that used the exponentially smoothed gradients as the criterion for pruning and regrowth. A fixed percentage of parameters are pruned at each pruning step. The pruning ratio and momentum scaling rate need to be searched from a relatively high parameter space.

Instead, our method can dynamically and automatically perform parameter pruning for each layer during the training process without pre-defined pruning percentages. Meanwhile, the pruning thresholds will have fine-grained adjustments dynamically via back-propagation. Consequently, the process of finding optimal sparse network structure is naturally embedded into the training process, which enables our method to get well-performed models with high sparsity using the same epochs as training the dense models. Additionally, the pattern of layer-wise sparsity in our method reveals useful information to guide the design of compact network architectures.

3 Dynamic Sparse Training

3.1 Notation

Deep neural network consists of a set of parameters $\{\boldsymbol{W}_i : 1 \leq i \leq C\}$, where $\boldsymbol{W}_i$ denotes the parameter matrix at layer i and C denotes the number of layers in this network. For each fully connected layer and recurrent layer, the corresponding parameter is $\boldsymbol{W}_i \in \mathbb{R}^{c_o \times c_i}$, where c_o is the output dimension and c_i is the input dimension. For each convolutional layer, there exists a convolution kernel $\mathbf{K}_i \in \mathbb{R}^{c_o \times c_i \times w \times h}$, where c_o is the number of output channels, c_i is the number of input channels, w and h are the kernel sizes. Each filter in a convolution kernel $\mathbf{K}_i$ can be flattened to a vector. Therefore, a corresponding parameter matrix $\boldsymbol{W}_i \in \mathbb{R}^{c_o \times z}$ can be derived from each convolution kernel $\mathbf{K}_i \in \mathbb{R}^{c_o \times c_i \times w \times h}$, where $z = c_i \times w \times h$. Actually, the pruning process is equivalent to finding a binary parameter mask $\boldsymbol{M}_i$ for each parameter matrix $\boldsymbol{W}_i$. Thus, a set of binary parameter masks $\{\boldsymbol{M}_i : 1 \leq i \leq C\}$ will be found by network pruning. Each element for these parameter masks $\boldsymbol{M}_i$ is either 1 or 0.

3.2 Dynamic parameter mask

Instead of pruning a pre-trained model, we define and store a binary parameter mask $\boldsymbol{M}$ for each parameter matrix $\boldsymbol{W}$. This binary parameter mask preserves the information about the sparse structure, which enables joint optimization of the network parameter and sparse network structure. Given the parameter set $\{\boldsymbol{W}_1, \boldsymbol{W}_2, \cdots, \boldsymbol{W}_C\}$, we will dynamically find the corresponding parameter masks $\{\boldsymbol{M}_1, \boldsymbol{M}_2, \cdots, \boldsymbol{M}_C\}$. To achieve this, for each parameter matrix $\boldsymbol{W}_i \in \mathbb{R}^{c_o \times c_i}$, a trainable pruning threshold is defined. Then we utilize a unit step function $S(x)$ as shown in Figure 2(a) to get the masks with the absolute value of parameter and corresponding threshold. For each parameter matrix $\boldsymbol{W} \in \mathbb{R}^{c_o \times c_i}$, there are three different choices for the threshold with flexible granularity, namely a scalar threshold t, a vector threshold $\boldsymbol{t} \in \mathbb{R}^{c_o}$ and a matrix threshold $\boldsymbol{T} \in \mathbb{R}^{c_o \times c_i}$ as present below.

$$\boldsymbol{Q}_{ij} = F(\boldsymbol{W}_{ij}) = \begin{cases} |\boldsymbol{W}_{ij}| - t & \text{for scalar threshold } t \\ |\boldsymbol{W}_{ij}| - \boldsymbol{t}_i & \text{for vector threshold } \boldsymbol{t} \\ |\boldsymbol{W}_{ij}| - \boldsymbol{T}_{ij} & \text{for matrix threshold } \boldsymbol{T} \end{cases} \tag{1}$$

$$\boldsymbol{M} = S(\boldsymbol{Q}) \tag{2}$$

The unit step function $S(x)$ is applied element-wise. To balance the overhead and the benefit involved by the additional trainable threshold, the vector threshold $\boldsymbol{t}$ is chosen for fully connected, convolutional and recurrent layers. The detailed analysis is present in Appendix A.2

3.3 Trainable masked layers

With this trainable threshold, the trainable masked fully connected, convolutional and recurrent layer are introduced as shown in Figure 1, where $\boldsymbol{X}$ is the input of current layer and $\boldsymbol{Y}$ is the output. For fully connected and recurrent layers, the masked parameter $\boldsymbol{W} \odot \boldsymbol{M}$ will be used in the batched matrix multiplication, where $\odot$ denote Hadamard product operator. For convolutional layers, each convolution kernel $\mathsf{K} \in \mathbb{R}^{c_o \times c_i \times w \times h}$ can be flatten to get $\boldsymbol{W} \in \mathbb{R}^{c_o \times z}$. Therefore, we can get the masked kernel using similar process as fully connected layers and use this masked kernel for subsequent convolution operation.

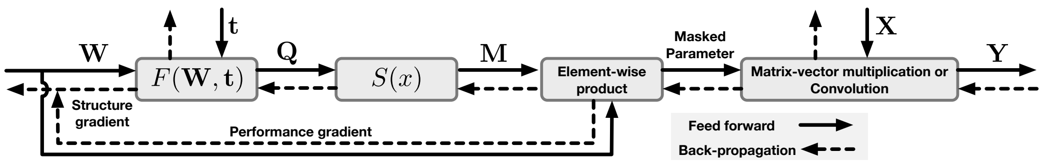


Figure 1: Detailed structure of trainable masked layer

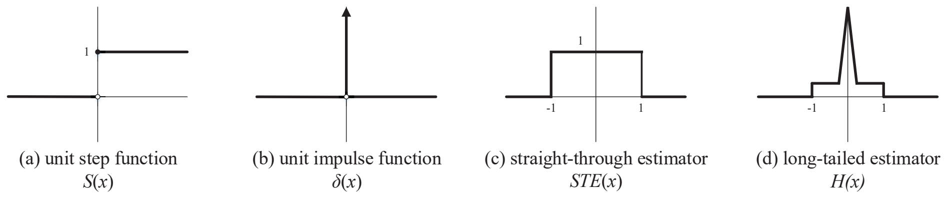


Figure 2: The unit step function $S(x)$ and its derivative approximations.

Since we want to make the threshold $\boldsymbol{t}$ trainable via back-propagation, the derivative of $S(x)$ is required. However its derivative is an impulse function whose value is zero almost everywhere and infinite at zero, as shown in Figure 2(b). Thus it cannot be applied in back-propagation and parameter updating directly. Some previous works (Hubara et al. (2016); Rastegari et al. (2016); Zhou et al. (2016)) demonstrated that by providing a derivative estimation, it is possible to train networks containing such binarization function. A clip function called straight through estimator (STE) (Bengio et al., 2013) was employed in these works and is illustrated in Figure 2(c).

Furthermore, Xu & Cheung (2019) discussed the derivative estimation in balance of tight approximation and smooth back-propagation. We adopt this long-tailed higher-order estimator $H(x)$ in our training, shown in Figure 2(d). It has a wide active range between $[-1, 1]$ with non-zero gradient to avoid gradient vanishing during training. On the other hand, the gradient value near zero is a piecewise polynomial function giving tighter approximation than STE. The estimator is represented as

$$\frac{d}{dx}S(x) \approx H(x) = \begin{cases} 2 - 4|x|, & -0.4 \le x \le 0.4 \\ 0.4, & 0.4 < |x| \le 1 \\ 0, & otherwise \end{cases} \tag{3}$$

With this derivative estimator, the threshold can be trained via back-propagation. Therefore, the percentage of parameter pruned in each layer is adjusted dynamically according to the training loss. The underlying parameters can be preserved and the masked parameters are still able to be updated during the back-propagation, which enables previously pruned network connection to be recovered. As demonstrated in Section 4.2, previously unimportant parameter may turn to be important during the training process, this ability to splice the pruned connections is crucial to find the optimal pruning masks. Meanwhile, in trainable masked layers, the network parameter $\boldsymbol{W}$ can receive two branches of gradient, namely the performance gradient for better model performance and the structure gradient for better sparse structure, which helps to properly update the network parameter under sparse network connectivity.

3.4 Sparse regularization term

Now that the threshold of each layer is trainable, higher percentage of pruned parameter is desired. To get the parameter masks $\boldsymbol{M}$ with high sparsity, higher pruning thresholds are needed. To achieve this, we add a sparse regularization term L_s to the training loss that penalizes the low threshold value. For a trainable masked layer with threshold $\boldsymbol{t} \in \mathbb{R}^{c_o}$, the corresponding regularization term is $R = \sum_{i=1}^{c_o} \exp(-\boldsymbol{t}_i)$. Thus, the sparse regularization term L_s for the a deep neural network with C trainable masked layers is:

$$L_s = \sum_{i=1}^{C} R_i \tag{4}$$

We use $\exp(-x)$ as the regularization function since it is asymptotical to zero as x increases. Consequently, it penalizes low thresholds without encouraging them to become extremely large.

3.5 Dynamic sparse training

The traditional fully connected, convolutional and recurrent layers can be replaced with the corresponding trainable masked layers in deep neural networks. Then we can train a sparse neural network directly with back-propagation algorithm given the training dataset $\mathcal{D} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \cdots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$, the network parameter $\boldsymbol{W}$ and the layer-wise threshold $\boldsymbol{t}$ as follows:

$$\mathcal{J}(\boldsymbol{W}, \boldsymbol{t}) = \frac{1}{N}\left(\sum_{i=1}^{N} \mathcal{L}((\boldsymbol{x}_i, \boldsymbol{y}_i); \boldsymbol{W}, \boldsymbol{t})\right) + \alpha L_s \tag{5}$$

$$\boldsymbol{W}^*, \boldsymbol{t}^* = \underset{\boldsymbol{W}, \boldsymbol{t}}{\arg\min}\, \mathcal{J}(\boldsymbol{W}, \boldsymbol{t}) \tag{6}$$

where $\mathcal{L}(\cdot)$ is the loss function, e.g. cross-entropy loss for classification and α is the scaling coefficient for the sparse regularization term, which is able to control the percentage of parameter remaining. The sparse regularization term L_s tends to increase the threshold $\boldsymbol{t}$ for each layer thus getting higher model sparsity. However, higher sparsity tends to increase the loss function, which reversely tends to reduce the threshold and level of sparsity. Consequently, the training process of the thresholds can be regarded as a contest between the sparse regularization term and the loss function in the sense of game theory. Therefore, our method is able to dynamically find the sparse structure that properly balances the model sparsity and performance.

4 Experiments

The proposed dynamic sparse training method is evaluated on MNIST and CIFAR-10 with various modern network architectures. To quantify the pruning performance, the layer remaining ratio is defined to be $k_l = n/m$, where n is the number of element equal to 1 in the mask $\boldsymbol{M}$ and m is the total number of element in $\boldsymbol{M}$. The model remaining ratio k_m is the overall ratio of non-zero element in the parameter masks for all trainable masked layers. On all the tested architectures, our method yields highly sparse models with minimal accuracy drop. Meanwhile, despite the simplicity of our method, the dynamic change of layer remaining ratios during the training procedure and the consistent sparse pattern under different model remaining ratios provide strong evidence to the effectiveness of our method. For all trainable masked layers, the trainable thresholds are initialized to zero since it is assumed that originally the network is dense. The detailed experimental setup is present in Appendix A.1.

4.1 Sparse learning on various modern architectures

Our method can be naturally and generally applied to almost all modern deep learning architectures that consist of fully connected, convolutional and recurrent layers. The proposed method is tested on popular modern networks with the same training epoch and hyperparameter configuration as training the dense baselines despite the additional α. Table 1 presents the performance of our method on various network architectures. The model remaining percentage is defined as $k_m \times 100\%$. From Table 1 we can conclude that on all these tested architectures, the proposed approach is able to prune considerably high portion of model parameter ($> 95\%$) without significant performance loss ($< 1\%$). For architectures like Lenet-5-Caffe and LSTM, the sparse models can achieve almost the same accuracy as dense baselines with less than 2% parameter remained.

Dataset	Architecture	Model Remaining Percentage (%)	Sparse Accuracy (%)	Dense Accuracy (%)
MNIST	Lenet-300-100	2.48±0.21	97.69±0.14	98.16±0.06
	Lenet-5-Caffe	1.64±0.13	99.11±0.07	99.18±0.05
	LSTM-a	1.54±0.04	98.56±0.06	98.64±0.12
	LSTM-b	0.74±0.02	98.62±0.12	98.87±0.07
CIFAR-10	VGG-16	3.76±0.53	93.02±0.37	93.75±0.23
	WideResNet-16-8	4.64±0.15	94.73±0.11	95.18±0.06
	WideResNet-16-10	4.73±0.21	94.79±0.16	95.37±0.13
	WideResNet-28-8	4.55±0.16	94.80±0.09	95.65±0.11

Table 1: The level of remaining percentage achieved without obvious performance drop on various modern architectures. The result is averaged over 5 runs. Sparse and dense accuracy are the test accuracy on corresponding sparse and dense models. All LSTM models have two LSTM layers with hidden size 128 for LSTM-a and 256 for LSTM-b

Architecture	Method	Model Remaining Percentage (%)	Accuracy (Dense→Sparse %)	Accuracy Reduction (%)
VGG-16	Sparse Momentum	5	93.51→93.00	-0.51
	Ours	**3.76**	93.75→**93.02**	-0.73
WideResNet-16-8	DEEP-R	5	95.21→93.16	-2.05
	SET	5	95.21→93.98	-1.23
	DSR	5	95.21→94.68	-0.53
	Sparse Momentum	5	95.43→94.38	-1.05
	Ours	**4.67**	95.18→**94.73**	**-0.45**

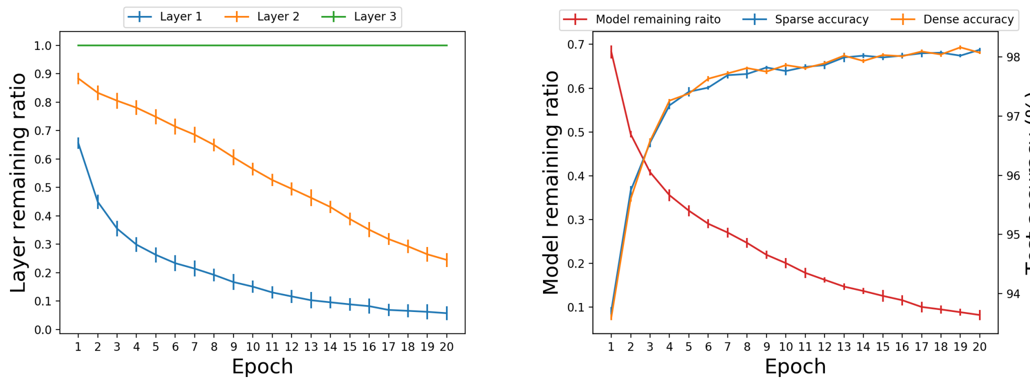
Table 2: Comparison with other sparse training methods on CIFAR-10.

To further evaluate the proposed approach, we compare our method with other sparse learning algorithms on CIFAR-10 as presented in Table 2. The state-of-the-art algorithms, DEEP-R (Bellec et al., 2017), SET (Mocanu et al., 2018), DSR (Mostafa & Wang, 2019), and Sparse momentum (Dettmers & Zettlemoyer, 2019), are selected for comparsion. Since different methods have different training hyperparameter configurations, we list the model remaining percentage, accuracy change from dense to sparse model and the corresponding accuracy reduction. On VGG-16, despite the relatively high accuracy reduction compared with Sparse Momentum, our method achieves better test accuracy with 1.24% more parameter pruned. On WideResNet-16-8 (Zagoruyko & Komodakis, 2016), our method realizes the best results on all the criterion. Besides the ability to get better sparse models compared to other methods, we present two interesting observations about the layer remaining ratio that provide strong evidence to the efficiency and effectiveness of our algorithm.

4.2 Dynamic and abrupt change of layer remaining ratio

For multilayer neural networks, the parameters in different layers will have different relative importance. For example, Lenet-300-100 has three fully connected layers. Changing the parameter of the last layer (layer 3) can directly affect the model output. Thus, the parameter of layer 3 should be pruned more carefully. The input layer (layer 1) has the largest amount of parameters and takes the raw image as input. Since the images in MNIST dataset consist of many unchanged pixels (the background) that have no contribution to the classification process, it is expected that the parameters that take these invariant background as input can be pruned safely. Therefore, the remaining ratio should be the highest for layer 3 and the lowest for layer 1 if a Lenet-300-100 model is pruned. To check the pruning effect of our method on these three layers, Lenet-300-100 model is sparsely trained by the default hyperparameters setting present in Appendix A.1. The pruning trends of these three layers during dynamic sparse training is present in Figure 3(a). During the whole sparse training process, the remaining ratios of layer 1 and layer 2 keep decreasing and the remaining ratio of layer 3 maintains to be 1. The remaining ratio of layer 1 is the lowest and decrease to less that 10% quickly, which is consistent with the expectation.

Furthermore, as presented in Figure 3(b), instead of decreasing by manually set fixed step size as in traditional pruning methods, the proposed method makes the model remaining ratio decrease smoothly and continuously during training. In the meantime, the test accuracy of the sparse model is almost the same as original accuracy on dense model in the whole training process. This indicates that our method can properly balance the model remaining ratio and the model performance by continuous fine-grained parameter pruning throughout the entire sparse training procedure. During dynamic sparse training of Lenet-300-100, the model remaining percentage continues to decrease to 7.87% and the best sparse accuracy achieved is 98.12%, which is almost the same compared with 98.16% for the dense counterpart. Similar training tendency can be observed in other network architectures with various α. The training curve of VGG-16 on CIFAR-10 is shown in Figure 4(b), where the best sparse test accuracy (93.93%) is even higher than test accuracy (93.75%) of dense baseline with only 8.82% parameters remained in the sparse model. The detailed results for other architectures are present in Appendix A.3.



(a) Remaining ratio of each layer

(b) Model remaining ratio and accuracy

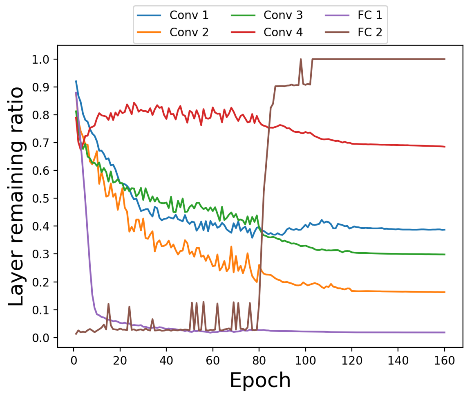
Figure 3: Change of layer remaining ratio for LeNet-300-100 with $\alpha = 0.0005$. The result is averaged over 10 runs with different random seeds. The difference within this 10 runs is quite limited, which indicates the robustness of our method

Figure 4(a) shows the change of the remaining ratios for the first four convolutional layers and last two fully connected layers of VGG-16 during dynamic sparse training. On this figure there are some interesting observations. Overall, the remaining ratios tend to decrease for most layers. However instead of decreasing monotonically, the remaining ratios of all layers fluctuate dynamically during the sparse training, which is in line with expectations since we assume that the relative parameter importance will change actively during sparse training process.

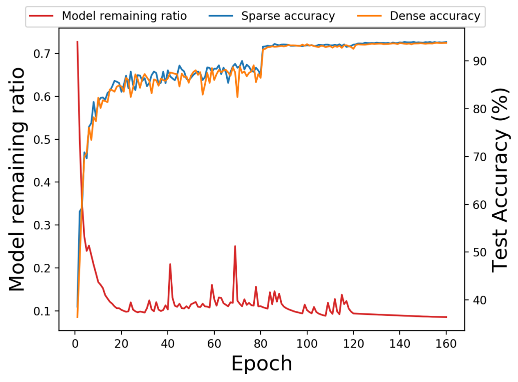
Similar to the layer 3 of Lenet-300-100, the second fully connected layer (FC 2) of VGG-16 is the output layer, hence its remaining ratio is expected to be relatively high. The result in Figure 4(a) also confirms this assumption. But a surprising observation is that the remaining ratio of FC 2 is quite low at around 0.1 for the first 80 epochs and increases almost immediately just after the 80 epoch.

We suppose that this is caused by the decay of learning rate from 0.1 to 0.01 at 80 epoch. Before the learning rate decay, the layers preceding to the output layer fail to extract enough useful features for classification due to the relatively coarse-grained parameter adjustment incurred by high learning rate. This means that the corresponding parameters that take those useless features as input can be pruned, which leads to the low remaining ratio of the output layer. The decayed learning rate enables fine-grained parameter update that makes the neural network model converge to the local minimal quickly (Kawaguchi, 2016), where most of the features extracted by the preceding layers turn to be helpful for the classification. This makes previously unimportant network connections in the output layer become important thus the remaining ratio of this layer get abrupt increase. There are two facts that support our assumptions for this phenomenon. Firstly, the test accuracy fluctuates below 90% for many epochs before the learning rate decay. However, it increases suddenly just after the learning rate decay and keeps almost the same afterward. Secondly, the remaining ratios of the preceding convolutional layers are almost unchanged after the remaining ratio of the output layer increases up to 1, which means that the remaining parameters in these layers are necessary to find the critical features.

Similar abrupt changes of the layer remaining ratios are observed in almost all the complex modern network architectures as presented in Appendix A.3 This surprising fact indicates that the importance of parameter will be affected by many factors and may change dramatically. Consequently, the ability to detect the abrupt parameter importance change and swiftly recover network connectivity is of high significance. Our method can properly handle this abrupt change with the timely gradient feedback from the training loss and fine-grained adjustments of layer-wise pruning thresholds.



(a) Remaining ratio of each layer



(b) Model remaining ratio and accuracy

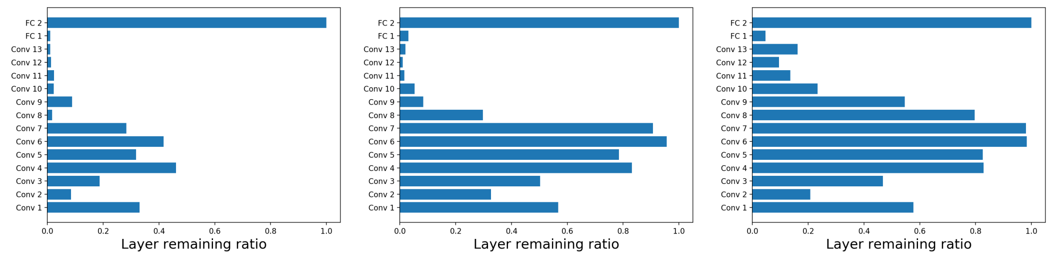
Figure 4: The change of layer remaining ratios and for VGG-16 with $\alpha = 5 \times 10^{-6}$ is present in (a). The change of corresponding sparse accuracy during the dynamic sparse training and dense accuracy during normal training is present in (b).

4.3 Information revealed from consistent sparse pattern

Many works have tried to design more compact model with mimic performance to over-parameterized models (He et al., 2016; Howard et al., 2017; Zhang et al., 2018). Network architecture search has been viewed as the future paradigm of deep neural network design. However, it is extremely difficult to determine whether the designed layer consists of redundancy. Therefore, typical network architecture search methods rely on evolutionary algorithms (Liu et al., 2017) or reinforcement learning (Baker et al., 2016), which is extremely time-consuming and computationally expensive. Network pruning can actually be reviewed as a kind of architecture search process (Liu et al., 2018b; Frankle & Carbin, 2018) thus the sparse structure revealed from network pruning may provide some guidance to the network architecture design. However, The layer-wise equal remaining ratio generated by the unified pruning strategy fails to indicate the different degree of redundancy for each layer. And the global pruning algorithm is non-robust, which fails to offer consistent guidance.

Here we demonstrate another interesting observation called consistent sparse pattern during dynamic sparse training that provides useful information about the redundancy of individual layers as the guidance for compact network architecture design. For the same architecture trained by our method with various α values, the relative relationship of sparsity among different layers keeps consistent. The sparse patterns of VGG-16 on CIFAR-10 are present in Figure 5 with three different α.

In all configurations, the last four convolutional layers (Conv 10-13) and the first fully connected layers (FC 1) are highly sparse. Meanwhile, some layers (Conv 3-7, FC 2) keep a high amount of parameters after pruning. This consistent sparse pattern indicates that these heavily pruned layers consist of high redundancy and a large number of parameters can be reduced in these layers to get more compact models. This phenomenon also exists in other network architectures, which is present in details in the appendix A.4. Therefore, with this consistent sparse pattern, after designing a new network architecture, our method can be applied to get useful information about the layer-wise redundancy in this new architecture.

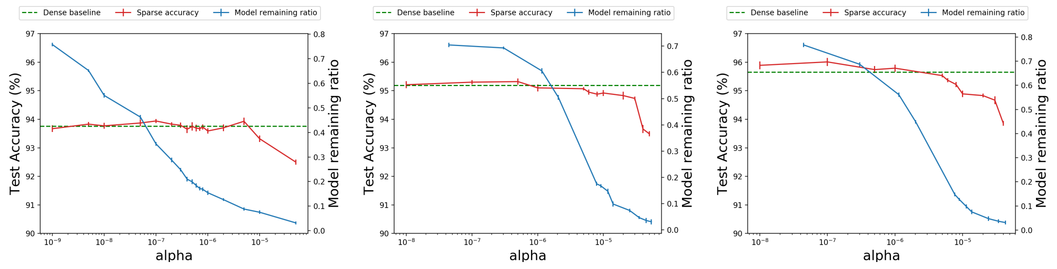


(a) $\alpha = 10^{-5}$, 6.66% remaining (b) $\alpha = 10^{-6}$, 15.47% remaining (c) $\alpha = 10^{-7}$, 35.35% remaining

Figure 5: The sparse pattern and percentage of parameter remaining for different choices of α on VGG-16

4.4 Model performance on various remaining ratio

By varying the scaling coefficient α for sparse regularization term, we can control the model remaining ratios of sparse models generated by dynamic sparse training. The relationships between α, model remaining ratio and sparse model accuracy of VGG, WideResNet-16-8 and WideResNet-28-8 are presented in Figure 6. As demonstrated, the model remaining ratio keeps decreasing with increasing α. With a moderate α value, it is easy to obtain a sparse model with comparable or even slightly higher accuracy than the dense counterpart. On the other side, if the α value is too large that makes the model remaining percentage less than 5%, there is a noticeable accuracy drop. Interestingly, this accuracy drop usually emerges together with the break of consistent sparse pattern, which means that large α forces the algorithm to mask the critical parameters, thus degrades the model performance. As demonstrated in Figure 6, the choice of α ranges from 10^{-9} to 10^{-4}. Depending on the application scenarios, we can either get models with similar or better performance as dense counterparts by a relatively small α or get a highly sparse model by a larger α.



(a) VGG-16 (b) WideResNet-16-8 (c) WideResNet-28-8

Figure 6: Test accuracy of sparse model on CIFAR-10 and model remaining ratio for different α

5 Conclusion

We propose Dynamic Sparse Training (DST) with trainable masked layers that enables direct training of sparse models with trainable pruning thresholds. DST can be easily applied to various types of neural network layers and architectures to get highly sparse or better performed model than dense counterparts. With the ability to reveal consistent sparse pattern, our method can also provide useful guidance to the design of more compact network.

References

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. *arXiv preprint arXiv:1711.05136*, 2017.

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

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

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

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pp. 1379–1387, 2016.

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, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1389–1397, 2017.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 304–320, 2018.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pp. 4107–4115. 2016.

Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in neural information processing systems*, pp. 586–594, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. Accelerating convolutional networks via global & dynamic filter pruning. In *IJCAI*, pp. 2425–2432, 2018.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018a.

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

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pp. 5058–5066, 2017.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2498–2507. JMLR. org, 2017.

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.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11264–11272, 2019.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. *arXiv preprint arXiv:1902.05967*, 2019.

Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. Exploring sparsity in recurrent neural networks. *arXiv preprint arXiv:1704.05119*, 2017.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, pp. 525–542. Springer, 2016.

Xavier Suau, Luca Zappella, Vinay Palakkode, and Nicholas Apostoloff. Principal filter analysis for guided network compression. *arXiv preprint arXiv:1807.10585*, 2018.

Zhe Xu and Ray C. C. Cheung. Accurate and compact convolutional neural networks with trained binarization. In *British Machine Vision Conference (BMVC)*, 2019.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9194–9203, 2018.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856, 2018.

Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

A Appendix

A.1 Experimental setup

MNIST: LeNet-300-100 and LeNet-5-Caffe are trained using SGD with momentum of 0.9 and batch size of 64 with 20 training epochs. The learning rate is 0.01 without learning rate decay. Meanwhile, the default scaling factor α for sparse regularization term is 0.0005 for both network architectures. LSTM models are trained using Adam optimization scheme (Kingma & Ba, 2014) with default Adam hyperparameter setting for 20 epochs. The batch size is 100 and the default learning rate is 0.001. Meanwhile, the default scaling factor α for sparse regularization term is 0.001.

CIFAR-10 Models on CIFAR-10 are trained using SGD with momentum 0.9 and batch size of 64 with 160 training epochs. The initial learning rate is 0.1 and decayed by 0.1 at 80 and 120 epoch. The default scaling factor α for sparse regularization term is 5×10^{-6} for all tested architectures.

For all trainable masked layers, the trainable thresholds are initialized. Additionally, we find that extremely high scaling coefficient α will make the sparse regularization term dominates the training loss thus making the mask $\boldsymbol{M}$ to be all zero in certain layer, which may impede the training process. To prevent this, the pruning threshold $\boldsymbol{t}$ will be reset to zero if more than 99% elements in the mask are zero in a certain layer despite the LSTM models. This mechanism makes the training process smoother and enables a wider range choice of α.

A.2 Analysis of different threshold choices

All the three choices of trainable thresholds are tested on various architectures. Considering the effects on network pruning, the matrix threshold has almost the same model remaining ratio compared with the vector threshold. The scalar threshold is less robust than the vector and matrix threshold. In terms of the storage overhead, the matrix threshold almost doubles the amount of parameter during the training process in each architectures. Meanwhile, the vector threshold only adds less than 1% additional network parameter.

The extra trainable threshold will also bring additional computation. In both feed forward and back-propagation phase, the additional computations are matrix-element-wise operations ($\mathcal{O}(n^2)$), that is apparently light-weighted compared to the original batched matrix multiplication ($\mathcal{O}(n^3)$). For practical deployment, only the masked parameter $\boldsymbol{W} \odot \boldsymbol{M}$ need to be stored, thus no overhead will be introduced. Therefore, considering the balance between the overhead and the benefit incurred by these three choices, the vector threshold is chosen for trainable masked layers.

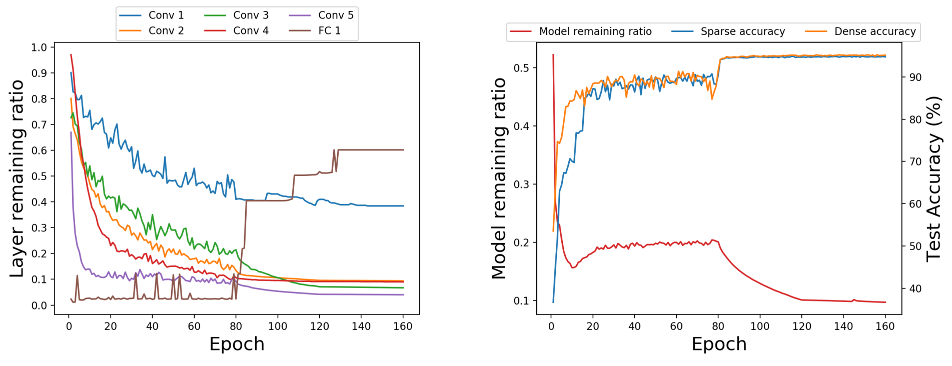
A.3 Change of remaining ratio in other network architectures

Here we present the change of remaining ratios during dynamic sparse training for the other tested architectures. Since WideResNet models only have one fully connected layer at the end as the output layer, the first five convolutional layers and the last fully connected layer are present. Figure 7 demonstrates the corresponding result for WideResNet-16-8. Figure 8 demonstrates the corresponding result for WideResNet-16-10. And Figure 9 demonstrates the corresponding result for WideResNet-28-8.

The similar phenomenon as described in section 4.2 can be observed in all these three network architectures for various α.

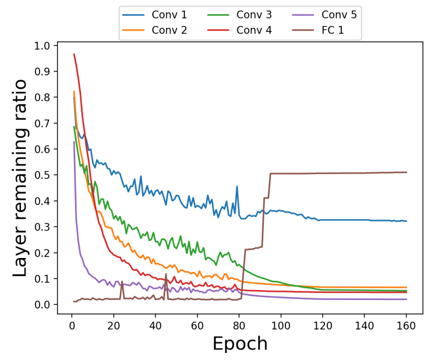
A.4 Consistent sparse pattern in other network architectures

Here we present the consistent sparse pattern for the other tested architectures. Figure 10 demonstrates the corresponding result for WideResNet-16-8. And Figure 11 demonstrates the corresponding result for WideResNet-16-10.

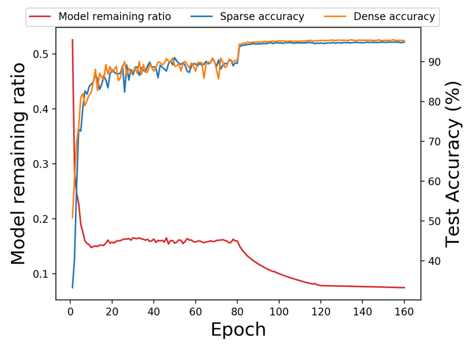


(a) Remaining ratio of each layer (b) Model remaining ratio and accuracy

Figure 7: WideResNet-16-8 trained by dynamic sparse training with $\alpha = 10^{-5}$

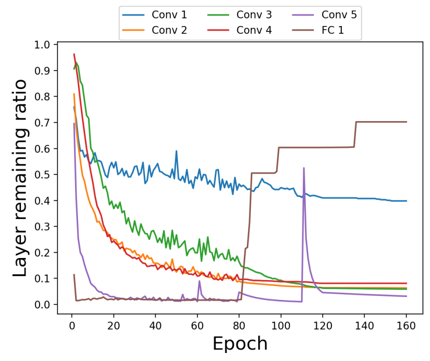


(a) Remaining ratio of each layer

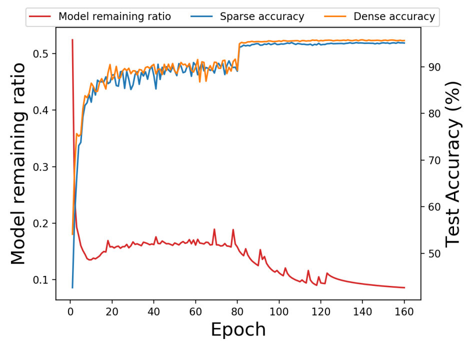


(b) Model remaining ratio and accuracy

Figure 8: WideResNet-16-10 trained by dynamic sparse training with $\alpha = 10^{-5}$

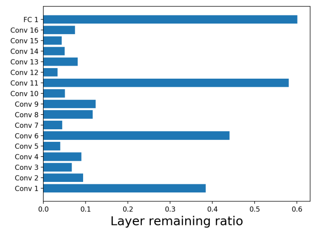


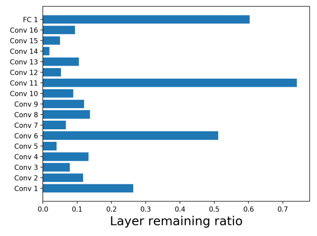
(a) Remaining ratio of each layer

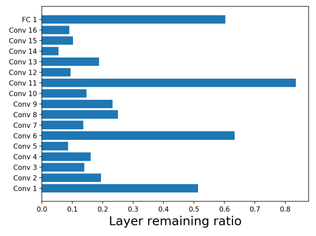


(b) Model remaining ratio and accuracy

Figure 9: WideResNet-28-8 trained by dynamic sparse training with $\alpha = 10^{-5}$

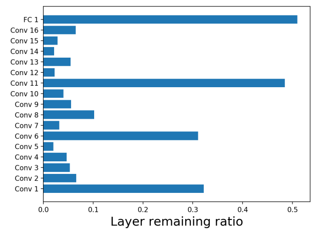


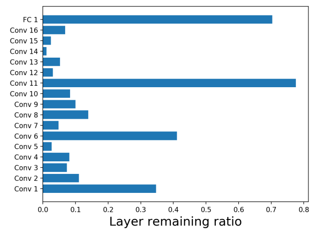


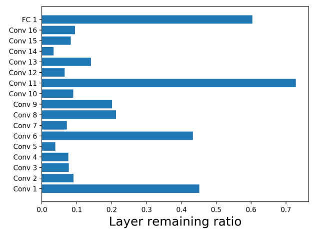


(a) $\alpha = 10^{-5}$, 9.86% remaining (b) $\alpha = 8 \times 10^{-6}$, 14.47% remaining (c) $\alpha = 5 \times 10^{-6}$, 19.18% remaining

Figure 10: The sparse pattern and percentage of parameter remaining for different choices of α on WRN-16-8







(a) $\alpha = 10^{-5}$, 7.55% remaining (b) $\alpha = 8 \times 10^{-6}$, 10.47% remaining (c) $\alpha = 5 \times 10^{-6}$, 14.75% remaining

Figure 11: The sparse pattern and percentage of parameter remaining for different choices of α on WRN-16-10