Towards Unified INT8 Training for Convolutional Neural Network

Feng Zhu\(^1\)  Ruihao Gong\(^{1,2}\)  Fengwei Yu\(^1\)  Xianglong Liu\(^2\)*  Yanfei Wang\(^1\)
Zhelong Li\(^1\)  Xiuqi Yang\(^1\)  Junjie Yan\(^1\)
\(^1\)SenseTime Group Limited
\(^2\)State Key Laboratory of Software Development Environment, Beihang University
{zhufeng1, yufengwei, wangyanfei, lizhelong, yangxiuqi, yanjunjie}@sensetime.com
{gongruihao, xlliu}@nlsde.buaa.edu.cn

Abstract

Recently low-bit (e.g., 8-bit) network quantization has been extensively studied to accelerate the inference. Besides inference, low-bit training with quantized gradients can further bring more considerable acceleration, since the backward process is often computation-intensive. Unfortunately, the inappropriate quantization of backward propagation usually makes the training unstable and even crash. There lacks a successful unified low-bit training framework that can support diverse networks on various tasks. In this paper, we give an attempt to build a unified 8-bit (INT8) training framework for common convolutional neural networks from the aspects of both accuracy and speed. First, we empirically find the four distinctive characteristics of gradients, which provide us insightful clues for gradient quantization. Then, we theoretically give an in-depth analysis of the convergence bound and derive two principles for stable INT8 training. Finally, we propose two universal techniques, including Direction Sensitive Gradient Clipping that reduces the direction deviation of gradients and Deviation Counteractive Learning Rate Scaling that avoids illegal gradient update along the wrong direction. The experiments show that our unified solution promises accurate and efficient INT8 training for a variety of networks and tasks, including MobileNetV2, InceptionV3 and object detection that prior studies have never succeeded. Moreover, it enjoys a strong flexibility to run on off-the-shelf hardware, and reduces the training time by 22% on Pascal GPU without too much optimization effort. We believe that this pioneering study will help lead the community towards a fully unified INT8 training for convolutional neural networks.

*corresponding author

Figure 1. The fundamental idea of our unified INT8 training. \(\hat{g}_x\) and \(g_x\) represent the original float gradient and the quantized one, respectively. \(\alpha\) and \(\beta\) represent different direction deviations that quantization brings. The red lines present crash cases when the direction deviation is large. The left subfigure indicates that clipping gradient properly to reduce direction deviation within the convergence boundary can avoid crash. The right subfigure points out that controlling learning rate (step size) could promise a stable parameter updating by counteracting negative effect of deviation.

1. Introduction

Deep convolutional neural networks (DCNNs) have achieved remarkable success in many fields, such as computer vision, natural language processing, etc. However, training and deploying DCNNs usually require a large amount of time cost and power consumption, which is greatly challenging the extensive applications in industry. As a result, many recent studies have been focusing on how to accelerate the inference of neural networks by fixed-point quantization on weights or activations [6, 3, 24, 29, 27, 64, 42, 52, 63, 54, 21, 44, 44, 57], and design dedicated hardware utilizing the efficient integer arithmetic [17, 5, 26, 23]. The successful progress surprisingly shows that the bit-width can be reduced to extremely low such as 4-bit while bringing quite little hurt to the accuracy for inference [15, 59, 13].

Besides inference, low-bit training can also promise considerable acceleration, which further quantizes gradients and utilizes low-bit efficient compute kernel for both the forward and backward propagation. As analyzed in [25], the
computation of backward propagation occupies more time than that of forward propagation. So accelerating the training utilizing low-bit quantization has greater potential when considering the backward process. There has existed 16-bit floating-point (FP16) training, which proves the feasibility of low-bit training \cite{41, 9, 35}. But it is restricted to limited advanced GPUs based on Turing or Volta architecture. Compared with FP16, the 8-bit integer (INT8) operation is widely supported by general GPUs based on Turing, Volta and even low-end Pascal architectures. Besides, the 8-bit integer arithmetic is theoretically and practically 2× faster than FP16 and 4× faster than FP32. Therefore, INT8 training enjoys better efficiency, lower power consumption and better versatility on off-the-shelf hardware.

Despite the attractive benefits, when quantizing gradients to 8-bit, the normal training tends to become unstable, since the distortion of gradients easily misleads the direction of training and causes crash of optimization. This definitely makes INT8 training very difficult, especially for the deep networks. Currently only a few studies have attempted to solve this problem \cite{64, 56, 60, 2, 53, 48}. Unfortunately, all of them just tested limited quantization-friendly networks with high redundancy, and usually require complex structure adjustment or introduce additional operation to reduce quantization error, while significantly increasing the computational complexity. Besides, most of these works lack the theoretical analysis on the ad-hoc tricks, and even worse, none of them reports the practical speedup in the real-world case. All these reasons make the existing INT8 training methods stay far away from the practicality without the universal design.

To build a robust and unified INT8 training framework, we conduct deeper explorations in the challenges of gradient quantization. We empirically find that the distribution of gradients owns four special characteristics: sharp and wide, evolutionary, depth-specific and structure-specific. These unique characteristics make gradient quantization quite different from the naive quantization on weights or activations, and INT8 training more difficult to be stabilized. It is important to understand the behaviors and effects of quantized gradient in the convergence of the training. Therefore, we theoretically establish the convergence bound with respect to the gradient quantization error and the learning rate.

Based on the special characteristics and the theoretical analysis, we propose two universal techniques: Direction Sensitive Gradient Clipping and Deviation Counteractive Learning Rate Scaling to stabilize the INT8 training. The Direction Sensitive Gradient Clipping minimizes the direction deviation by pursuing an appropriate clipping as the training process evolves. Sometimes even if the clipping helps reduce the quantization error, it may still suffer from the accumulated gradient deviations across deep layers. To eliminate this effect, the Deviation Counteractive Learning Rate Scaling is further devised to promise stable parameter updating. The fundamental idea of our method is shown in Figure 1. Extensive experiments on a variety of network structures and tasks prove the superiority and versatility of our method.

Our contribution can be summarized as below:

- We observe four special characteristics on the gradient distribution: sharp and wide, evolutionary, depth-specific and structure-specific, which cause the larger quantization error of gradients.

- We theoretically provide the convergence bound of INT8 training, and respectively devise two universal techniques that can stabilize the INT8 training.

- We are the first to achieve stable INT8 training of various networks such as MobileNetV2/InceptionV3 and various tasks such as object detection, with comparable accuracy to full-precision training.

- We build a flexible and unified INT8 training framework for various tasks using various networks, which can easily replace the original full-precision training.

- We are the first to complete practical acceleration of INT8 training on low-end GPUs with Pascal architecture, i.e., NVIDIA GeForce GTX 1080Ti, achieving about 22% speedup without too much optimization.

2. Related Work

Compared to huge amount of studies on accelerating inference by model quantization \cite{45, 62, 7, 52, 11, 40}, there are few works exploring quantized training including backward propagation comprehensively. DoReFa-Net \cite{64} quantizes gradients to 4 and 6 bits, but only experiments AlexNet with low precision gradient. WAGE \cite{56} and WAGEUBN \cite{60} quantize gradient to 8-bit integer, but they both incur considerable loss of accuracy (greater than 5%). RangeBN \cite{2} and FP8 training \cite{53} achieve accuracy comparable to full-precision models, but they both use floating-point number in gradients, which is not beneficial for hardware optimization to boost the speed. Besides quantized training, most low-precision training research keeps gradient precision in 16-bit floating-point. Flexpoint \cite{33}, MPT \cite{41} and DFP \cite{9} all use 16-bit floating-point to train DNNs with accuracy comparable to full-precision model. To perform more efficient training of neural networks, INT8 training has more advantages over FP16 training.

3. Unified INT8 Training

In this paper, we aim to build a unified INT8 training framework, which utilizes 8-bit integer arithmetic to accelerate the expensive training process of deep neural networks including both the forward and backward propagation.
Figure 2. Crashed training of MobileNetV2 on CIFAR-10 after quantizing gradients to 8-bit.

3.1. Preliminaries

Symmetric uniform quantization[30] is the most efficient scheme among existed quantization methods, due to its hardware-friendly computation. Therefore, to guarantee the acceleration performance, we build the INT8 training framework based on it. Given the data \( x \) (i.e., weights, activations, and gradients) following in the range \((l, u)\) and a clipping value \(c \in (0, \max(|l|, |u|)]\), the symmetric uniform quantization can be formulated as:

\[
q = \text{round}(\frac{\text{clip}(x, c)}{s}),
\]

where \(\text{clip}(x, c) = \min(\max(x, -c), c)\), \(s = 2^8 - 1\) indicates the scaling factor to project the floating-point number to fixed-point 8-bit integer, and \(q\) represents the quantized fixed-point number. Subsequently, the corresponding dequantized data \( \hat{x} \) can be calculated by:

\[
\hat{x} = q \cdot s. \tag{2}
\]

Different from most prior studies that mainly focus on speeding up the inference (i.e., the forward propagation), our INT8 training framework attempts to further accelerate the backward propagation during the training stage, by applying quantization to the gradients. Namely, we pursue the quantize-dequantized gradients \( \hat{g} \) from full-precision gradients \( g \) in a proper way.

To ensure the quantized gradients maintain an unbiased expectation compared with the original ones, we adopt the stochastic rounding following [16]:

\[
\text{round}_x(x) = \begin{cases} [x], & \text{w.p. } 1 - (x - [x]) \\ [x] + 1, & \text{w.p. } x - [x] \end{cases}. \tag{3}
\]

Unfortunately, although the stochastic rounding technique limits the quantization error to some extent from the statistical view, the perturbation for each training iteration is still inevitable and harmful for convergence, whose reasons will be discussed in the following section.

3.2. Challenges of Gradient Quantization

Gradients determine the direction of optimization and the magnitude of parameter update and thus play a critical role in pursuing high accurate models. In INT8 training, after we apply quantization to gradients, the perturbation introduces deviation to the optimization direction. Once the deviation accumulates to an unacceptable degree, the training process may be unstable and even crash, resulting in severe performance degradation. Figure 2 shows our empirical observation that for some special network architectures like MobileNetV2, directly quantizing gradients causes a rapid crash of training.

To further investigate the essential reasons behind this phenomenon, we conduct detailed analysis on the distribution of gradients during training without gradient quantization, as shown in Figure 3. We surprisingly observe that the gradients own the following unique characteristics:

C1: Sharp and Wide. As shown in Figure 3(a), compared to weights and activations, gradients follow an unusual distribution that has more values concentrated around zero while a certain number of extreme values also ex-
ists. Therefore, the distribution curve is very sharp with small values taking the majority of gradients, but the range is relatively very wide. This makes many gradients quantized to zero and the quantization error significantly large when using uniform quantization.

C2: Evolutionary. Figure 3(b) depicts how the gradient distribution of the same layer evolves with respect to the training iterations. We can find that as the training goes on, the shape of gradient distribution becomes much sharper and narrower, which means it is impossible to fix the quantization settings throughout the training process, as we usually do for weights and activations, such as assuming the same clipping range in the whole training.

C3: Depth-Specific. Figure 3(c) compares the distribution of gradients in different layers at the same epoch. It is obvious that the distributions at the shallow layers are sharper with larger extreme values than the deeper layers. This means that the preceding layers of the neural networks often face more severe quantization loss.

C4: Structure-Specific. As can be seen in Figure 3(d), the gradients of layers with different structures at the same epoch present apparently different patterns. For MobileNetV2, the second convolutional layer in each block is of depth-wise structure. Its gradients own larger range and sharper shape even in the deeper block, making MobileNetV2 harder to quantize from the aspect of gradients.

Based on the above observations, we can conclude that the gradients differ from weights and activations largely, which inevitably causes an unstable training, when simply adopting the common quantization techniques for weights and activations. This means that we need certain techniques to take care of distinctiveness in gradient quantization, which brings great challenges to the real and unified INT8 training in practice.

Before turning to devise the desired techniques considering the speciality of gradients, we first attempt to understand the gradient’s effect on the training stability, by theoretically revealing the connections between training convergence and gradient quantization. This will provide us a reliable clue to build the robust and unified INT8 training framework.

3.3. Stabilize Training: A Theoretical Perspective

As commonly used in the analysis of deep learning optimizers [12, 28, 46, 39], the ability of convergence is usually evaluated by the regret $R(T)$.

$$R(T) = \sum_{t=1}^{T} (f_t(w_t) - f_t(w^*)),$$  \hspace{1cm} (4)$$

where $T$ indicates the number of iterations. $w_t \in \mathbb{S}$ is the parameter at time $t$ in the convex compact set $\mathbb{S}$, and $f_t(w_t)$ denotes the corresponding loss function. The optimal parameter is represented by $w^*$. If the average regret $\frac{R(T)}{T}$ approaches zero quickly as $T$ increases, the speed and ability of convergence can be guaranteed.

Due to the complexity of the DCNNs, it is very difficult to directly analyze its behaviors. As the prior studies [1, 34, 22, 61] do, we first make the following assumptions:

Assumption 1. $f_t$ is convex;

Assumption 2. $\forall w_i, w_j \in \mathbb{S}$, $\|w_i - w_j\|_\infty \leq D_\infty$.

Although the convexity assumption may not hold for deep networks, analysis based on this can provide reasonable and valuable insights for us, which has been proved in previous studies [12, 39, 22, 61].

Taking the standard stochastic gradient descent algorithm into consideration, the optimization based on quantized gradient $\hat{g}_t$ and learning rate $\eta_t$ can be formulated as:

$$w_{t+1} = w_t - \eta_t \hat{g}_t.$$  \hspace{1cm} (5)$$

Then we have the following theoretical finding (see the supplementary materials for detailed proof):

Theorem 1. If define the error of quantized gradients as $\epsilon_t = g_t - \hat{g}_t$ and the dimension of weights as $d$, then with assumption 1 and 2, we have:

$$\frac{R(T)}{T} \leq \frac{dD^2}{2T\eta_r} + \frac{D_\infty}{T} \sum_{t=1}^{T} \|\epsilon_t\|_2 + \frac{1}{T} \sum_{t=1}^{T} \frac{\eta_t}{2} \|\hat{g}_t\|^2.$$  \hspace{1cm} (6)$$

We can find that the bound of average regret is dominated by three terms. Term (1) approaches zero as $T$ increases and thus can be ignored in gradient quantization. Term (2) indicates the quantization error of gradients greatly affects the ability to converge, and it is usually large, as analyzed in Section 3.2. For term (3), its magnitude is mainly influenced by the learning rate and 2-norm of quantized gradients. Based on the theoretical analysis, to stabilize INT8 training, we have two basic principles for designing better quantization techniques: (1) reduce the quantization error of gradients; (2) scale down the learning rate. They are also very intuitive since, on the one hand, a lower quantization error means small deviation of optimization direction and thus avoids the training crash, on the other hand, it is a common sense that decreasing the learning rate gradually promises a better solution in the optimization.

Now with the design principles, the question is how to devise the universal techniques for INT8 training, meanwhile take the characteristics of gradients into consideration. We respectively present two novel techniques: Direc-
Table 1. KS-statistics of gradient and weight with respect to different layers’ conv3 in MobileNetV2, the last column indicates the maximum value that can accept the hypothesis at significance level of 0.05.

<table>
<thead>
<tr>
<th>Data</th>
<th>Distribution</th>
<th>Critical value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gaussian</td>
<td>Laplace</td>
</tr>
<tr>
<td>layer0</td>
<td>g</td>
<td>0.1934</td>
</tr>
<tr>
<td></td>
<td>w</td>
<td>0.0391</td>
</tr>
<tr>
<td>layer8</td>
<td>g</td>
<td>0.2061</td>
</tr>
<tr>
<td></td>
<td>w</td>
<td>0.0294</td>
</tr>
</tbody>
</table>

3.4. Direction Sensitive Gradient Clipping

Considering the basic operation \( z = W^T a \) in deep neural networks, the gradients of weights \( g_w \) actually can be calculated by \( g_z = a \). From this aspect, the quantization error of \( g_w \) in \( 6 \) mainly stems from that of activation gradients \( g_z \). Therefore, in our INT8 training we can mainly concern the quantization of \( g_z \), which will help control the error of quantized gradients in \( 6 \). For simplicity of notations, in the following discussion we directly use \( g \) to denote \( g_z \).

To minimize quantization error, previous works mainly seek the optimal clipping value \( c \) in \( 1 \) by assuming certain data distribution, e.g. Gaussian distribution \([3, 4, 20, 2, 22, 11, 55, 58]\). However, according to the gradient characteristics C1 and C2 we discover, it is impractical to make a common assumption for an evolutionary and unusual gradient distribution. To further prove this point, we do the Kolmogorov–Smirnov test\([50]\) with distribution parameter solved by maximum likelihood estimation, and report the KS-statistics that consistently reject the assumption that gradients obey any common distribution in Table 1.

To find the optimal clipping value \( c \) without any assumption, a straightforward idea is to keep the quantized gradient consistent with the original one by gradient descent algorithm. Usually, one can model the consistency using the popular mean-square error (MSE). Unfortunately, due to characteristics C2 of gradients with huge discrepancy and fluctuation in their magnitudes, MSE makes the optimization vulnerable and unable to work under the same simple setting across various layers.

Therefore, to pursue the desired clipping values of different layers that promise stable training, we choose cosine distance to guide the learning of clipping values, which not only avoids the negative effect of the varied gradients’ magnitudes, but also keeps the network optimization directions consistent:

\[
d_c = 1 - \cos(<g, \hat{g}>) = 1 - \frac{g \cdot \hat{g}}{|g| \cdot |\hat{g}|}
\]

where \( g \) and \( \hat{g} \) denote the original floating-point gradient and its quantize-dequantized counterpart.

The cosine distance measures the direction deviation of quantized gradients, and the strong correlation between cosine distance and training stability will be verified in the supplementary materials. By minimizing the cosine distance, we subsequently reduce term \( 2 \) in \( 6 \). Figure 4(a) shows the quantization error using different clipping values, where there exists an optimal clipping value that substantially reduces the cosine distance.

3.5. Deviation Counteractive Learning Rate Scaling

The theoretical analysis on convergence ability of quantized training indicates the necessity of scaling down learning rate, since the quantization error of gradients cannot vanish completely. To validate this point, we decrease the learning rate of the original crashed training of MobileNetV2 mentioned in Section 3.2 and find that it defers and even eliminates the crash with an extremely low learning rate, although facing a performance degradation (see the red, green and orange lines in Figure 4(b)).

Since the gradients are backward propagated layer by layer, the minor gradient deviation will accumulate exponentially after massive multiplication and addition calculation. To address this issue, we further propose the Deviation Counteractive Learning Rate Scaling to balance out the error by exponentially decaying the learning rate according to the degree of direction deviation \( d_c \), the scaling function is formulated at:

\[
\phi(d_c) = \max(e^{-\alpha d_c}, \beta)
\]

where \( \alpha \) controls the decay degree and \( \beta \) limits the lower bound of scaling.

This scaling function generates a factor to scale down the original full-precision learning rate. We empirically find that the self-adapting scaling function performs well in a layer-wise way, adaptively adjusting the learning rate according to the direction deviations in different layers. This counteracts the undesired effects of the gradient deviations across layers, and exactly addresses the challenges of the
depth-specific and structure-specific patterns as observed in characteristics C3 and C4 in Section 3.2. The blue line in Figure 4(b) demonstrates that the training equipped with $\phi(d_c)$ scaling achieves higher accuracy than the manually adjusted ones (tested with MobileNetV2 on CIFAR-10).

3.6. General Purpose Training Framework

In addition to ensuring the stable and accurate convergence, in practice our unified INT8 training framework should also satisfy the following three features:

(1) **Easy to plug into any DCNN architecture.** To realize this, we implement an automatic match and replacement mechanism in PyTorch [43] that correspondingly substitutes convolutional and fully-connected layers with 8-bit counterpart. The whole workflow including both forward and backward passes is shown in Figure 5.

(2) **No excessive extra computational overhead.** To avoid the extra time cost of calculating clipping value, we design a Periodic Update method to optimize the clipping value periodically. As we can see in Table 2, the Periodic Update method dramatically reduces the computational overhead of optimizing the clipping value.

(3) **Easy to implement on off-the-shelf hardware.** To validate the potential of that, we utilizes the DP4A instruction on low-end GPUs with Pascal architecture to implement efficient 8-bit kernels for calculating gradients.

4. Experiments

We conduct extensive experiments to demonstrate that our proposed framework is unified for various network structures on popular image classification and object detection tasks with state-of-the-art accuracy, and meanwhile it can be easily deployed on the mainstream devices (NVIDIA Pascal GPU) with satisfactory speedup, compared to full-precision training.

4.1. Ablation Study

**Settings.** We first conduct the ablation study on CIFAR-10 dataset with MobileNetV2 [49], to validate the effectiveness of the proposed techniques. We use cosine scheduler [1] with initial learning rate set to 0.1 for all experiments. In the Periodic Update experiment, the $\alpha$ and $\beta$ in learning rate scaling are set to 20 and 0.1 respectively.

**Direction Sensitive Gradient Clipping.** Figure 6(a) shows the cosine distance with respect to the training steps. We can observe that conv2 (the second convolutional layer) of each block owns a much larger cosine distance than other layers most of the time. This is consistent with C4 that the gradients of conv2 own sharper shape, indicating that cosine distance can well reflect the gradient characteristics.

Moreover, as Table 3 lists, our proposed direction sensitive gradient clipping technique indeed prevents INT8 training from crashing, which proves the fact that optimizing a clipping value of gradients to minimize direction deviation $d_c$ can certainly ensure a stable INT8 training.

**Deviation Counteractive Learning Rate Scaling.** We
evaluate three forms of learning rate scaling strategies without clipping to control variable for a reasonable comparison. The results shown in Figure 6(b) reveal that linear and quadratic forms are too weak to control optimization direction within the convergence boundary and model crashes in the training process. Compared with linear and quadratic form, the scaling with exponential form is more powerful to counteract the direction deviation and prevents optimization from stepping out of the convergence boundary. We further explore its sensitivity to the selection of hyperparameter in Table 4, and we can see that different settings of $\alpha$ and $\beta$ achieve similar accuracy, which presents the stability of our Deviation Counteractive Learning Rate Scaling.

Table 4. Comparison of different hyperparameters for learning rate scaling.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>10</th>
<th>10</th>
<th>20</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta$</td>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>Accuracy (%)</td>
<td>92.82</td>
<td>93.28</td>
<td>93.38</td>
<td>93.27</td>
</tr>
</tbody>
</table>

Periodic Update for clipping value. To reduce the extra computational overhead, we increase the period to update clipping value and find that it brings little hurt to the accuracy, as shown in Table 5. This empirical conclusion brings possibilities for the practical acceleration of INT8 training. Besides, here we apply both gradient clipping and learning rate scaling, and obtain better performance (see that with period 1) than those in Table 3 and 4. This further verifies the positive effects of the two general techniques.

Table 5. Ablation study on update period.

<table>
<thead>
<tr>
<th>Period</th>
<th>1</th>
<th>10</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy (%)</td>
<td>93.66</td>
<td>93.07</td>
<td>93.38</td>
<td>92.75</td>
</tr>
</tbody>
</table>

4.2. Image Classification

Now we consider the popular image classification task that most prior studies choose to evaluate the quantization performance. We experiment with AlexNet [32], ResNet [19], MobileNetV2 [49] and InceptionV3 [51] on CIFAR-10 [31] and ImageNet (ILSVRC2012) [10]. The CIFAR-10 dataset contains a training set of 50K images and a testing set of 10k images. Each image is of size 32×32 with 10 classes. ImageNet (ILSVRC2012) consists of 1.2 million training images and 50K test images with 1000 classes.

Settings. As for the hyperparameters of ResNet, we use the same settings described in [19]. For other neural networks, we use cosine scheduler [1] with initial learning rate set to 0.1. The $\alpha$ and $\beta$ in learning rate scaling are set to 20 and 0.1 respectively. Clipping value is updated per 100 iterations for all experiments.

CIFAR-10. As Table 6 shows, our method achieves comparable accuracy on ResNet-20 to FP8 training, but takes much less memory and computation consumption due to the fixed-point operation. Moreover, our method performs surprisingly good on MobileNetV2 (1.01% accuracy drop) and InceptionV3 (even better than full precision model).

ImageNet. Table 7 lists existing state-of-the-art quantized training methods including WAGE [56], WAGEUBN [60] and FP8 training [53]. For AlexNet INT8 training, our method obtains 5.84% improvement over DoReFa-Net [64]. Free from the extra overhead like tanh, our method enjoys higher efficiency than DoReFa-Net. As for the 2-bit weight and 8-bit activation/gradient case, we significantly outperform WAGE with about 3% accuracy gain. What’s more, equipped with our method, the INT8 training for ResNet architecture achieves almost no performance degradation, while none of the previous studies has done that. Compared with the FP8 training method, our method improves the accuracy by nearly 3%. It should be noted that we can directly get a real speedup on popular off-the-shelf devices while methods like FP8 training need specially designed hardware, which means that our framework is more general for unified training acceleration.

As analyzed in [36], the convolutional layer occupies most of the training time while other layers like BatchNorm and ReLU are not computation-intensive. Therefore, we mainly focus on quantizing convolutional layers currently and do not quantize BatchNorm layer like RangeBN [2] and WAGEUBN [60]. Even so, there is still a significant speedup for INT8 training. In addition, we could get comparable accuracy to full precision training, much higher than RangeBN and WAGEUBN.

Networks using INT8 training for the first time. To our best knowledge, we are the first to quantize gradient of MobileNetV2, which is known to be difficult in this community. Our method gets very good performance on both CIFAR-10 and ImageNet datasets using MobileNetV2, with only around 1% accuracy loss. We also try INT8 training on InceptionV3 for the first time, and achieve comparable accuracy to full precision model. Note that for InceptionV3 on CIFAR-10, our INT8 training method can even achieve better performance than the full-precision model.

4.3. Object Detection

To prove the versatility of our method, we further conduct experiments with the popular object detection networks including Faster-RCNN [47], RFCN [8] and RetinaNet [37] on two widely used datasets: PASCAL VOC [14] and COCO [38]. The PASCAL VOC dataset consists of 11k images with 20 classes. The COCO dataset contains more than 20k images and 80 object categories. Note that
we are the first to successfully achieve INT8 training on the object detection task.

Settings. As for the hyperparameters, we follow the same rules described in [35]. And α and β for learning rate scaling are the same as those used in image classification task.

PASCAL VOC. We test RFCN and Faster R-CNN with different backbones, and find that quantized training equipped with our method only suffers a very slight detection accuracy (mAP) drop. The result of RFCN shows that even for a deeper backbone such as ResNet-101, our INT8 training still maintains almost the same accuracy as full-precision.

COCO. On the large scale COCO dataset, we experiment with RetinaNet (one-stage) and Faster R-CNN (two-stage). Our method performs stably with less than 1.8% accuracy degradation on both networks. We find that RetinaNet incurs higher mAP loss than Faster R-CNN, which is inconsistent with the conclusions in the previous study [35]. This may be caused by the fact that the focal loss used in one stage detector is more sensitive to gradient quantization.

4.4. Speed Result on NVIDIA GPU

None of the existing libraries can directly support the complete INT8 training. Thus we implement it by ourselves on NVIDIA Pascal GPU using DP4A instruction to verify the acceleration power of our method. Table 10 shows that in the forward process using our solution, INT8 can bring an average 1.63× speedup, while in the backward process, it can achieve a higher 1.94× speedup. Even if we only replace the FP32 convolutional layer with the slightly optimized INT8 one, the training time for ResNet-50 can be reduced by about 22%. More details about speed result are included in the supplementary materials.

5. Conclusions

In this paper, we attempt to build an INT8 training framework for common DCNNs. We found four distinctive characteristics of gradients and then gave two theoretical principles stabilizing training with the convergence bound. Based on that, we proposed Direction Sensitive Gradient Clipping and Deviation Counteractive Learning Rate Scaling. Extensive experiments prove the versatility of our method for various networks and tasks. We reduced the training time by 22% on Pascal GPU with only trivial optimization. If each layer is sufficiently optimized, the training will achieve higher speedup and lower memory consumption. We hope our first successful attempt can help lead the community towards a fully unified INT8 training.

Acknowledgement

This work was supported by National Natural Science Foundation of China (61872021, 61690202), Beijing Nova Program of Science and Technology (z19110000119050), and SenseTime Group Limited.
References


