Dynamic Forward and Backward Sparse Training (DFBST): Accelerated Deep Learning through Completely Sparse Training Schedule

Tejas Pote TEJASPOTE915@GMAIL.COM
Indian Institute of Technology, Kharagpur
Muhammad Athar Ganaie MATHARG7@GMAIL.COM
Indian Institute of Technology, Kharagpur
Atif Hassan ATIF.HIT.HASSAN@GMAIL.COM
Indian Institute of Technology, Kharagpur
Swanand Khare SRKHARE@MATHS.IITKGP.AC.IN
Indian Institute of Technology, Kharagpur

Editors: Emtiyaz Khan and Mehmet Gönen

Abstract

Neural network sparsification has received a lot of attention in recent years. A number of dynamic sparse training methods have been developed that achieve significant sparsity levels during training, ensuring comparable performance to their dense counterparts. However, most of these methods update all the model parameters using dense gradients. To this end, gradient sparsification is achieved either by non-dynamic (fixed) schedule or computationally expensive dynamic pruning schedule. To alleviate these drawbacks, we propose Dynamic Forward and Backward Sparse Training (DFBST), an algorithm which dynamically sparsifies both the forward and backward passes using trainable masks, leading to a completely sparse training schedule. In contrast to existing sparse training methods, we propose separate learning for forward as well as backward masks. Our approach achieves state of the art performance in terms of both accuracy and sparsity compared to existing dynamic pruning algorithms on benchmark datasets, namely MNIST, CIFAR-10 and CIFAR-100.

Keywords: Sparse weight update, completely sparse training schedule.

1. Introduction

Modern deep neural networks have achieved phenomenal performance over a wide variety of tasks in areas like computer vision, natural language processing, recommendation systems and many more. Most of these networks involve training dense overparameterized models, resulting in significantly large number of parameters and subsequently large computational overhead. While overparameterization may enhance inference and generalization of the network, the additional memory footprints and computational efforts are raising energy concerns. This has motivated a plethora of works around model sparsification techniques that drop the redundant model parameters. Sparse models have been able to deliver comparable performance with respect to their dense counterparts in a number of memory constrained applications.

Much of the work in this domain has centered around finding a subset of weights in numerous ways, for instance based on some saliency criterion (Lee et al. (2019)), low rank approximation (Novikov et al. (2015)) for the weight matrices or sparsity inducing regularization functions (Scardapane et al. (2017)). The Lottery Ticket Hypothesis (Frankle and Carbin (2019)) proves the existence of subnetworks within dense, randomly initialized networks which give comparable performance when trained in isolation. However, it uses iterative pruning and fine-tuning to find the subnetworks, which turn out to be computationally expensive. To account for this, a number of methods have been proposed recently wherein the sparse structure is determined during training without the requirement of re-training. Many of these methods have been able to achieve dense accuracy levels with sufficiently high sparsity.

Besides pruning of components used for inference (neurons/filters), there exist many other sparsification methods that are aimed at sparsification of gradients. Most of these gradient sparsification techniques use either computationally expensive operations like retaining top-K gradients (Sun et al. (2017)) or predefined fixed pruning schedules (Aji and Heafield (2017)). meProp (Sun et al. (2017)) observes that a very small proportion of gradient updates are sufficient to obtain performance comparable to dense baselines. It would thus be desirable to have a technique which can assess gradient importance within individual layers and prune them dynamically during the training.

In general, almost all the existing model sparsification techniques focus on pruning either the components of forward pass or backward pass. Although a few methods try to achieve completely sparse training schedules (i.e. sparse forward and backward pass), they use the same sparse structure for both the forward and backward pass (Zhou et al. (2021)). This would mean assuming that only those gradients corresponding to the most important weights are significant or vice-versa; which need not be necessarily true.

We suggest a novel approach which dynamically sparsifies the weights as well as the gradients between the training epochs. We use neuron/filter-wise trainable thresholds similar to Dynamic Sparse Training (Liu et al. (2020)) for masking both weights and gradients. We include a variance penalty term in the loss function so that thresholds for gradients are incorporated in the computational graph and updated during the backward pass. These thresholds learn according to the changing gradient importance during training.

1.1. Our Contribution

Following are the key contributions of this article:

- We propose Dynamic Forward and Backward Sparse Training (DFBST), in which both weights and gradients of the network are sparsified separately by binary masks generated using trainable thresholds.
- The incorporation of variance of masked gradients in the loss function penalizes gradients with high variance and facilitates updates for gradient thresholds.
- We conduct our experiments on three benchmark datasets namely MNIST, CIFAR-10 and CIFAR-100. We obtain state of the art performance with significant sparsity.

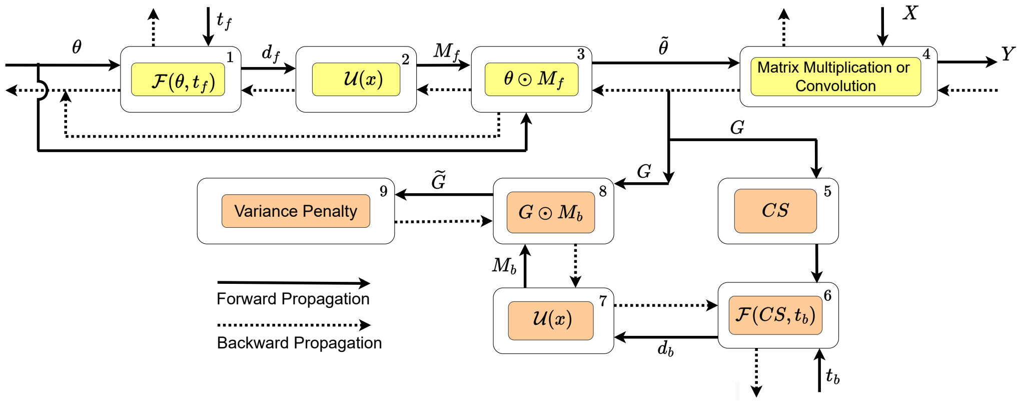

Figure 1: Detailed Structure of a single layer in DFBST. Our forward pass works on magnitude pruning with a trainable forward mask. Here, $\mathcal{F}$ returns the difference of absolute values of its inputs, and $\mathcal{U}$ is the unit step function. The mask is generated using trainable forward thresholds and the masked gradients are fed into the network. In the backward pass, we generate a trainable backward mask for the gradients. We include the variance penalty term in the loss function. The weights receive updates using the masked gradients.

2. Related Work

The early phase of sparsification has predominantly been focused on dense-to-sparse training. (Thimm and Fiesler (1995)) suggested magnitude pruning to be an effective technique. (Han et al. (2016)) proposed the method of training, fine-tuning and retraining using parameter magnitude as the pruning criterion. (Narang et al. (2017)) progressively grow network sparsity by using monotonically increasing thresholds that are determined using a set of hyperparameters. Apart from the magnitude of the weights, several other criteria have been used for pruning. For instance, (Mozer and Smolensky (1989)) consider the sensitivity of the loss with respect to the neurons, (LeCun et al. (1990)), (Hassibi et al. (1993)) use the Hessian matrix of the model to prune the network.

Another class of methods utilize regularization methods for enforcing sparsity. (Louizos et al. (2018)) use L_0 regularization while (Scardapane et al. (2017)) use a modified form of the lasso penalty, termed as sparse group lasso penalty. Apart from these weight pruning techniques, there exist structured sparsification methods like (Li et al. (2017)), (Molchanov et al. (2017)) that prune filters of the convolutional layers. (Novikov et al. (2015)) represent the dense weight matrices using tensor train decomposition having reduced number of parameters. Apart from these, a number of sparsification methods focus on sparsifying gradients, thereby resulting in partial weight updates. Gradient sparsification is one of the gradient compression techniques along with quantization and low-rank approximation, that are primarily used to reduce communication overheads in distributed data-parallel training. (Wangni et al. (2018a)) propose a random-k sparsification approach to deal with high gradient variance. (Sun et al. (2017)) perform sparsification of the gradient vectors by retaining the top-k elements. (Tsuzuku et al. (2018)) take into consideration the gradients with minimum variance. Although these methods achieve impressive results, they do not take into account the fact that the significance of gradients may differ across layers. Also, many methods have been described in machine learning literature that reduce variance (Gower et al. (2020)). (Flet-Berliac et al. (2020)) apply gradient variance regularization in order to achieve more stable policy gradients in reinforcement learning. (Wangni et al. (2018b)), (Hoefler et al. (2021)), (Johnson and Zhang (2013)) observe that gradient variance adversely affects SGD convergence.

Recently, there has been an impetus to the development of sparse training methods. (Dai et al. (2019)) propose a training schedule inspired by the biological brain consisting of three stages viz random seed architecture, followed by growth of the network and then pruning redundant connections and neurons based on magnitude. Sparse Evolutionary Training (SET) (Mocanu et al. (2018)) adopts magnitude pruning and the regrowth of weights happens randomly. Dynamic Sparse Reparameterization (DSR) (Mostafa and Wang (2019)) improves upon SET by using an adaptive global threshold for magnitude pruning and reallocating parameters across layers during training. RigL (Evci et al. (2020)) maintains a sparse structure throughout training which provides for pruning and regrowth of weights after specified intervals.

Sparse Networks from Scratch (Dettmers and Zettlemoyer (2019)) uses exponentially smoothed gradients (momentum) to identify layers and weights that significantly reduce inference error. Top-KAST (Jayakumar et al. (2020)) maintains constant forward and backward sparsity by keeping top D weights in the forward pass and additional top B gradients along with those of non-zero weights in the backward pass. Though it promises significant reduction in FLOPS, the top-K operation heavily increases computational cost. Dynamic Sparse Training (DST) (Liu et al. (2020)) utilize trainable pruning thresholds to dynamically obtain a sparse structure jointly during the optimization procedure.

3. DFBST

3.1. Notation

We denote the parameters of the fully-connected layers by θ. Let n_l be the number of neurons in the l^{th} layer, then $\theta_l \in \mathbb{R}^{n_l \times n_{l-1}}$ where $1 \leq l \leq L$ (L is the number of hidden layers). In case of convolutional layers, the kernel is denoted by $K_l \in \mathbb{R}^{c_o \times c_i \times w \times h}$ for the l^{th} layer where c_o is the number of output channels, c_i is the number of input channels and w and h are the kernel sizes. The filters are then flattened into a vector and the new parameters are obtained as $\theta_l \in \mathbb{R}^{c_o \times z}$ where $z = c_i \times w \times h$. The gradients for the respective layers are denoted by G_l. Our method generates two binary masks, one for forward pass and the other for backward pass. We denote the forward and backward masks for the l^{th} layer as M_{f_l} and M_{bl} respectively.

3.2. Forward Pass

The dynamic forward masks for each layer are generated by comparing the absolute value of the weights with neuron/filter wise absolute thresholds. We denote the difference between the absolute value of the weight matrix and its corresponding threshold vector as d_{f_l}.

$$\begin{aligned} d_{fl}(i,j) &= |\theta_l(i,j)| - |t_{fl}(i)|, \quad 1 \leq l \leq L,\ 1 \leq i \leq n_l \\ M_{fl}(i,j) &= \mathcal{U}\left(d_{fl}(i,j)\right), \quad 1 \leq j \leq n_{l-1} \end{aligned}$$

Here $\mathcal{U}(\cdot)$ is the Unit Step function defined as follows.

$$\mathcal{U}(x) = \begin{cases} 1, & x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Similar to DST, $t_f \in \mathbb{R}^{n_l}$ is a neuron-wise threshold vector, which will receive updates in the same training schedule. $t_{fl}(i)$ is the i^{th} element of the threshold vector for the l^{th} layer. If the absolute magnitude of the parameter exceeds the absolute value of its threshold, it is retained for inference, else it is dropped. This is accomplished by taking the Hadamard product $\theta_l \odot M_{f_l}$ where M_{f_l} is the mask generated by the unit step function. Steps 1-4 in Fig.1 illustrate this sparse forward pass. The difference between our method and DST's forward pass is that we consider the absolute value of the threshold in the difference. During the update of the thresholds, the signs of the respective parameters play a crucial role in deciding the sign of the threshold. In case of low magnitude negative weights, if the thresholds have negative values, the weights might still not be pruned since the step function always receives a positive input. To avoid this, we consider the absolute value of the thresholds. We also add the regularization term of DST $\mathcal{L}_s$ to the final loss function given by,

$$\mathcal{L}_s = \sum_{l=1}^{L} \sum_{i=1}^{n_l} \exp(-t_{f_l}(i))$$

3.3. Dynamic Gradient Sparsification

Even though DST's foward pass ends up giving us a sparse network, it still uses dense gradients to update the weights at each training step. We try to overcome this by performing gradient sparsification via a dynamic backward mask, which retains the most significant gradients. We hereby present the formulation of the dynamic backward mask.

3.3.1. Contribution Scores

In the backward pass, for $1 \leq l \leq L$, we have the gradient matrix for the l^{th} layer as,

$$G_l = \begin{bmatrix} g_{11} & \cdots & g_{1n_{l-1}} \\ \vdots & \ddots & \vdots \\ g_{n_l1} & \cdots & g_{n_ln_{l-1}} \end{bmatrix}$$

We define the 'Contribution Score' matrix (**CS**) of a particular gradient as its value normalized by the sum of the absolute values of all the gradients of the corresponding output neuron. Thus, the **CS** matrix is obtained as,

$$CS_l = \begin{bmatrix} \frac{g_{11}}{\sum_{i=1}^{n_{l-1}} |g_{1i}|} & \cdots & \frac{g_{1n_{l-1}}}{\sum_{i=1}^{n_{l-1}} |g_{1i}|} \\ \vdots & \ddots & \vdots \\ \frac{g_{n_l1}}{\sum_{i=1}^{n_{l-1}} |g_{n_li}|} & \cdots & \frac{g_{n_ln_{l-1}}}{\sum_{i=1}^{n_{l-1}} |g_{n_li}|} \end{bmatrix}$$

We use this contribution score to account for neuron-wise significance of the gradients. For a particular layer, the gradients obtained through backpropagation may be on different scales.

Also, as the training proceeds, the scale of a particular gradient may change drastically. These scaling issues can create problems for the training of dynamic backward mask. When these gradients are normalized, the resulting contribution scores are on the same scale. Hence, the relative importance of the gradients can be assessed using these contribution scores.

3.3.2. Backward Mask

We now extend the concept of trainable thresholds used in the forward sparsification to mask the gradients by comparing the magnitudes of their contribution scores with backward thresholds ($t_b \in \mathbb{R}^{n_l}$). We thereby generate and apply the backward mask as follows,

$$\begin{aligned} d_{bl}(i,j) &= |CS_l(i,j)| - |t_{bl}(i)|, \quad 1 \leq l \leq L,\ 1 \leq i \leq n_l \\ M_{bl}(i,j) &= \mathcal{U}\left(d_{bl}(i,j)\right), \quad 1 \leq j \leq n_{l-1} \end{aligned}$$

M_{bl} is the binary mask which is applied on the gradient matrix during backpropogation. This can be seen from steps 5-8 in Fig. 1. In case of the convolutional layers, the gradient tensor is flattened into a 2-D matrix and then the same steps are followed. The only difference is that instead of neuron-wise thresholds, we have channel-wise thresholds i.e, $t_b \in \mathbb{R}^{c_o}$ where c_o is the output channels. In backpropagation, we consider the derivative approximation of the unit step function as suggested by DST (Liu et al. (2020)), which is given as follows.

$$\frac{d}{dx}\mathcal{U}(x) \approx \mathcal{D}(x) = \begin{cases} 2 - 4|x|, & -0.4 \leq x \leq 0.4 \\ 0.4, & 0.4 < |x| \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

These backward thresholds receive updates through the Variance Penalty term included in the loss expression. Details about this are presented in Section 3.4.

3.4. Variance Penalty

We introduce a penalty term in the loss function which calculates standard deviation of masked gradient vector of the entire model (step 9 in Fig.1). Minimizing this loss function would inherently lead to reducing variance of the gradients. Since the gradients are masked based on the backward threshold values, the updates received by the backward thresholds result in penalizing gradients with high variance. We denote $\widetilde{G}$ as the masked gradient vector of the entire model. This vector is obtained by flattening the masked gradient matrices of the individual layers into vectors and then concatenating these vectors into a single vector. The variance loss is given by,

$$\mathcal{L}_v = \sqrt{Var(\widetilde{G})}$$

where $Var(\cdot)$ gives the variance of the input vector.

3.5. Dynamic Forward and Backward Sparse Training

Given our training dataset $\mathcal{D} = \{(x_i, y_i) \quad \forall \quad i \in [1, N]\}$ where N is the total number of training examples. We put together all the components discussed above and present a unified optimization process which dynamically sparsifies both forward and backward passes.

The forward mask is generated by magnitude pruning using dynamic forward thresholds. The loss function is given by,

$$\mathcal{J}(\theta, t_f, t_b) = \frac{1}{N}\left(\sum_{i=1}^{N} \mathcal{L}\left((x_i, y_i); \theta\right)\right) + \alpha\mathcal{L}_s + \beta\mathcal{L}_v$$

$\mathcal{L}_v$ is the variance penalty term explained in section 3.4 and $\mathcal{L}_s$ is the regularization term explained in section 3.2. The factor β controls the rate of increase of backward sparsity. As the backward thresholds try to reduce the variance of the masked gradients, it leads to better convergence. The coefficient α can be thought of as a scaling factor used to control the level of sparsity generated in the forward pass. If the value of α is increased, the network tries to reduce the loss function by decreasing the exponential term. Thus, the thresholds will increase resulting in higher sparsity levels. Hence, the optimization procedure would be aimed at finding θ^*, t_f^* and t_b^* such that,

$$\theta^*, t_f^*, t_b^* = \underset{\theta, t_f, t_b}{\arg\min} \mathcal{J}(\theta, t_f, t_b)$$

4. Experiments

We assess DFBST on a variety of deep neural network architectures and show that our method has consistent performance on different datasets. We also compare DFBST with similar methods and show that our approach demonstrates state of the art results.

4.1. Datasets

4.1.1. MNIST

MNIST (LeCun et al. (1998)) is a dataset of handwitten digits with 10 classes. It has a training set of 60,000 examples, and a test set of 10,000 examples. The images are 28×28 pixels with a single channel.

4.1.2. CIFAR-10 and CIFAR-100

CIFAR-10 (Krizhevsky et al. (2009)) consists of a training set of 50,000 examples and a test set of 10,000 examples with 10 target classes. Each example is a three channel image of 32 $\times$ 32 pixels. CIFAR-100 (Krizhevsky et al. (2009)) is similar to CIFAR-10 except that it consists of 100 target classes.

4.2. Models

We follow the same experimental setting as DST (Liu et al. (2020)). DFBST is evaluated on Lenet-300-100, Lenet-5 Caffe (LeCun et al. (1990)), VGG-16 (Simonyan and Zisserman (2015)) and Resnet-18 (He et al. (2015)) architectures. LeNet-300-100 is a two hidden layer perceptron while LeNet-5 Caffe is a CNN with two convolutional layers and two fully connected layers. VGG-16 is a standard deep CNN having 16 layers including fourteen convolutions and two fully connected layers. ResNet-18 is a CNN which uses residual connections. These connections allow very deep models to be trained efficiently without facing the issue of vanishing gradients. ResNet-18 has seventeen convolution layers with eight residual connections and one fully connected layer.

4.3. Baselines

We consider state-of-the-art sparse training methods like DST, Sparse Momentum, Sparse Evolutionary Training(SET) and Rigging the Lottery(RigL) as our baselines. DST utilizes dynamic masks obtained via trainable thresholds in the forward pass to induce sparsity. Sparse Momentum uses exponentially smoothed gradients (momentum) as a criterion to assess which layers are most efficient at reducing error. SET prunes fraction of weights closest to zero and randomly adds new weights at the end of every training epoch. RigL removes a fraction of connections at regular intervals based on their magnitude and activates new ones using instantaneous gradient information.

4.4. Results

We define the forward model remaining ratio as the ratio of total number of non-zero elements in all the forward masks to the total number of model parameters. Similarly, the backward model remaining ratio would be the ratio of number of non-zero elements in all the backward masks to the total number of model parameters. Few baselines, that we compare our method with, train the network for a greater number of epochs than we do. Since these methods maintain constant sparsity throughout training, we linearly scale down the FLOPs corresponding to the number of epochs for which we train the network. We use PyTorch for implementing and obtaining the results on an Nvidia Tesla P100 GPU. Each experiment is run 5 times and the corresponding confidence intervals are reported in Tables 1, 2, 3 and 4.

4.4.1. MNIST

We perform experiments on LeNet-300-100 and LeNet-5-Caffe architectures on the MNIST dataset. We use the SGD optimizer with a momentum of 0.9 and a batch size of 64. The model is trained for 20 Epochs with a learning rate of 0.01. We choose α equal to 0.0005 as suggested by DST for sparse regularization in both models. The scaling factor for the variance penalty term is set to 0.01. Table 1 and Table 2 present the pruning results of our method for Lenet-300-100 and Lenet-5-Caffe, respectively.

DFBST prunes $\sim 98\%$ parameters with little loss of performance on Lenet-300-100 and Lenet-5-Caffe. Fig.2 shows the epoch-wise comparison of DFBST with DST as well as dense network for LeNet-300-100. Our approach achieves better accuracy than all the other models. The forward model remaining ratio of our model is lower than DST which implies our model has higher sparsity than DST.

4.4.2. CIFAR-10

We evaluate the performance of DFBST on CIFAR-10 with the VGG architecture. Table 3 shows a comparison between different sparse learning algorithms and DFBST. We use SGD with momentum 0.9 and batch size of 64 with 160 training epochs. The learning rate is 0.1 and decayed by 0.1 at 80^{th} and 120^{th} epoch. α for sparse regularization term is set to

Table 1: Results of Lenet-300-100 on MNIST for various methods

Models	Accuracy (%)	Adjusted FLOPS Reduction	Parameter Count ($\sim 10^3$)
Dense	98.08 ± 0.07	-	31
Sparse Momentum	**99.70 ± 0.23**	2.1×	3.1
SET	98.19 ± 0.35	1.48×	1.55
RigL	98.12 ± 0.16	3.9×	1.55
DST	97.6 ± 0.74	1.59×	0.93
DFBST	98.06 ± 0.09	**4.83**×	**0.774**

Table 2: Results of Lenet-5 Caffe on MNIST for various methods

Models	Accuracy (%)	Adjusted FLOPS Reduction	Parameter Count ($\sim 10^3$)
Dense	99.18 ± 0.03	-	60
Sparse Momentum	**99.30 ± 0.19**	3.1×	5.4
SET	98.96 ± 0.06	2.42×	3
RigL	99.12 ± 0.04	3.47×	3
DST	99.08 ± 0.66	5.47×	2.28
DFBST	99.06 ± 0.09	**5.56**×	**1.38**

5×10^{-6} as suggested by DST and the scaling factor β for the variance penalty term is set to 0.1.

Table 3: Results of VGG-16 on Cifar-10 for various methods

Models	Accuracy (%)	Adjusted FLOPS Reduction	Parameter Count ($\sim 10^6$)
Dense	93.48 ± 0.12	-	138
Sparse Momentum	93.10 ± 0.21	3.51×	13.8
SET	93.32 ± 0.03	2.15×	**6.9**
RigL	93.43 ± 0.11	6.4×	**6.9**
DST	93.52 ± 0.14	9.09×	11.04
DFBST	**93.75 ± 0.05**	**10.99**×	7.45

Fig.3a shows comparison of DFBST with DST. The accuracy of our model is marginally higher than DST. But we obtain a significantly lower forward keep ratio. Fig.3b demonstrates backward model sparsity induced in VGG-16. As evident from the plot, the initial

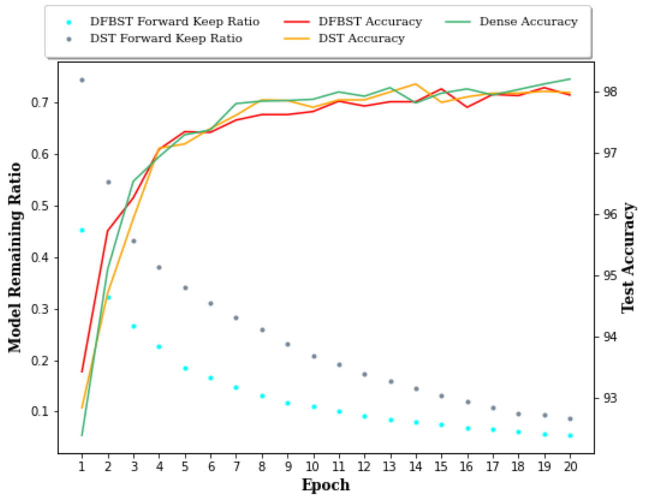

Figure 2: Model remaining ratio and Test accuracy v/s Epochs for Lenet-300-100 in comparison with dense and DST models. For all models $\alpha = 5 \times 10^{-4}$.

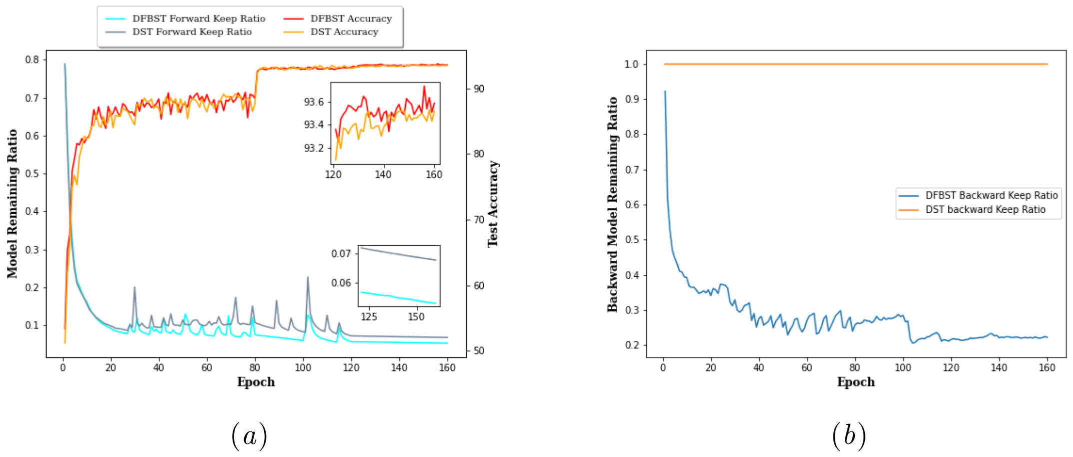

Figure 3: (a) Model remaining ratio and Test accuracy v/s Epochs for VGG-16 in comparison with dense and DST models. For all models $\alpha = 5 \times 10^{-6}$. (b) Backward Model Keep Ratio v/s Epochs for $\beta = 0.1$ at $\alpha = 5 \times 10^{-6}$.

backward keep ratio is high and decreases gradually with training. Towards the end of training, very few gradients are retained.

4.4.3. CIFAR-100

Experimentation on CIFAR-100 is done using ResNet-18. With our method we achieve 88% forward sparsity and comparable performance against the dense model. As shown in Table 4, DFBST surpasses all the sparse training methods in terms of model sparsity. The optimizer used was SGD with momentum 0.9 and batch size of 64 with 100 training epochs. The learning rate is 0.1 and decayed by 0.1 at the 80th epoch. α for sparse regularization

Table 4: Results of ResNet-18 on CIFAR-100 for various methods

Models	Accuracy (%)	Adjusted FLOPS Reduction	Parameter Count ($\sim 10^6$)
Dense	63.8 ± 0.088	-	11
Sparse Momentum	62.14 ± 0.035	$4.59\times$	2.75
SET	62.28 ± 0.028	$2.0\times$	2.2
RigL	62.09 ± 0.059	$3.2\times$	2.2
DST	62.62 ± 0.04	$2.34\times$	6.82
DFBST	**63.66 ± 0.07**	**$4.97\times$**	**1.32**

term is 5×10^{-6} as suggested by DST (Liu et al. (2020)) and the scaling factor β for the variance penalty term is 0.01.

5. Discussion

5.1. Improved fine grained step pruning

The forward pass of our method uses a trainable dynamic mask obtained using trainable thresholds. These trainable thresholds ensure that the layer remaining ratios change smoothly and continuously at each training step. This was demonstrated by DST as well. However, considering the absolute values of forward thresholds and addition of a backward trainable mask decreases the layer keep ratios and in essence, the overall model keep ratio resulting in a more sparse architecture. This forward trainable mask is controlled by α for which we have provided an ablation study in Fig.4.

5.2. Ablation Study

5.2.1. Forward Sparsity

α is used as a scaling coefficient for the regularization term $\mathcal{L}_s$. It controls the level of forward sparsity induced in the parameters. Fig.4 shows the variation in test accuracy and forward model remaining ratio with training epochs. We observe that for higher values of α, the forward sparsity increases almost linearly while the performance also drops in an approximate linear fashion. This is desirable as one can expect good correlation between sparsity and accuracy.

5.2.2. Backward Sparsity

β is used as a scaling coefficient for the variance penalty term. It controls the level of backward sparsity induced in the gradients. Fig.5 shows the variation in test accuracy and backward model remaining ratio with training epochs. We observe that for higher values of β, the backward sparsity quickly reaches a large value. This is not desirable, as shutting off a large number of gradients early in the training does not allow the model to learn. For very small values of β, no backward sparsity is achieved. This implies that most of the

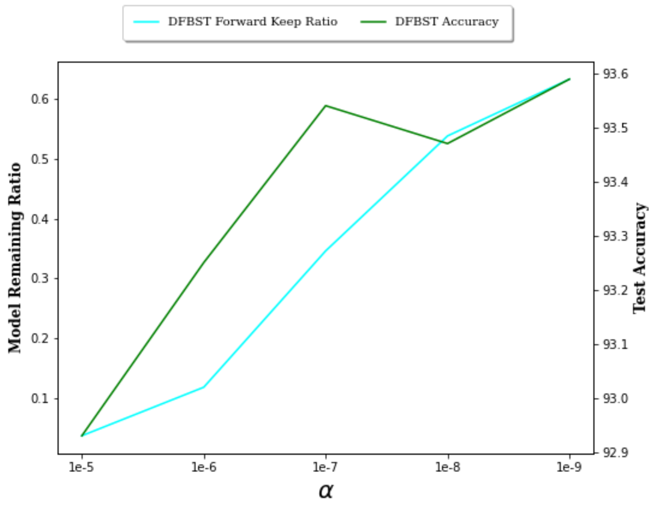

Figure 4: Forward Sparsity and Test accuracy for VGG-16 model trained on CIFAR-10 dataset at different values of α and at constant $\beta = 0.01$.

high variance, low magnitude gradients are not sparsified, which again does not satisfy our purpose. We therefore use a moderate value of β which allows the model to learn sufficiently

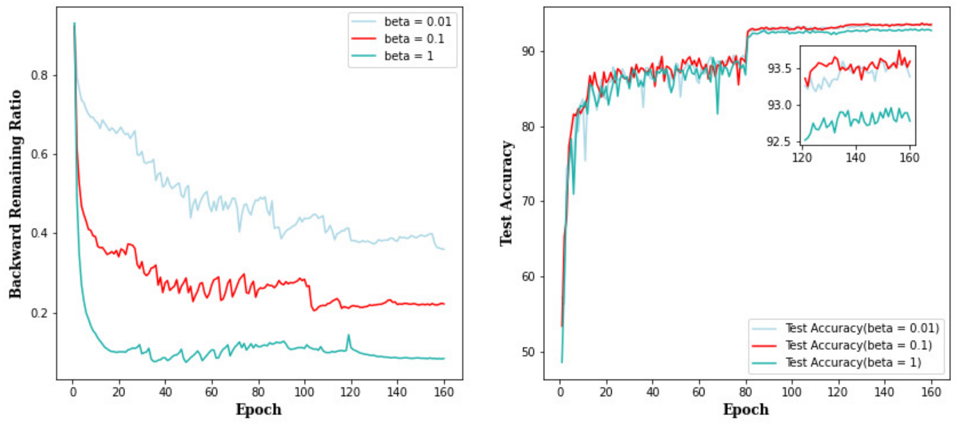

Figure 5: Backward Sparsity and Test accuracy for VGG-16 model trained on CIFAR-10 dataset at different values of β and at constant $\alpha = 5 \times 10^{-6}$.

well during the initial phase of training and then gradually sparsifies more gradients towards the end of training. To understand the β parameter, one can draw parallel with the learning rate of a neural network. A large learning rate may lead to oscillation of weights about the global minima, while a very small value might never result in convergence. Thus, we choose an optimal value of learning rate which tackles both the issues.

5.3. Gradient Variance Regularization

Gradient sparsification increases variance while it is well known that an increase in gradient variance hurts convergence rates of SGD (Wangni et al. (2018b), Hoefler et al. (2021), Johnson and Zhang (2013)). Thus, we introduce a variance regularization term into the loss which is used to update the backward mask thresholds. This ensures removal of only those gradient values that contribute to an increase in the overall variance of model gradient. This is why increasing the β parameter leads to pruning of most gradients except those that are exceedingly similar which subsequently leads to higher sparsification but reduced performance as shown in the ablation study in Fig. 5.

5.4. Layer-wise Pruning Ratios

We demonstrate layer-wise pruning using DFBST with a VGG-16 model trained on the CIFAR-10 dataset. As can be seen from the graph, the deeper layers are subject to stronger pruning than the initial layers except for the final output layer.

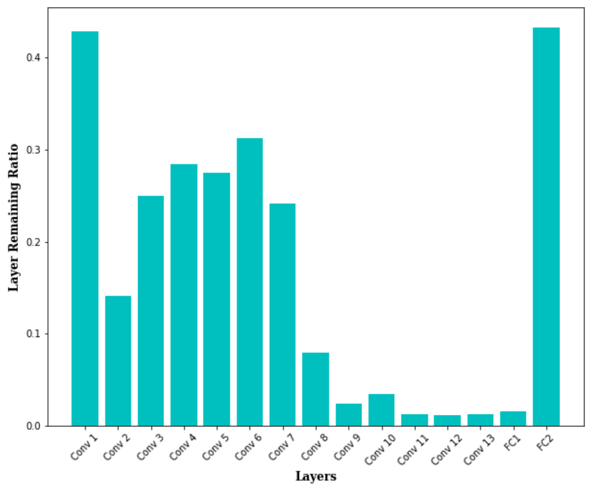

Figure 6: The layer-wise pruned ratios for a VGG-16 model trained on CIFAR-10 dataset when trained with DFBST

Conclusion

In this paper, we proposed DFBST a method which ensures Completely Sparse Training. DFBST starts with a completely dense architecture and sparsifies the parameters using magnitude pruning with trainable thresholds. During back propagation it sparsifies the gradients using backward trainable thresholds. This is achieved by penalizing high variance gradients. The backward thresholds prune the low magnitude gradients. Consequently, we obtain a sparse network with state of the art accuracy and high sparsity during inference. DFBST has the capability of being conceptualized with various types of neural network layers and then into architectures. Thus, as our future work, we would like to extend our method to language and sequential models.

References

Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 440–445, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1045. URL https://aclanthology.org/D17-1045.

Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497, 2019.

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

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

Yannis Flet-Berliac, Reda Ouhamma, Odalric-Ambrym Maillard, and Philippe Preux. Learning value functions in deep policy gradients using residual variance. *arXiv preprint arXiv:2010.04440*, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Robert M Gower, Mark Schmidt, Francis Bach, and Peter Richtárik. Variance-reduced methods for machine learning. *Proceedings of the IEEE*, 108(11):1968–1983, 2020.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1510.00149.

Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.

Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. In H. Larochelle, M. Ranzato, R. Hadsell, M. F.

Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20744–20754. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/ee76626ee11ada502d5dbf1fb5aae4d2-Paper.pdf.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems*, 26, 2013.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

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

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: single-shot network pruning based on connection sensitivity. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=B1VZqjAcYX.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=rJqFGTslg.

Junjie Liu, Zhe Xu, Runbin Shi, Ray C. C. Cheung, and Hayden Kwok-Hay So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SJlbGJrtDB.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=H1Y8hhg0b.

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):1–12, 2018.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJGCiw5gl.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pages 4646–4655. PMLR, 2019.

Michael C Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems*, pages 107–115, 1989.

Sharan Narang, Greg Diamos, Shubho Sengupta, and Erich Elsen. Exploring sparsity in recurrent neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=BylSPv9gx.

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry P. Vetrov. Tensorizing neural networks. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 442–450, 2015. URL https://proceedings.neurips.cc/paper/2015/hash/6855456e2fe46a9d49d3d3af4f57443d-Abstract.html.

Simone Scardapane, Danilo Comminiello, Amir Hussain, and Aurelio Uncini. Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556.

Xu Sun, Xuancheng Ren, Shuming Ma, and Houfeng Wang. meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting. In *International Conference on Machine Learning*, pages 3299–3308. PMLR, 2017.

Georg Thimm and Emile Fiesler. Evaluating pruning methods. In *Proceedings of the International Symposium on Artificial neural networks*, pages 20–25. Citeseer, 1995.

Yusuke Tsuzuku, Hiroto Imachi, and Takuya Akiba. Variance-based gradient compression for efficient distributed deep learning. *arXiv preprint arXiv:1802.06058*, 2018.

Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 1306–1316, 2018a. URL https://proceedings.neurips.cc/paper/2018/hash/3328bdf9a4b9504b9398284244fe97c2-Abstract.html.

Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization. *Advances in Neural Information Processing Systems*, 31, 2018b.

Xiao Zhou, Weizhong Zhang, Zonghao Chen, Shizhe Diao, and Tong Zhang. Efficient neural network training via forward and backward propagation sparsification. *Advances in Neural Information Processing Systems*, 34, 2021.