Accelerating CNN Training by Pruning Activation Gradients

Xucheng Ye[1], Pengcheng Dai[2], Junyu Luo[1], Xin Guo[1], Yingjie Qi[1], Jianlei Yang[1(✉)], and Yiran Chen[3]

[1] SCSE, BDBC, Beihang University, Beijing, China
jianlei@buaa.edu.cn
[2] SME, BDBC, Beihang University, Beijing, China
[3] ECE, Duke University, Durham, NC, USA

Abstract. Sparsification is an efficient approach to accelerate CNN inference, but it is challenging to take advantage of sparsity in training procedure because the involved gradients are dynamically changed. Actually, an important observation shows that most of the activation gradients in back-propagation are very close to zero and only have a tiny impact on weight-updating. Hence, we consider pruning these very small gradients randomly to accelerate CNN training according to the statistical distribution of activation gradients. Meanwhile, we theoretically analyze the impact of pruning algorithm on the convergence. The proposed approach is evaluated on AlexNet and ResNet-{18, 34, 50, 101, 152} with CIFAR-{10, 100} and ImageNet datasets. Experimental results show that our training approach could substantially achieve up to 5.92× speedups at back-propagation stage with negligible accuracy loss.

Keywords: CNN training · Acceleration · Gradients pruning

1 Introduction

Convolutional Neural Networks (CNNs) have been widely applied to many tasks and various devices in recent years. However, the network structures are becoming more and more complex, making the training of CNN on large scale datasets very time consuming, especially with limited hardware resources. Some previous researches have shown that CNN training could be finished within minutes on high performance computation platforms [1–3], but thousands of GPUs have to be utilized, which is not feasible for many scenarios. Even though there are many existing works on network compressing, most of them are focused on inference [4]. Our work aims to reduce the training workloads efficiently, enabling large scale training on budgeted computation platforms.

This work is supported in part by the National Natural Science Foundation of China (61602022), State Key Laboratory of Software Development Environment (SKLSDE-2018ZX-07), CCF-Tencent IAGR20180101 and the 111 Talent Program B16001.

A. Vedaldi et al. (Eds.): ECCV 2020, LNCS 12370, pp. 322–338, 2020.
https://doi.org/10.1007/978-3-030-58595-2_20

The essential optimization step of CNN training is to perform Stochastic Gradient Descent (SGD) algorithm in back-propagation procedure. There are several data types involved in training dataflow: weights, weight gradients, activations, activation gradients. Back-propagation starts from computing the weight gradients with the activations and then performs weights update [5]. Among these steps, *activation gradients back-propagation* and *weight gradients computation* require intensive convolution operations thus dominate the total training cost. It is well known that computation cost can be reduced by skipping over zero-values. Since these two convolution steps require the activation gradients as input, improving the sparsity of activation gradients should significantly reduce the computation cost and memory footprint during back-propagation procedure.

Without loss of generality, we assume that the numerical values of activation gradient satisfy normal distributions, and a threshold τ can be calculated based on this hypothesis. And then *stochastic pruning* is applied on the activation gradients with the threshold τ while the gradients are set to zero or $\pm\tau$ randomly. Since the `ReLU` layers usually make the gradients distributed irregularly, we divide common networks into two categories, one is networks using `Conv-ReLU` as basic blocks such as AlexNet [6] and VGGNet [7], another is those using `Conv-BN-ReLU` structure such as ResNet [8]. Experiments show that our pruning method works for both `Conv-ReLU` structure and `Conv-BN-ReLU` structure in modern networks. A mathematical analysis is provided to demonstrate that stochastic pruning can maintain the convergence properties of CNN training. Additionally, our proposed training scheme is evaluated both on Intel CPU and ARM CPU platforms, which could achieve $1.71\times \sim 3.99\times$ and $1.79\times \sim 5.92\times$ speedups, respectively, when compared with no pruning utilized at back-propagation stage.

2 Related Works

Weight pruning is a well-known acceleration technique for CNN inference phase which has been widely researched and achieved outstanding advances. Pruning of weights can be divided into five categories [4]: element-level [9], vector-level [10], kernel-level [11], group-level [12] and filter-level pruning [13–17]. Weight pruning focuses on raising parameters sparsity of convolutional layers.

Weight gradients pruning is proposed for training acceleration by reducing communication cost of weight gradients exchanging in distributed learning system. Aji [18] prunes 99% weight gradients with the smallest absolute value by a heuristic algorithm. According to filters' correlationship, Prakash [19] prunes 30% filters temporarily to improve training efficiency.

Activation gradients pruning is another approach to reduce training cost but is rarely researched because activation gradients are generated dynamically during back-propagation. Most previous works adopt *top-k* as the base algorithm for sparsification. For MLP training, Sun [20] adopts *min-heap* algorithm to find and retain the k elements with the largest absolute value in the activation gradients for each layer, and discards the remaining elements to improve sparsity. Wei

[21] further applies this scheme to CNN's training, but only evaluated on LeNet. In the case of larger networks and more complex datasets, directly dropping redundant gradients will cause significant loss of learnt information. To alleviate this problem, Zhang [22] stores the un-propagated gradients at the last learning step in memory and adds them to the gradients before *top-k* sparsification in the current iteration. Our work can be categorized into this scope. We propose two novel algorithms to determine the pruning threshold and preserve the valuable information, respectively.

Quantization is another common way to reduce the computational complexity and memory consumption of training. Gupta's work [23] maintains the accuracy by training the model in the precision of 16-bit fixed-point number with stochastic rounding. DoReFaNet [24] derived from AlexNet [6] utilizes 1-bit, 2-bit and 6-bit fixed-point number to represent weights, activations and gradients respectively, but brings visible accuracy drop. Park [25] proposed a value-aware quantization method by using low-precision on small values, which can significantly reduce memory consumption when training ResNet-152 [8] and Inception-V3 [26] with 98% activations quantified to 3-bit. Micikevicius [5] keeps an FP32 copy for weight update and adopts FP16 for computation, which is efficient for training acceleration. Our approach can be regarded as gradients sparsification, and can be also integrated with gradients quantization methods.

3 Methodologies

3.1 General Dataflow

The convolution (`Conv`) layer involved in each training iteration usually includes four stages: *Forward*, *Activation Gradients Back-propagation*, *Weight Gradients Computation* and *Weight Update*. To present the calculation of these stages, some definitions and notations are introduced and adopted throughout this paper:

- $\mathbf{I}$ denotes the input of each layer at *Forward* stage.
- $\mathbf{O}$ denotes the output of each layer at *Forward* stage.
- $\mathbf{W}$ denotes the weights of `Conv` layer.
- d$\mathbf{I}$ denotes the gradients of I.
- d$\mathbf{O}$ denotes the gradients of O.
- d$\mathbf{W}$ denotes the gradients of W.
- $*$ denotes the 2-D convolution.
- η denotes the learning rate.
- $\mathbf{W}^+$ denotes the sequentially reversed of $\mathbf{W}$.

And the four training stages of `Conv` layer can be summarized as:

- *Forward* $\mathbf{O} = \mathbf{I} * \mathbf{W}$ (notice that we leave out bias here)
- *Activation Gradients Back-Propagation*(*AGBP*): $\mathrm{d}\mathbf{I} = \mathbf{W}^+ * \mathrm{d}\mathbf{O}$
- *Weight Gradients Computation*(*WGC*): $\mathrm{d}\mathbf{W} = \mathrm{d}\mathbf{O} * \mathbf{I}$
- *Weight Update*: $\mathbf{W} \leftarrow \mathbf{W} - \eta \cdot \mathrm{d}\mathbf{W}$

We found that activation gradients involved in back-propagation stage are almost full of *very small values* that are extremely close to zero. It is reasonable to assume that pruning those extremely small values has little effect on weight update stage. Meanwhile, existing works show that pruning redundant elements in convolution calculations can effectively reduce arithmetic complexity. Therefore, we make a hypothesis that the involved `Conv` layers computations in training can be accelerated substantially by pruning activation gradients.

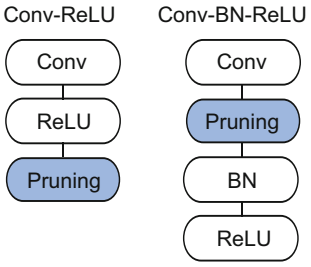


Fig. 1. Pruning stages involved for two typical structures: `Conv-ReLU` and `Conv-BN-ReLU`.

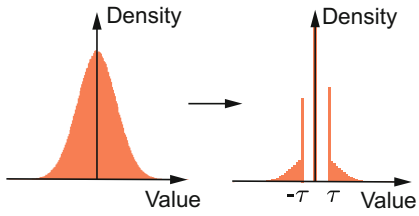


Fig. 2. Effect of *stochastic pruning*, where τ is the pruning threshold.

3.2 Sparsification Algorithms

Distribution Based Threshold Determination (DBTD). The most important concern of pruning is to determine which elements should be selected for discarding. Previous works [20] use min-heap algorithm to select which elements going to be pruned. However, they will introduce inevitable overhead significantly when implemented on heterogeneous platforms such as FPGA or ASIC. Hence, we propose a new threshold determination method with less time complexity and more hardware compatibility.

Firstly, we analyze the distribution of activation gradients for two typical structures of modern CNN models, as shown in Fig. 1. For `Conv-ReLU` structure, where a `Conv` layer is followed by a `ReLU` layer, output activation gradients $\mathrm{d}\mathbf{O}$ are sparse, but subject to an irregular distribution. On the other hand, the input activation gradients $\mathrm{d}\mathbf{I}$, which will be propagated to the previous layer, is almost full of non-zero values. Statistics show that the probability distribution of $\mathrm{d}\mathbf{I}$ is symmetrical around zero and its probability density function decreases with the increment of absolute value $|\mathrm{d}\mathbf{I}(\cdot)|$. For `Conv-BN-ReLU` structure, a `BN` layer is located between `Conv` and `ReLU` layer, and $\mathrm{d}\mathbf{O}$ subjects to the similar distribution of $\mathrm{d}\mathbf{I}$. With the same hypothesis [27], these gradients are assumed to subject to a normal distribution with mean value 0 and variance σ^2.

For `Conv-ReLU` structure, d$\mathbf{O}$ can inherit the sparsity from d$\mathbf{I}$ of last `Conv` layer because `ReLU` layer will not reduce the sparsity. Thus d$\mathbf{I}$ can be treated as pruning target g in `Conv-ReLU` structure. For `Conv-BN-ReLU` structure, d$\mathbf{O}$ is considered as pruning target g. In this way, the distribution of g in both situations could be unified to normal distribution. Supposing that the scale of g is n, we calculate the mean value of the absolute values from gradient data g, and the expectation of it is:

$$E\left(\frac{1}{n}\sum_{i=1}^{n}|g_i|\right) = \frac{n}{\sqrt{2\pi\sigma^2}}\int |x|\exp\left\{-\frac{x^2}{2\sigma^2}\right\}dx = \sqrt{\frac{2}{\pi}}n\sigma. \tag{1}$$

Let

$$\hat{\sigma} = \frac{1}{n}\sqrt{\frac{2}{\pi}}\sum_{i=1}^{n}|g_i|, \tag{2}$$

then

$$E(\hat{\sigma}) = E\left(\frac{1}{n}\sqrt{\frac{2}{\pi}}\sum_{i=1}^{n}|g_i|\right) = \sigma. \tag{3}$$

Clearly, $\hat{\sigma}$ is an unbiased estimator of parameter σ.

Here we adopt the mean value of the absolute values because the computational overhead is acceptable. Base on the assumption, we can compute the threshold τ with the cumulative distribution function of the standard normal distribution Φ, target pruning rate p and $\hat{\sigma}$ by:

$$\tau = \Phi^{-1}\left(\frac{1-p}{2}\right)\hat{\sigma}. \tag{4}$$

Stochastic Pruning. Pruning a few gradients with small values has little impact on weights update. However, once all of these small gradients are set to 0, the distribution of activation gradients will be affected significantly, which will influence the weights update and cause severe accuracy loss. Inspired by *Stochastic Rounding* in [23], we adopt stochastic pruning to solve this problem.

Stochastic pruning treats gradients as an one-dimensional vector g with length n, and all the components whose absolute value is smaller than the threshold τ will be pruned. The algorithm details are demonstrated in Algorithm 1. The effect of stochastic pruning on gradient distribution is illustrated in Fig. 2.

Algorithm 1: Stochastic Pruning

Input: original activation gradients g, threshold τ
Output: sparse activation gradients $\hat{g}$
for $i = 1; i \leq n; i = i + 1$ **do**
  if $|g_i| < \tau$ **then**
    Generate a random number $r \in [0, 1]$;
    if $|g_i| > r\tau$ **then**
      $\hat{g}_i = (g_i > 0)\ ?\ \tau : (-\tau)$;
    else
      $\hat{g}_i = 0$;
    end
  end
end

Stochastic pruning could maintain the mathematical expectation of the gradients distribution while completing the pruning. Mathematical analysis in Sect. 4 will show that such a gradients sparsification method for CNN training does not affect its convergence.

In summary, compared with existing works, our scheme has two advantages:

(1) *Lower runtime cost*: the arithmetic complexity of *DBTD* is $\mathcal{O}(n)$, less than `top-k` which is at least $\mathcal{O}(n \log k)$, where k stands for the number of reversed elements. Meanwhile, *DBTD* is more hardware friendly and easier to be implemented on heterogeneous platform because it does not require frequent comparison operations.
(2) *Lower memory footprint*: our *Stochastic Pruning* approach could preserve the convergence rate and does not require any extra memory consumption. In contrast, [22] needs to store the un-propagated gradients of the last training steps, which is more memory consuming.

4 Convergence Analysis

The convergence rate of our proposed stochastic pruning is analyzed in this section. *Please note that it is not a rigorous mathematical proof, but just provide some intuition on why the gradients pruning method works.* We expect that our training method with stochastic pruning has similar convergence rate with origin training process under the GOGA (General Online Gradient Algorithm) framework [28].

In [28], L. Bottou considers a learning problem as follows: suppose that there is an unknown distribution $P(z)$ and can only get a batch of samples z_t each iteration, where t denotes iteration times. The goal of training is to find the optimal parameters w which minimize the loss function $\mathrm{Q}(z, w)$. For convenience, we define the cost function as:

$$\mathrm{C}(w) \triangleq \mathbf{E}_{\mathbf{z}} \mathrm{Q}(z, w) \triangleq \int \mathrm{Q}(z, w)\, \mathrm{d}P(z). \tag{5}$$

And the involved update rule for the online learning system is formulated as:

$$w_{t+1} = w_t - \gamma_t \mathrm{H}\left(z_t, w_t\right), \tag{6}$$

where γ_t is the learning rate, $\mathrm{H}(\cdot)$ is the update function. It will finally converge as long as the following assumptions are satisfied.

Assumption 1. *The cost function* $\mathrm{C}(w_t)$ *has a single global minimum* w^* *and satisfies the condition that*

$$\forall \varepsilon, \quad \inf_{(w-w^*)^2>\varepsilon} (w - w^*) \nabla_w \mathrm{C}(w) > 0. \tag{7}$$

Assumption 2. *Learning rate* γ_t *fulfills that*

$$\sum_{t=1}^{\infty} \gamma_t = \infty, \qquad \textit{and} \qquad \sum_{t=1}^{\infty} \gamma_t^2 < \infty. \tag{8}$$

Assumption 3. *For each iteration, the update function* $\mathrm{H}(z_t, w_t)$ *meets that*

$$\mathbf{E}\left[\mathrm{H}(z, w)\right] = \nabla_w \mathrm{C}(w), \tag{9}$$

and

$$\mathbf{E}\left[\mathrm{H}(z, w)^2\right] \leq \alpha + \beta (w - w^*)^2, \tag{10}$$

where α *and* β *are finite constants, the update function* $\mathrm{H}(z, w)$ *consists of the calculated gradients by back-propagation algorithm.*

The only difference between our proposed algorithm and [28] is the update function $\mathrm{H}(z, w)$. For original algorithm, the update function $\mathrm{H}(z_t, w_t)$ satisfies:

$$\mathbf{E}\left[\mathrm{H}(z, w)\right] = \nabla_w \mathrm{C}(w). \tag{11}$$

In proposed algorithm, a gradients pruning method is applied on the update function, denoted as $\hat{\mathrm{H}}(z, w)$. In this case, if we assume original back-propagation algorithm meets all the assumptions, the proposed algorithm also satisfies *Assumption* 1 and 2. If *Assumption* 3 can be also held by the proposed algorithm, we can say that both algorithms have similar convergence. For convenience, their corresponding gradients are denoted as $G \triangleq \mathrm{H}(z, w)$ and $\hat{G} \triangleq \hat{\mathrm{H}}(z, w)$.

In the following we will first prove that though $\hat{G} \neq G$, the expectations of them are the same. What's more, we expect the extra noise introduced by gradient pruning is not significant enough to violate *Assumption* 3. More precisely, the following equations should be held:

$$\mathbf{E}\left[\hat{G}\right] = \mathbf{E}\left[G\right], \tag{12}$$

$$\mathbf{E}\left[\hat{G}^2\right] \leq \alpha + \beta \mathbf{E}\left[G^2\right]. \tag{13}$$

To discuss *Assumption* 3, we first give a lemma:

Lemma 1. *For a stochastic variable x, we get another stochastic variable y by applying Algorithm 1 to x with threshold τ, which means*

$$y = \text{Prune}(x) = \begin{cases} x & \textit{i.f.f. } |x| \geq \tau \\ 0 & \textit{with probability } p = \dfrac{\tau - x}{\tau} \quad \textit{i.f.f. } |x| < \tau \\ \tau & \textit{with probability } p = \dfrac{x}{\tau} \quad \textit{i.f.f. } |x| < \tau \end{cases} \tag{14}$$

Then y satisfies

$$\mathrm{E}[y] = \mathrm{E}\left[\text{Prune}(x)\right] = \mathrm{E}[x], \tag{15}$$

$$\mathrm{E}[y^2] = \mathrm{E}\left[\text{Prune}(x)^2\right] \leq \tau^2 + \mathrm{E}[x^2]. \tag{16}$$

Then we can discuss the expectation and variance of gradients $\hat{G}$.

4.1 Expectation of Gradients

Lemma 1 means that gradients pruning will not affect the expectation of activation gradients, which can be utilized to prove Eq. (12). Let G represent the gradients of the whole network parameters with N layers. Thus we can split it into layer-wise gradients:

$$G = (G_1, G_2, \cdots, G_l, \cdots, G_N) \tag{17}$$

where G_l represents the gradients of l-th layer weights. Let GO_l represents the activation gradients for l-th layer, we have:

$$GO_l = F_1(GO_{l+1}, \omega), \qquad \textit{and} \qquad G_l = F_2(GO_l) \tag{18}$$

where F_1 and F_2 represents the back-propagation operation for l-th layer.

The same thing can be done for $\hat{G}$ which means:

$$\hat{G} = (\hat{G}_1, \hat{G}_2, \cdots, \hat{G}_l, \cdots, \hat{G}_N), \tag{19}$$

$$\hat{GO}_l = \text{Prune}\left[F_1(\hat{GO}_{l+1}, \omega)\right], \tag{20}$$

$$\hat{G}_l = F_2\left(\hat{GO}_l\right). \tag{21}$$

To prove Eq. (12), we only need to prove that for each l,

$$\mathbf{E}\left[\hat{G}_l\right] = \mathbf{E}\left[G_l\right], \tag{22}$$

$$\mathbf{E}\left[\hat{GO}_l\right] = \mathbf{E}\left[GO_l\right]. \tag{23}$$

Note that Eq. (23) is already held for the last layer. Because the last layer is the start of back-propagation and the proposed algorithm is the same with original

algorithm before the last layer's gradients G are calculated, we only need to prove that:

$$\mathbf{E}\left[G_l\right] = F_1\left(\mathbf{E}\left[GO_l\right]\right) \tag{24}$$

$$\mathbf{E}\left[GO_l\right] = F_2\left(\mathbf{E}\left[GO_{l+1}\right]\right) \tag{25}$$

Because if Eq. (24) and Eq. (25) are held, we can prove Eq. (9) by using *Lemma* 1.

Proof. Assume Eq. (23) is satisfied for $(l+1)$-th layer. Then for l-th layer

$$\mathrm{E}[\hat{G}_l] = F_1\left(\mathbf{E}\left[\hat{GO}_l\right]\right) \tag{26}$$

$$= F_1\left(F_2\left(\mathbf{E}\left[\mathrm{Prune}\left(\hat{GO}_{l+1}\right)\right]\right)\right) \tag{27}$$

$$= F_1\left(F_2\left(\mathbf{E}\left[\hat{GO}_{l+1}\right]\right)\right) \tag{28}$$

$$= F_1\left(F_2\left(\mathbf{E}\left[GO_{l+1}\right]\right)\right) \tag{29}$$

$$= F_1\left(\mathbf{E}\left[GO_l\right]\right) \tag{30}$$

$$= \mathrm{E}[G_l] \tag{31}$$

The equality of Eq. (26) and Eq. (31) could be guaranteed by Eq. (24). Equation (27) and Eq. (30) is true because of Eq. (25). Equation (28) is right due to *Lemma* 1. Since the assumption Eq. (23) is true for the last layer, then for all l, Eq. (9) is right.

As for Eq. (24) and Eq. (25) is true because they are linear operation except `ReLU` in the case of CNN and the back-propagation of `ReLU` can exchange with expectation. Here we denote the back-propagation of `ReLU` as `ReLU'`. `ReLU'` will set the operand to zero or hold its value. For the former one,

$$\mathbf{E}\left[\texttt{ReLU'}(x)\right] = 0 = \texttt{ReLU'}\left(\mathbf{E}\left[x\right]\right).$$

For the latter one,

$$\mathbf{E}\left[\texttt{ReLU'}(x)\right] = \mathbf{E}\left[x\right] = \texttt{ReLU'}\left(\mathbf{E}\left[x\right]\right).$$

Thus we prove that the expectation of gradients in the proposed algorithm is the same with the original algorithm.

4.2 Variance of Gradients

It is difficult to prove that Eq. (10) can be also satisfied in the proposed gradient pruning algorithm. However, we can give some intuition that this may be right if original training method meets this condition. Equation (10) tell us that, to guarantee the convergence during stochastic gradient descend, variance of gradients in each step should not be too large. The proposed gradient pruning method will indeed bring extra noise to the gradients. But we believe the extra

noise is not significant enough to violate Eq. (10). The extra gradients noise is determined by two factors. First is the noise generated by pruning method. Second is the propagation of the pruning noise in the following back-propagation process.

From Eq. (16) we can tell that the variance of the pruned gradients will only increase by a constant number relating to threshold τ. This will certainly obey the condition in Eq. (10). What's more, the noise is then propagating through `Conv` and `ReLU` layers, whose operation is either linear or sublinear. Thus we can expect that the increase of variance will still be quadratic, which satisfies Eq. (10). In this way, we can say that the proposed pruning algorithm has almost the same convergence with the original algorithm under the GOGA framework.

5 Implementation

5.1 Accuracy Evaluation

PyTorch [29] framework is utilized to estimate the impact on accuracy for our gradient pruning method. The straight-through estimator (STE) is adopted in our implementation. We introduce an extra `Pruning` layer for different `Conv` block as shown in Fig. 1. As mentioned above, the input and output of this layer can be denoted as $\mathbf{I}$ and $\mathbf{O}$. The essence of this `Pruning` layer is a STE which can be defined as below:

$$\textbf{Forward: } \mathbf{O} = \mathbf{I}$$

$$\textbf{Backward: } \mathrm{d}\mathbf{I} = \mathit{Stochastic\_Pruning}\left(\mathrm{d}\mathbf{O}, \mathit{DBTD}(\mathrm{d}\mathbf{O}, p)\right)$$

5.2 Speedup Evaluation

To estimate the acceleration effect of our algorithm, we modify the backward `Conv` layers in Caffe [30] framework, which is widely used in deep learning deployment. As mentioned in Sect. 3.1, two main steps of training stage: *AGBP*, *WGC* are all based on convolution. Most modern deep learning frameworks including Caffe convert convolution into matrix multiplication by applying the combination of `im2col` and `col2im` functions, where `im2col` turns a 3-D feature map tensor into a 2-D matrix for exploiting data reuse, and `col2im` is the inverse function of `im2col`. Hence, our training acceleration with sparse activation gradients can be accomplished by replacing the original matrix multiplication with sparse matrix multiplication.

With our proposed algorithm, the activation gradients $\mathrm{d}\mathbf{O}$ can be fairly sparse. However, weight $\mathbf{W}$ and activation $\mathbf{O}$ are completely dense. We found that `dense` $\times$ `sparse` matrix multiplication is required for *AGBP* step. However, the existing BLAS library such as Intel MKL only supports `sparse` $\times$ `dense` multiplication. To solve this problem, we turn to compute the transpose of $\mathrm{d}\mathbf{I}$ according to the basic property of matrix multiplication $(AB)^T = B^T A^T$, where both A and B are matrices.

To reduce the computation cost, we modify the original `im2col` and `col2im` functions to `im2col_trans` and `col2im_trans` so that we can get transposed matrix directly after calling these functions. Since plenty of runtime can be saved by using `sparse` $\times$ `dense` multiplication, we can also achieve relatively high speedup in the overall back-propagation process, though transpose functions will cost extra runtime. The modified procedure can be summarized as:

$$AGBP: \mathrm{d}\mathbf{I} = \texttt{col2im\_trans}\left(\texttt{sdmm}\left(\texttt{im2col\_trans}\left(\mathrm{d}\mathbf{O}\right), \texttt{transpose}\left(\mathbf{W}\right)\right)\right)$$

$$WGC: \mathrm{d}\mathbf{W} = \texttt{sdmm}\left(\mathrm{d}\mathbf{O}, \texttt{im2col}\left(\mathbf{I}\right)\right)$$

Here `sdmm` denotes the general `sparse` $\times$ `dense` matrix multiplication.

6 Experimental Results

In this section, experiments are conducted to demonstrate that the proposed approach could reduce the training cost significantly with a negligible model accuracy loss.

6.1 Datasets and Models

Three datasets are utilized including CIFAR-10, CIFAR-100 [31] and ImageNet [32]. AlexNet [6] and ResNet [8] are evaluated while ResNet include Res-{18, 34, 50, 101, 152}. The last layer size of each model is changed in order to adapt them on CIFAR datasets. Additionally for AlexNet, the kernels in first two convolution layers are set as 3×3 with padding $= 2$ and stride $= 1$. For FC-1 and FC-2 layers in AlexNet, they are also resized to 4096×2048 and 2048×2048, respectively. For ResNet, kernels in first layer are replaced by 3×3 kernels with padding $= 1$ and stride $= 1$. Meanwhile, the pooling layer before FC-1 in ResNet is set to Average-Pooling with the size of 4×4.

6.2 Training Settings

All the 6 models mentioned above are trained for 300 epochs on CIFAR-{10, 100} datasets. While for ImageNet, AlexNet, ResNet-{18, 34, 50} are only trained for 180 epochs due to our limited computing resources.

The Momentum SGD is adopted for all training with momentum $= 0.9$ and weight decay $= 5 \times 10^{-4}$. Learning rate `lr` is set to 0.05 for AlexNet and 0.1 for the others. `lr-decay` is set to 0.1/100 for CIFAR-{10, 100} and 0.1/45 for ImageNet.

6.3 Results and Discussions

We set the target pruning rate p defined in Sect. 3.2 varying from 70%, 80%, 90% to 99% for comparison with the baseline. All the training are run directly without any fine-tuning.

Table 1. Evaluation results on CIFAR-10, where `acc`% means the training accuracy and ρ_{nnz} means the average density of non-zeros.

Model	Baseline		$p = 70\%$		$p = 80\%$		$p = 90\%$		$p = 99\%$	
	`acc`%	ρ_{nnz}	`acc`%	ρ_{nnz}	`acc`%	ρ_{nnz}	`acc`%	ρ_{nnz}	`acc`%	ρ_{nnz}
AlexNet	90.50	0.09	90.34	0.01	**90.55**	0.01	90.31	0.01	89.66	0.01
ResNet-18	95.04	1	**95.23**	0.24	95.04	0.22	94.91	0.20	95.18	0.16
ResNet-34	94.90	1	95.13	0.24	95.09	0.21	**95.16**	0.19	95.02	0.15
ResNet-50	94.94	1	**95.36**	0.22	95.13	0.20	95.01	0.17	95.28	0.14
ResNet-101	95.60	1	**95.61**	0.24	95.48	0.22	95.60	0.19	94.77	0.12
ResNet-152	**95.70**	1	95.13	0.18	95.58	0.18	95.45	0.16	93.84	0.08

Table 2. Evaluation results on CIFAR-100, where `acc`% means the training accuracy and ρ_{nnz} means the average density of non-zeros.

Model	Baseline		$p = 70\%$		$p = 80\%$		$p = 90\%$		$p = 99\%$	
	`acc`%	ρ_{nnz}	`acc`%	ρ_{nnz}	`acc`%	ρ_{nnz}	`acc`%	ρ_{nnz}	`acc`%	ρ_{nnz}
AlexNet	67.61	0.10	67.49	0.03	**68.13**	0.03	67.99	0.03	67.93	0.02
ResNet-18	76.47	1	76.89	0.27	**77.16**	0.25	76.44	0.23	76.66	0.19
ResNet-34	77.51	1	77.72	0.24	**78.04**	0.22	77.84	0.20	77.40	0.17
ResNet-50	77.74	1	78.83	0.25	78.27	0.22	**78.92**	0.20	78.52	0.16
ResNet-101	**79.70**	1	78.22	0.23	79.10	0.21	79.08	0.19	77.13	0.13
ResNet-152	79.25	1	**80.51**	0.22	79.42	0.19	79.76	0.18	76.40	0.10

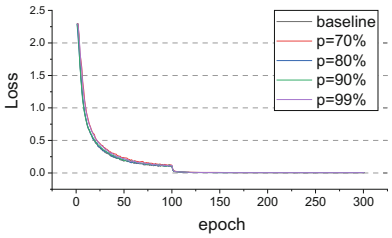
Accuracy Analysis. From Table 1, Table 2 and Table 3 we find that there is no obvious accuracy lost for most situations. And even for ResNet-50 on CIFAR-100, there is 1% accuracy improvement. But for AlexNet on ImageNet, there is a significant accuracy loss when using very aggressive pruning policy like $p = 99\%$. In summary, the accuracy loss is almost negligible when a non-aggressive policy is adopted for gradients pruning.

Gradients Sparsity. The gradients density illustrated in Table 1, Table 2, Table 3 has shown the ratio of non-zero gradients over all gradients, which is related to the amount of calculations. Notice that the output of *DBTD* is the estimation of pruning threshold, so the actual sparsity of each `Conv` layer's activation gradients will be different, and ρ_{nnz} is calculated by dividing the number of non-zero activation gradients by the number of all gradients for all `Conv` layers.

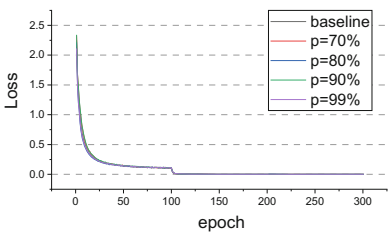
Although the basic block of AlexNet is `Conv-ReLU` whose activation gradients are relatively sparse, our method could still reduce the gradients density for about $5\times \sim 10\times$ on CIFAR-{10, 100} and $3\times \sim 5\times$ on ImageNet. While it comes to ResNet, whose basic block is `Conv-BN-ReLU` and activation gradients are naturally fully dense, our method could reduce the gradients density to

Table 3. Evaluation results on ImageNet, where acc% means the training accuracy and ρ_{nnz} means the average density of non-zeros.

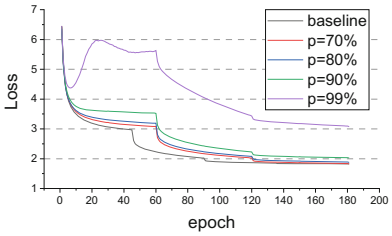
Model	Baseline		$p = 70\%$		$p = 80\%$		$p = 90\%$		$p = 99\%$	
	acc%	ρ_{nnz}	acc%	ρ_{nnz}	acc%	ρ_{nnz}	acc%	ρ_{nnz}	acc%	ρ_{nnz}
AlexNet	56.38	0.07	**57.10**	0.05	56.84	0.04	55.38	0.04	39.58	0.02
ResNet-18	68.73	1	**69.02**	0.34	68.85	0.33	68.66	0.31	68.74	0.28
ResNet-34	**72.93**	1	72.92	0.35	72.86	0.33	72.74	0.30	72.42	0.30



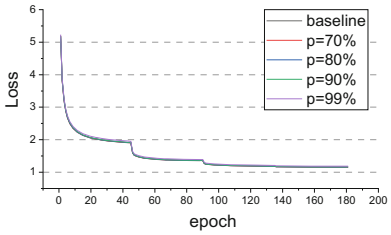
(a) AlexNet on CIFAR-10.



(b) ResNet-18 on CIFAR-10.



(c) AlexNet on ImageNet.



(d) ResNet-18 on ImageNet.

Fig. 3. Training loss of AlexNet/ResNet on CIFAR-10 and ImageNet.

10% $\sim$ 30%. In addition, the deeper networks could obtain a relative lower gradients density, which means that it works better for complicated networks.

Convergence Rate. The training loss is also displayed Fig. 3 for AlexNet, ResNet-18 on CIFAR-10 and ImageNet datasets. Figure 3b and Fig. 3d show that ResNet-18 is very robust for gradients pruning. For AlexNet, the gradients pruning could be still robust on CIFAR-10. However, Fig. 3d confirms that sparsification with a larger p will impact the convergence rate. In conclusion, our pruning method doesn't have significant effect on the convergence rate in most cases. This conclusion accords with the our convergence analysis on Sect. 4.

Acceleration on Desktop CPU. To examine the performance of our proposed approach in practical applications, we implement experiments on low computation power scenarios, where there exists an urgent need for acceleration in the training process. We use 1 core Intel CPU (Intel Xeon E5–2680 v4 2.4 GHz) as computation platform and Intel MKL as BLAS library for evaluation. We set

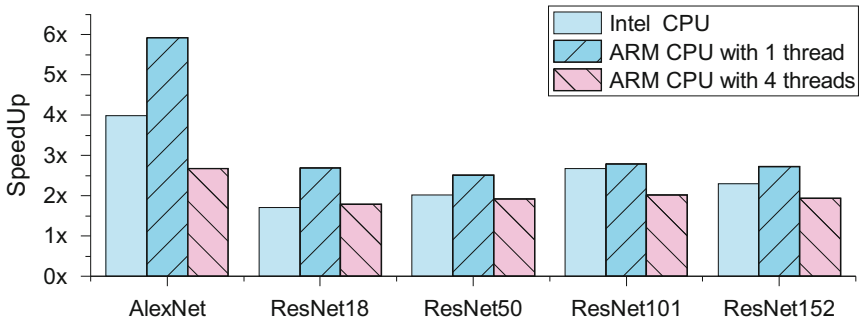


Fig. 4. Speedup evaluation results on CPU. The height of the bar denotes the average acceleration rate of all selected epochs.

$p = 99\%$ for ResNet-{18,50,101,152} and AlexNet on CIFAR-10 dataset and export the d**O**/**I**/**W** from the training process of accuracy evaluation experiment every 50 epochs, and use those data to collect the latency of *AFBP* and *WGC* in our framework. The baseline of this experiment is the original back-propagation implementation of Caffe. According to the results in Fig. 4, our algorithm can achieve $1.71\times \sim 3.99\times$ speedup on average. These speedups refer to the acceleration of back-propagation while the forward stage is not included.

Acceleration on ARM CPU. We further evaluate our approach on ARM platform which is wildly used in edge computing. We choose Raspberry Pi 4B (with ARMv7 1500 Hz) as experimental device and Eigen3 [33] as BLAS library in this experiment because Intel MKL can't be deployed on ARM. In ARM experiment, we use the same setting as the desktop CPU experiment in Sect. 6.3. Besides, we evaluate our approach with both single thread and four threads. According to Fig. 4, in single thread experiments, the speedup of `Conv` Layer's back-propagation stage on ARM platform can be up to $5.92\times$ with AlexNet. As for those networks that use `Conv-BN-RELU` as the basic module such as ResNet-{18,50,101,152}, our approach can also achieve $2.52\times \sim 2.79\times$ acceleration. On the other hand, the acceleration rate decrease in four thread experiment but can still reach $1.79\times \sim 2.78\times$ speedup. The results illustrate that our algorithm still performs well on embedded device which is more urgent in reducing calculation time.

Comparison with Existing Works. Meprop [20] has only experiments on MLP. [21] supplements the CNN evaluation on the basis of Meprop [20]. However, their chosen networks are unrepresentative because they are too naive to be adopted in practical applications. Based on [21], MSBP [22] makes further improvements, which is comparing with our method as illustrated in Table. 4. Our proposed algorithm can achieve higher sparsity than MSBP while keeping a better accuracy than baseline and MSBP on CIFAR-10. More importantly, the experiment result also shows that our work is also well performed on ImageNet which is more challenging but has not been evaluated in existing works.

Table 4. Comparison with MSBP [22]. The network and dataset are ResNet-18 and CIFAR-10. The definition of acc% and ρ_{nnz} can be found in Table 3.

Method	acc%	ρ_{nnz}	Acceleration on Intel CPU	Acceleration On ARM
Baseline	95.08	1	1×	1×
MSBP [22]	94.92	0.4	\	\
Ours	**95.18**	**0.16**	**1.71**×	**2.70**×

7 Conclusion

In this paper, we propose a new dynamically gradients pruning algorithm for CNN training. Different from the existing works, we assume the activation gradients of CNN satisfy normal distribution and then estimate their variance according to their average absolute value. After that, we calculate the pruning threshold according to the variance and a preset parameter p. The gradients are pruned randomly if they are under the threshold. Evaluations on state-of-the-art models have confirmed that our gradients pruning approach could accelerate the back-propagation up to 3.99× on desktop CPU and 5.92× on ARM CPU with a negligible accuracy loss.

References

1. Goyal, P., et al.: Accurate, large minibatch sgd: training imagenet in 1 hour. arXiv preprint arXiv:1706.02677 (2017)
2. You, Y., Zhang, Z., Hsieh, C.J., Demmel, J., Keutzer, K.: Imagenet training in minutes. In: Proceedings of the 47th International Conference on Parallel Processing, p. 1. ACM (2018)
3. Jia, X., et al.: Highly scalable deep learning training system with mixed-precision: training imagenet in four minutes. arXiv preprint arXiv:1807.11205 (2018)
4. Cheng, J., Wang, P.S., Li, G., Hu, Q.H., Lu, H.Q.: Recent advances in efficient computation of deep convolutional neural networks. Front. Inform. Technol. Electron. Eng. **19**(1), 64–77 (2018)
5. Micikevicius, P., et al.: Mixed precision training. arXiv preprint arXiv:1710.03740 (2017)
6. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Proceedings of the Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
7. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
9. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015)

10. Mao, H., et al.: Exploring the regularity of sparse structure in convolutional neural networks. arXiv preprint arXiv:1705.08922 (2017)
11. Anwar, S., Hwang, K., Sung, W.: Structured pruning of deep convolutional neural networks. ACM J. Emerg. Technol. Comput. Syst. **13**(3), 32 (2017)
12. Lebedev, V., Lempitsky, V.: Fast convnets using group-wise brain damage. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2554–2564 (2016)
13. Luo, J.H., Wu, J., Lin, W.: 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)
14. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1389–1397 (2017)
15. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2736–2744 (2017)
16. Wen, W., Xu, C., Wu, C., Wang, Y., Chen, Y., Li, H.: Coordinating filters for faster deep neural networks. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 658–666 (2017)
17. Wen, W., et al.: Learning intrinsic sparse structures within long short-term memory. arXiv preprint arXiv:1709.05027 (2017)
18. Aji, A.F., Heafield, K.: Sparse communication for distributed gradient descent. arXiv preprint arXiv:1704.05021 (2017)
19. Prakash, A., Storer, J., Florencio, D., Zhang, C.: Repr: improved training of convolutional filters. arXiv preprint arXiv:1811.07275 (2018)
20. Sun, X., Ren, X., Ma, S., Wang, H.: meprop: sparsified back propagation for accelerated deep learning with reduced overfitting. In: Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 3299–3308 (2017)
21. Wei, B., Sun, X., Ren, X., Xu, J.: Minimal effort back propagation for convolutional neural networks. arXiv preprint arXiv:1709.05804 (2017)
22. Zhang, Z., Yang, P., Ren, X., Sun, X.: Memorized sparse backpropagation. arXiv preprint arXiv:1905.10194 (2019)
23. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: Proceedings of the International Conference on Machine Learning, pp. 1737–1746 (2015)
24. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: DoReFa-Net: training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160 (2016)
25. Park, E., Yoo, S., Vajda, P.: Value-aware quantization for training and inference of neural networks. In: Proceedings of the European Conference on Computer Vision, pp. 580–595 (2018)
26. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818–2826 (2016)
27. Wen, W., et al.: TernGrad: ternary gradients to reduce communication in distributed deep learning. In: Proceedings of the Advances in Neural Information Processing Systems, pp. 1509–1519 (2017)
28. Bottou, L.: Online learning and stochastic approximations. On-Line Learn. Neural Netw. **17**(9), 142 (1998)
29. Paszke, A., et al.: Automatic differentiation in PyTorch. In: NIPS Workshop (2017)

30. Jia, Y., et al.: Caffe: convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093 (2014)
31. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Technical report, Citeseer (2009)
32. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Li, F.F.: ImageNet: a large-scale hierarchical image database. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255 (2009)
33. Guennebaud, G., Jacob, B., et al.: Eigen v3. http://eigen.tuxfamily.org (2010)