2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)

Fixed-Point Back-Propagation Training

Xishan Zhang[1,2], Shaoli Liu*[1], Rui Zhang[1,2], Chang Liu[1], Di Huang[1], Shiyi Zhou[1], Jiaming Guo[1],
Qi Guo[2], Zidong Du[1,2], Tian Zhi[1,2] and Yunji Chen[2]

[1]Cambricon Technologies Corporation Limited, Beijing, China
[2]State Key Lab of Computer Architecture, Institute of Computing Technology, CAS, Beijing, China

Abstract

Recent emerged quantization technique (i.e., using low bit-width fixed-point data instead of high bit-width floating-point data) has been applied to inference of deep neural networks for fast and efficient execution. However, directly applying quantization in training can cause significant accuracy loss, thus remaining an open challenge. In this paper, we propose a novel training approach, which applies a layer-wise precision-adaptive quantization in deep neural networks. The new training approach leverages our key insight that the degradation of training accuracy is attributed to the dramatic change of data distribution. Therefore, by keeping the data distribution stable through a layer-wise precision-adaptive quantization, we are able to directly train deep neural networks using low bit-width fixed-point data and achieve guaranteed accuracy, without changing hyper parameters. Experimental results on a wide variety of network architectures (e.g., convolution and recurrent networks) and applications (e.g., image classification, object detection, segmentation and machine translation) show that the proposed approach can train these neural networks with negligible accuracy losses (-1.40%∼1.3%, 0.02% on average), and speed up training by 252% on a state-of-the-art Intel CPU.

1. Introduction

While deep neural networks have become state-of-the-art techniques for a wide range of machine learning applications, such as image recognition [14], object detection [21], machine translation [32, 8], the computation costs of deep neural networks are continuously increasing, which greatly hampers the development and deployment of deep neural networks. For example, 10,000 GPU hours are used to perform neural architecture search on ImageNet [2]. Quantization is a promising technique to reduce the computation cost of neural network training, which can replace high-cost floating-point numbers (e.g., float32) with low-cost fixed-point numbers (e.g., int8/int16). Recently, both the software society [6, 12, 16, 19, 27, 35] and the hardware society [11, 24, 23, 31] have carried out extensive researches on quantization of deep neural network for inference tasks.

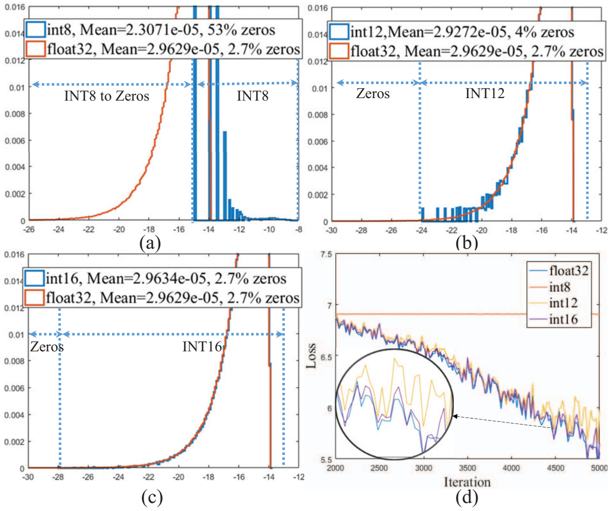


Figure 1: AlexNet fc2 layer activation gradient distribution (base-2 logarithm) and training convergence.

Though various investigations have demonstrated that deep learning inference can be accurately performed with low bit-width fixed-point numbers through quantization [16, 17, 6, 33, 35, 35, 37], the quantified training remains an open challenge. Some existing approaches quantify the backward-pass to low-bit (e.g., int8) but incur significant accuracy drop, for examples, 3˜7% loss for AlexNet [38, 36]. [7] uses int16 for both forward-pass

*Shaoli Liu is the corresponding author of this paper (Email: liushaoli@cambricon.com). This work was funded and supported by Cambricon Technologies.

2575-7075/20/$31.00 ©2020 IEEE
DOI 10.1109/CVPR42600.2020.00240

Authorized licensed use limited to: Cornell University Library. Downloaded on August 15,2020 at 14:00:39 UTC from IEEE Xplore. Restrictions apply.

and backward-pass to ensure accuracy. However, there is no guarantee that unified int16 precision works for all the tasks and networks. Compared to efficiency, accuracy is the primary pursuit in most practical training-from-scratch scenarios, so activation gradients in back propagation are still kept as float32 for most quantification work [16, 17, 6, 35, 39, 37]. Our aim is to study the correlation between training accuracy and calculation bit-width (efficiency) in back-propagation.

Most previous investigations on quantified training use unified precision (i.e., bit-width) for all network layers. Intuitively, using mixed precisions for different layers will promote the network performance. However, it is hard to find the most appropriate precisions for so many layers in so many training iterations. Considering a widely used ResNet50 model, with 4 candidate quantization bit-widths (e.g., 8, 16, 24, 32 for weights, activations and activation gradients), the size of quantization precision combination search space for 450,000 training iterations can achieve $4^{3*50*450,000}$.

To avoid prohibitively long space searching of quantization bit-width combinations, we propose an efficient and adaptive technique to determine the bit-width layer by layer separately, which is based on our observation about the relationship between the layer-wise bit-width and the training convergence. Take AlexNet as an example, Figure. 1(a-c) depicts the distributions of activation gradients on AlexNet last layer when quantified with different bit-widths. Compared with the original float32, int8 introduces a significant change in data distribution, int12 introduces slightly change of data mean, and int16 shows almost the same distribution with float32. Figure. 1(d) depicts the corresponding training loss, which shows int8 quantization does not converge at beginning, int12 convergences slower than float32 and int16 behaves similar as float32. The above experimental results suggest if a quantization resolution does not change the data distribution of a layer (e.g., int16 for the last layer of AlexNet), quantified training with this resolution for the corresponding layer will almost keep the training accuracy.

Based on the above observation, one can train large-scale deep neural network using fixed-point numbers, with no change of hyper parameters and no accuracy degradation. For each layer in training, our approach automatically finds the best quantization resolution (i.e., the smallest bit-width which does not significantly change the data mean) for weights, activations and activation gradients respectively. Concretely, we first calculate the mean of the data before quantization. Then, we quantify the data using int8 and calculate the quantization error. If the ratio of quantization error exceeds a threshold (e.g., 3%), the quantization bit-width is increased. The above process is looped until the quantization error ratio is below the threshold.

We evaluate our approach on a wide variety of network architectures (e.g. convolution and recurrent networks) and applications (e.g. image classification, object detection, segmentation and machine translation). Our approach quantifies all weights and activations to int8. On average, 12.56%, 87.43% and 0.07% of activation gradients are quantified to int8, int16, and int24 respectively. Experimental results show that the proposed adaptive fixed-point training approach can achieve comparable accuracy with float32 for training from scratch. The accuracy loss is only 0.02% on average (-1.40%∼1.3%). Results on Intel Xeon Gold 6154 shows that the proposed approach can achieve 2.52 times speedup over float32 training for AlexNet.

We highlight three major contributions of the proposed adaptive fixed-point training:

1. **Flexibility:** The quantization precisions for different layers of different networks are automatically adapted to guarantee the network accuracy.

2. **Efficiency:** We quantify both the backward-pass and forward-pass with fixed-point numbers in training, which can accelerate training on real hardware. After training, int8 weights can be directly deployed, so no further quantification is needed.

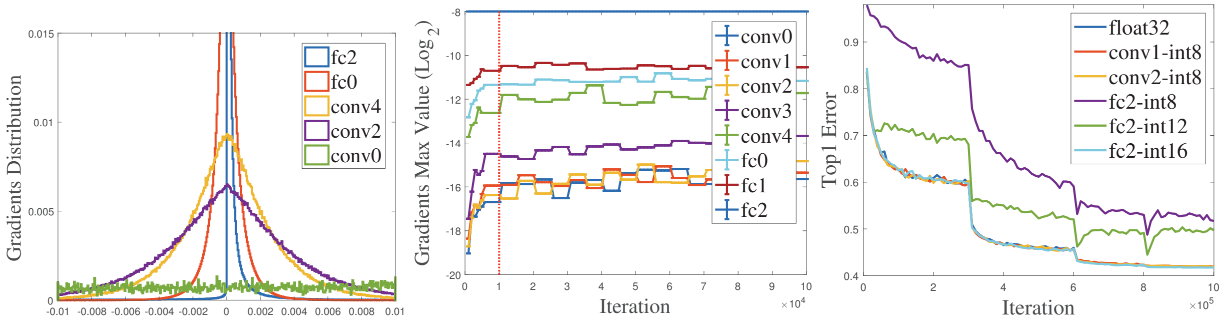
3. **Generalization:** Evaluations on various networks and applications demonstrate the proposed adaptive fixed-point training is effective and practical.

2. Related Works

Using reduced precision for deep learning has been an active research topic. Prior efforts explore floating-points(e.g., 8-bit and 16-bit) for training [34, 22] and maintain accuracy on a spectrum of deep learning models and datasets. However, as floating-point is more resource-intensive than fixed-point, the deployments always rely on quantization techniques.

There are recent attempts quantifying weight and activation on different layers with different bit-widths. For the inference of a trained network, there are some techniques that heuristically search the space of quantization bit-width combinations [35, 33, 37]. However, these inference techniques only need to consider single iteration, whose search space is much smaller than training. Hence, they are unsuitable for training. For training, there are some differentiable quantization methods [4, 30, 37], introducing extra calculations to learn the quantization parameters (e.g., quantization resolution, range and bit-width) with gradient descent. However, the quantization parameters for backward propagation are hard to learn using differentiable methods. [26] quantifies the backward propagation. Different from their method, which assigns layer-wise bit-width before training, our approach dynamically changes the bit-width during training and we evaluate on widely used networks.

Authorized licensed use limited to: Cornell University Library. Downloaded on August 15,2020 at 14:00:39 UTC from IEEE Xplore. Restrictions apply.



(a) Activation gradient distribution.

(b) Activation gradient evolution.

(c) Training curve of AlexNet.

Figure 2: Observations on AlexNet.

Researchers have shown that 16-bit is sufficient for back propagation in most vision training tasks [7]. However, further quantization to 8-bit results in severe degradation [38, 38, 36, 7, 1]. WAGE [36] claims that first and last layers require higher precision. TBP [1] shows weight gradient computation (WTGRAD) needs more bits than gradient back propagation (BPROP).

Our approach is different from others in three aspects. First, fixed-point is used in both forward-pass and backward-pass for training. Second, the quantization parameters for different layers are dynamically adapted to guarantee the accuracy. Lastly, we train a variety of vision and natural language processing applications on large scale dataset.

3. Observation

The key of fixed-point training is to find proper quantization parameters that ensure the training accuracy. Therefore, we study the relationship between the ever-changing data distribution of different layers and the training convergence.

Observation 1. Data distribution varies greatly between layers. Figure. 2a depicts the distributions of activation gradients of different layers on AlexNet. The majority of activation gradients concentrate in areas close to zero, and have long tail distributions. Compared to convolution layers, the fully connected layers have larger variances. Figure. 2b shows the base-2 logarithm of max absolute value of activation gradients on AlexNet, the max value on bottom layers (e.g., conv0, conv1, conv2) is smaller than the max value on upper layers(e.g., fc0, fc1, fc2). Intuitively, for those layers whose range of data is wide and distribution is centralized, higher quantization resolutions are demanded.

Observation 2. Data range of each layer changes during training. Figure. 2b shows the max absolute value of activation gradient evolution during training. At the early stage of training (less than 10,000 iterations, as shown on the left side of the red line), the data range changes rapidly, and after one or two epochs, the data range tends to be stable. This phenomenon suggests that when training from scratch, the quantization range should also be changed frequently within the initial epochs .

Observation 3. Data with large variance requires large bit-width. Figure. 2c shows the convergence curves using different bit-width of different layers. Float32 is the training convergence curve of using float32 for all the convolution and fully connected layers. After 5,000,000 iterations, the network's top1 accuracy on ImageNet is 58.00%. Then, we quantify the activation gradients of conv1 to int8 and keep other layers float32. The training curve of conv1-int8 is the same as float32 and the final top1 accuracy is 58.01%. However, when we quantify the activation gradients of fc2 to int8 and keep other layers float32 unchanged, the training convergence speed is significantly slower than float32, and within the first 5,000 iterations the training does not converge. The final top1 accuracy of fc2-int8 is only 48.27%. When quantifying the activation gradients of fc2 to int12, the training convergence speed is faster than int8 but still slower than float32. The final top1 accuracy of fc2-int12 is only 50.30%. Using int16 for the activation gradients of fc2, finally the training curve is the same as float32 with 58.28% top1 accuracy. In conclusion, int8 is enough to quantify the activation gradient of conv2, however, fc2 requires int16 to maintain the training accuracy. Together with the observation1, we find that data with large variance requires large bit-width, thus the quantization parameters should be dynamically determined by the data distribution.

According to network initialization principle [10, 13], all network parameters are initialized as Gaussian distribution with variance relating to the hyper-parameters of layers. Similar network initialization principle and similar SGD learning algorithm ensure that various network architectures should have similar observations.(More observations are in Appendix C.)

Authorized licensed use limited to: Cornell University Library. Downloaded on August 15,2020 at 14:00:39 UTC from IEEE Xplore. Restrictions apply.

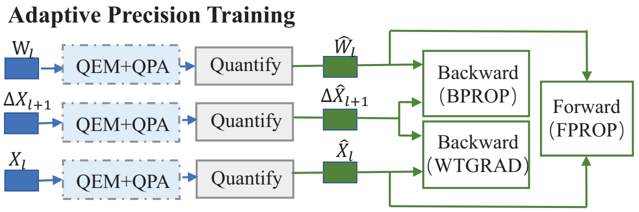


Figure 3: Adaptive fixed-point training for one iteration one layer. The green nodes and blocks indicate fixed-point data and calculations. Only 0.01%∼2% of the iterations activate the QEM and QPA components.

4. Adaptive Fixed-Point Training

In this section, we introduce the adaptive fixed-point training approach as shown in Figure. 3. In training, the main three computing units of single iteration include forward-pass (FPROP), backward-pass for gradient propagation (BPROP) and backward-pass for weight gradient computation (WTGRAD). The inputs of these three units include weight W_l, activation X_l and top layers' activation gradient ΔX_{l+1} of linear layer l. In adaptive fixed-point training, we quantify these three inputs to fixed-point numbers[1]. The quantification parameters, such as bit-width n and quantization resolution r are automatically determined by the proposed Quantization Error Measurement (QEM) and Quantification Parameter Adjustment (QPA) .

In the following part of this section, we will introduce two main components QEM and QPA of our training approach. Algorithm. 1 describes the entire adaptive fixed-point training algorithm. The output of QEM (denoted as $Diff$) serves as an explicit indicator for insufficiency of quantization resolution according to data distribution. QPA performs quantization parameter update and determines update frequency (denoted as Itv) according to the output of QEM.

Algorithm 1 Adaptive fixed-point training. Data such as weights W_l, activations X_l and top layers' activation gradients ΔX_{l+1} of the linear layer l are quantified to fixed-point numbers with different bit-widths n and quantization resolution r. The output $Diff$ of QEM indicates the insufficiency of quantization resolution, and the output Itv of QPA determines quantization parameter update frequency.

```
Initial all $update\_iter = 1$
**while** $i < max\ iterations$ **do**
    $//$**Forward Propagation**
    **while** $l$ in $layers$ **do**
        **if** $i == update\_iter_{w_l}$ **then**
            $Diff = QEM(W_l)$
            $Itv$,$n_{w_l}$,$r_{w_l}$ =$QPA(W_l, Diff)$
            $update\_iter_{w_l} = i + Itv$
        **end if**
        $\hat{W}_l$=Quantifiy($W_l, n_{w_l}, r_{w_l}$)
        **if** $i == update\_iter_{x_l}$ **then**
            $Diff = QEM(X_l)$
            $Itv$,$n_{x_l}$,$r_{x_l}$ =$QPA(X_l, Diff)$
            $update\_iter_{x_l} = i + Itv$
        **end if**
        $\hat{X}_l$=Quantifiy($X_l, n_{x_l}, r_{x_l}$)
        Forward:$X_{l+1} = \hat{X}_l * \hat{W}_l$//FPROP
    **end while**
    $//$**Backward Propagation**
    **while** $l$ in $layers$ **do**
        **if** $i == update\_iter_{\Delta x_{l+1}}$ **then**
            $Diff = QEM(\Delta X_{l+1})$
            $Itv$,$n_{\Delta x_{l+1}}$,$r_{\Delta x_{l+1}}$ =$QPA(\Delta X_{l+1}, Diff)$
            $update\_iter_{\Delta x_{l+1}} = i + Itv$
        **end if**
        $\Delta\hat{X}_{l+1}$=Quantifiy($\Delta X_{l+1}, n_{\Delta x_{l+1}}, r_{\Delta x_{l+1}}$)
        Backward:$\Delta X_l = \Delta\hat{X}_{l+1} * \hat{W}_l^T$ //BPROP
        Backward: $\Delta W_l = \hat{X}_l^T * \Delta\hat{X}_{l+1}$ //WTGRAD
        $//$**Weight Update**
        $W_l = W_l + f(\Delta W_l)$
    **end while**
**end while**
```

4.1. Quantization Error Measurement

Based on the observation 1 and observation 3, we propose to adjust quantization parameters according to data distribution. The difference of mean before and after quantization is a good quantization error measurement, which indicates the change of data distribution and suggests the need for adjusting quantization resolution.

Intuitively, as shown in Figure. 4, the orange line and blue line represent two different data distributions. The quantization resolution is shown as the distance between a and b, and the data in between is quantified to a. Using certain quantization parameters (reflected by quantization resolution in figure), the distribution difference can be reflected by the difference of shadow areas. Specifically, the shadow area S1 is approximately equal to S2 for orange distribution, but S3 is much larger than S4 for blue distribution, which means far more amount of data is quantified to a under the blue distribution. Therefore, for blue one, the mean value after quantization $m_{\hat{d}}$ is much smaller than the original mean value m_d. As the distance between a and b reduces, the area difference between S3 and S4 will also be reduced. Therefore, the difference of mean value before and after quantization reflects the connection between quantization resolution and data distribution.

Mathematically, assuming that the data is under Gaussian distribution $P(x) \sim G(0, \sigma)$, and data $x \in R^p$ is quantified to $\hat{x}$. Considering the positive x, the mean between $[a, b]$ is $m_x = \frac{\int_a^b P(x)xdx}{\int_a^b P(x)dx}$, and after quantification the

[1] The quantification method is described in Appendix B

Authorized licensed use limited to: Cornell University Library. Downloaded on August 15,2020 at 14:00:39 UTC from IEEE Xplore. Restrictions apply.

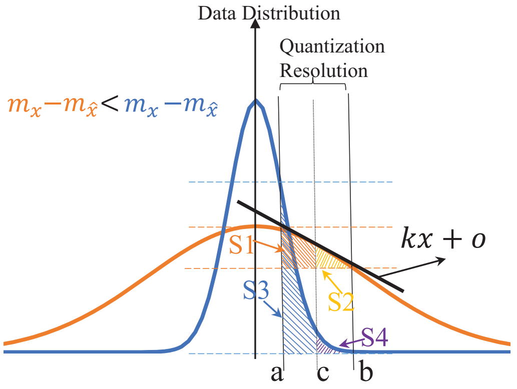


Figure 4: Data distribution and quantization resolution.

mean is $m_{\hat{x}} = \frac{a\int_a^c P(x)dx + b\int_c^b P(x)dx}{\int_a^b P(x)dx}$. The difference of mean before and after quantization is represented as $\frac{m_x}{m_{\hat{x}}} = \frac{\int_a^b P(x)xdx}{a\int_a^c P(x)dx + b\int_c^b P(x)dx}$. There is no elementary indefinite integral for Gaussian function, so we resort to numerical integration algorithms. We use $P(x) = kx + o$ to approximate the local value between $[a, b]$ with $b < -\frac{o}{k}, k < 0$, and assign $C = \frac{1}{4}k(a+b)^2 + \frac{o(a+b)}{2}$, then we have:

$$\frac{m_x}{m_{\hat{x}}} = 1 + \frac{1/24}{\frac{C}{(b-a)^2(-k)} - 1/8} \tag{1}$$

It is demonstrated that $\frac{m_x}{m_{\hat{x}}} > 1$ and $C > 0$ (see Appendix A for details), so we have $\frac{m_x}{m_{\hat{x}}} \propto (b-a)^2 * (-k)$. When decreasing $b - a$ or increasing k, the difference of mean will be reduced. Therefore, the difference of mean serves as an explicit indicator for adjusting quantization resolution (represented by $b-a$) according to data distribution (represented by $\sigma \propto (-k)$).

Equation. 2 is used in determining quantization parameters during training.

$$\begin{aligned} Diff &= log_2(|\frac{m_x - m_{\hat{x}}}{m_x}| + 1) \\ &= log_2(|\frac{\sum_i^p |x_i| - \sum_i^p |\hat{x_i}|}{\sum_i^p |x_i|}| + 1) \end{aligned} \tag{2}$$

$Diff$ is applied in training to adjust quantization resolution according to data distribution. Larger $Diff$ indicates the distribution has higher variance σ, so it is needed to decrease quantization resolution r.

4.2. Quantification Parameter Adjustment

According to observation 2, we propose to automatically determine the quantization parameter based on the data evolution. Under the circumstance of fixed-point representation, the quantization variables include data range, quantization resolution r and bit-width n. These three variables are inter-dependent, as $Range \approx r \times 2^n$. Therefore, we use only two of them as quantization parameters (i.e., r and n). The parameter adjustment process is triggered by insufficient quantization resolution and dramatic change of data range.

For insufficient quantization resolution, we use $Diff$ as indicator. When $Diff$ exceeds certain threshold T_{data}, the quantization resolution is reduced by increasing bit-width, as $n_{new} \leftarrow n_{old} + n'$,where $n' = 8$ is the bit-width growth step. We can either set the initial $n_{old} = 8$ and recursively adjust bit-width until proper n_{new} (denoted as Mode1), or we can set the initial n_{old} as the previous iteration's proper bit-width (denoted as Mode2). The quantization resolution is adjusted according to the new bit-width n as $r = 2^{ceil(log_2(\frac{Range}{2^{n-1}-1}))}$, where $Range$ is the max absolute value of data to be quantified.

For the change of data range, we propose another indicator R for iteration i as:

$$R_i = \alpha \times Range + (1 - \alpha) \times R_{i-1} \tag{3}$$

where R_i is the moving average of data $Range$ during several iterations.

The quantization parameter adjustment interval Itv is automatically determined by both $Diff$ and R. In initialization phase (one-tenth of the first epoch), Itv is set to 1. After initialization phase, the adjustment interval is $Itv = \frac{\beta}{max(I_1, I_2)} - \gamma$, as $I_1 = \delta \times Diff^2$ and $I_2 = |R_i - R_{i-1}|$. As shown in experiment, Itv increases during training. Within Itv iterations, the quantization parameters are kept the same, so there is no need to calculate $Diff$ and max absolute value of the data.

5. Experiment

We first evaluate the quantization error measurement, and show the computational complexity introduced by adaptive fixed-point. Then, we evaluate the proposed adaptive fixed-point training on a wide variety of deep learning tasks including image classification, object detection, segmentation and machine translation to demonstrate our observation and approach are widely applicable . At last, we show the training acceleration on existing hardware.

5.1. Evaluation of Error Measurement

We use Pearson correlation coefficient in Equation. 4 to show the correlation between network accuracy a and quantization error metric M.

$$R^2 = \frac{(\sum(M - \bar{M})(a - \bar{a}))^2}{\sum(M - \bar{M})^2 \sum(a - \bar{a})^2} \tag{4}$$

The evaluated quantization error metrics including the proposed $M1 = \frac{|\sum_i |x_i| - \sum_i |\hat{x_i}||}{\sum_i |x_i|}$ and several variants: $M2 =$

Authorized licensed use limited to: Cornell University Library. Downloaded on August 15,2020 at 14:00:39 UTC from IEEE Xplore. Restrictions apply.

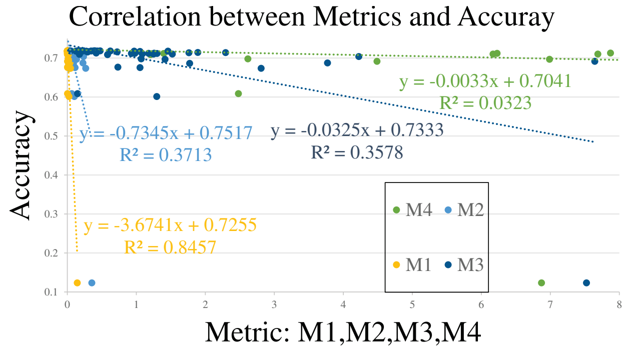


Figure 5: Correlation between MobileNet-v2 accuracy a and quantization error measurement M.

$\frac{\sum_i |x_i - \hat{x_i}|}{\sum_i |x_i|}$, $M3 = \sum_i \frac{|x_i - \hat{x_i}|}{|x_i|}$, $M4 = \sum_j P_j log(\frac{P_j}{Q_j})$. M2 is similar as in [27, 39]. M4 is the Kullback-Leibler divergence, with P_j and Q_j are the discrete probability distributions of original data and data after quantization. Specifically, we quantify each single layer of MobileNet-v2 and do the forward propagation to get the corresponding network accuracy. The quantization is done with different bit-width (i.e., 6, 8), so various degrees of quantization error and the corresponding network accuracy are generated.

Figure. 5 shows the linear correlation between network accuracy and several error metrics. Our proposed quantization error measurement M1 has the highest correlation score (0.84 for MobileNet and 0.85 for ResNet50 in Appendix D) with the network-level accuracy, which means the proposed error measurement can serve as a reasonable layer-wise accuracy indicator. MobileNet, as light-weight network, is relatively hard to quantified, so it can exhibit the most noticeable difference between different evaluation metrics M1, M2, M3 and M4.

5.2. Computational Complexity

We evaluate the extra computations introduced by adaptive fixed-point quantification. The extra computations refer to calculations in QEM, QPA and data quantification, denoted as **forward quantification** and **backward quantification** in Figure. 6 and Figure. 7. These two figures show the operation percentages and running time for different networks[2]. For light-weight network MobileNet, the extra computation is relatively large. For other networks, the extra computation is within 1%. The speed of our method is 1.7x to 2.8x of float32 speed on GPU.

During training, the adjustment frequency is calculated as $\frac{Adj\_Iter}{Iter}$, where Adj_Iter is the number of iteration that performs QEM and QPA, and $Iter$ is the total number of executed training iterations. As shown in Figure. 8a, at the initial epochs, the adjustment frequency is near 100%. As the training progresses, the adjustment frequency is dramatically decreasing to 0.1% at end.

Figure. 8b shows the percentage of activation gradients quantified to int8 during training on VGG16. Mode1 allows the bit-width both to increase and to decrease in the training, while Mode2 only allows the bit-width to increase. In Mode1, on average 39.6% layers are kept int8 (final top1 accuracy: 70.2%). In Mode2. on average 18.8% of layers are kept int8 (final top1 accuracy: 70.6%).

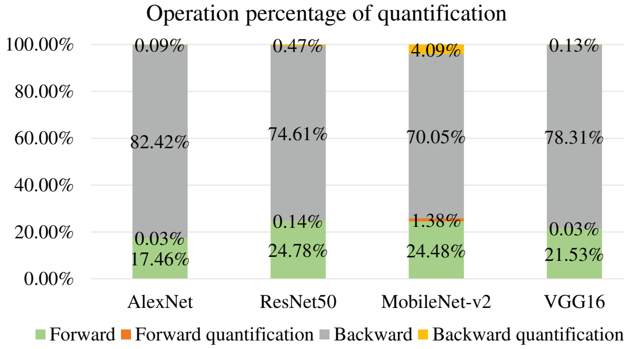


Figure 6: Operation percentage of forward and backward quantization for different models.

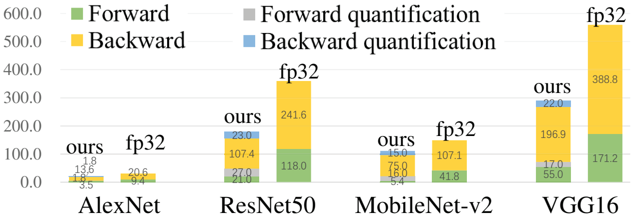
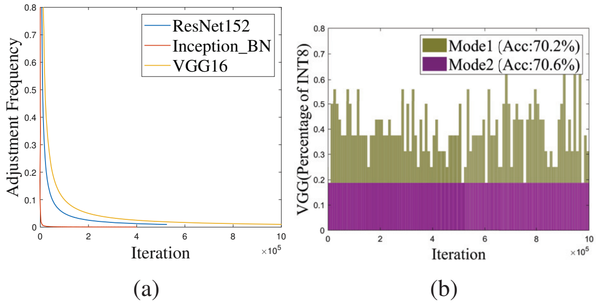


Figure 7: Time (ms) for one iteration on V100 GPU with128 batchsize. As V100 does not support int16, here float16 is used to approximate int16.



(a) (b)

Figure 8: (a)Quantification parameter adjustment frequency during training.(b)Int8 percentage during training.

[2] https://github.com/tensorflow/models/tree/master/research/slim

Authorized licensed use limited to: Cornell University Library. Downloaded on August 15,2020 at 14:00:39 UTC from IEEE Xplore. Restrictions apply.

Table 1: Classification, object detection and segmentation. For all the networks, 100% weights and and 100% activations are quantified to int8.

Classification	**float32**	**Adaptive**	**Activation Gradient**	
Network	**Acc**	**Acc**	int8	int16
AlexNet	58.0	58.22	22.5%	77.5%
VGG16	71.0	70.6	31.3%	68.7%
Inception_BN	73.0	72.8	4.5%	95.5%
ResNet50	76.4	76.2	0.8%	99.2%
ResNet152	78.8	78.2	1.7%	98.3%
MobileNet v2	71.8	70.5	0.7%	99.2%
SSD Detection	**float32**	**Adaptive**	**Activation Gradient**	
Network	**mAP**	**mAP**	int8	int16
COCO_VGG	43.1	42.4	31.4%	68.6%
VOC_VGG	77.3	77.2	34.3%	65.7%
IMG_Res101	44.1	44.4	28.6%	71.4%
Segmentation	**float32**	**Adaptive**	**Activation Gradient**	
Network	**meanIoU**	**meanIoU**	int8	int16
deeplab-v1	70.1	69.9	1.0%	99.0%

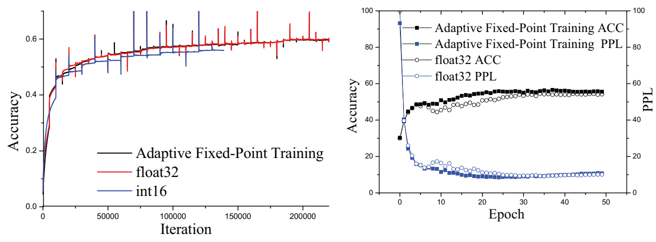
5.3. Accuracy Results

Our proposed Adaptive Fixed-Point Training approach uses **all the same** hyper-parameters (e.g., learning rate, max training iterations and *etc.*) as the original float32 training settings. For all the tasks, we fix the initial bit-width to int8 [3], with $\alpha = 0.04$, $\beta = 0.1$, $\delta = 100$, $\gamma = 2$, $T_{data} = 0.03$, and Mode2 is used in QPA.

5.3.1 Computer Vision

We train several convolution neural networks with ImageNet datasets using Tensorflow framework[4]. The networks include AlexNet [18], VGG [28], Inception_BN [29], ResNet [14] and MobileNet v2 [27][5]. We train SSD object detection networks [21][6] with VOC dataset [9], COCO dataset [20] and Imagnet Detection dataset (IMG) [25] upon two backbone networks VGG and ResNet101. We train deeplab [3][7] segmentation network on VOC dataset. For classification task, Top1 Accuracy (Acc) is used as evaluation metric. For object detection task, Mean Average Precision (mAP) is used as evaluation metric. For segmentation task, Mean Intersection over Union (meanIoU) is used as evaluation metric.

As shown in Table. 1, Adaptive Fixed-Point Training generates similar results as float32 baseline. The accuracy drop on MobileNet-v2 is consistent with the quantization results in Google's work (Acc:70.8) [16]. However, using our adaptive precision fixed-point training, int8 weights can be directly deployed and no further quantified fine-tuning is needed. The proposed QEM and QPA automatically change the bit-width used for different layers. During the whole training, the percentages of different bit-width in quantization are shown in Table. 1[8]. For backward activation gradient, int16 is needed for most layers and networks, but for some layers of AlexNet and SSD, int8 is enough.



(a) Sockeye (b) Transformer

Figure 9: Machine translation.

The use of same hyper-parameters in training indicates that Adaptive Fixed-Point Training has the same convergence speed as float32 training.(More loss curves are shown in Appendix F.)

5.3.2 Machine Translation

We train two widely used machine translation models from scratch with Adam optimizer. The first Sockeye [15] is a sequence-to-sequence RNN model implemented with MXNet [5][9], and trained on the WMT'17 news translation dataset (50k sentence pairs). The word vocabularies contain 50K entries for English and German. The second is Transformer [32][10], utilizing self-attention mechanism. This network is trained on the WMT'16 Multi30k dataset (3.9k sentence pairs). Word-level accuracy and perplexity (PPL) are used as evaluation metrics.

Training curve of Sockeye is shown in Figure. 9a. Adaptive Fixed-Point Training is compared with float32 baseline and an int16 method, which employs int16 to quantified all the layers of activation gradients without bit-width adaption. At the end of Adaptive Fixed-Point Training, 0.8% layers of activation gradients are quantified to int24, 10% layers are int8, and others are int16. As shown in Figure. 9a, the int16 method gradually results in 2% loss of accuracy, while our Adaptive Fixed-Point generates the same accuracy (62.05%) as float 32 baseline (61.97%). This comparison shows the proposed bit-width adaption is necessary to

[3] Appendix E shows results of adaptive lower bit-width (i.e., int4) for these forward-pass.

[4] https://github.com/tensorpack/tensorpack/tree/master/examples/

[5] https://github.com/tensorflow/models/tree/master/research/slim

[6] https://github.com/weiliu89/caffe/tree/ssd

[7] https://github.com/msracver/Deformable-ConvNets

[8] This is the results of Mode2, as Mode2 generates slightly better results than Mode1, as shown in Figure. 8(b).

[9] https://github.com/awslabs/sockeye

[10] https://github.com/jadore801120/attention-is-all-you-need-pytorch

Authorized licensed use limited to: Cornell University Library. Downloaded on August 15,2020 at 14:00:39 UTC from IEEE Xplore. Restrictions apply.

Table 2: Comparison of network quantization methods.

Methods	Backward Bit-width	Adaptive	Training	Accuracy Degradation	
Cited	(WTGRAD/BPROP)	Bit-width	from Scratch	CNN	RNN
[34]	float8, float16	no	yes	$<$ 1%(ResNet50)	n/a
[22]	float16	no	yes	$<$ 1%(ResNet50)	$<$ 1% (Translation)
[16]	float32	no	no	1.5% (ResNet50)	n/a
[17]	float32	no	no	$<$ 1%(ResNet18)	n/a
[6]	float32	no	yes	$<$ 1%(ResNet50)	n/a
[35]	float32	yes	no	$<$ 1%(ResNet18)	n/a
[39]	float32	yes	no	$<$ 1%(ResNet50)	n/a
[37]	float32	yes	yes	$<$ 1%(ResNet50)	n/a
[38]	int8, float32	no	yes	2.9%(AlexNet)	n/a
[36]	int8	no	yes	4%(AlexNet)	n/a
[1]	int16, float32	no	yes	$<$ 1%(ResNet50)	n/a
[7]	int16	no	yes	$<$ 1%(ResNet50)	2% (Translation)
Adaptive Fixed-Point	int8∼16 (CNN) int8∼24 (RNN)	yes	yes	$<$ 1%(ResNet50)	$<$ 1% (Translation)

guarantee training accuracy and reduces the total bit-width in computation.

The training convergence curve of Transformer is shown in Figure. 9b. We report the accuracy and PPL on validation set. Adaptive Fixed-Point (Acc: 55.54%) is slightly better than float32 (Acc: 54.13%). On average 2.28% of iterations trigger quantization parameter adjustment.

5.3.3 Comparison to Others

Table. 2 shows the comparison to other quantization work. We estimate the speeds of previous methods with a simple rule widely used in [38, 7]: Int16/float16 and int8/float8 operations are 2x and 4x faster than float32 operations respectively. Fig.10 compares the speeds and accuracies of quantization methods with low bit-width back propagation [34, 22, 38, 36, 1, 7] . It shows that our method remarkably outperforms the pareto front of previous methods. Regarding accuracy, our method has the best accuracy among all quantization methods, and is the only method having similar accuracies with float32 across all networks; regarding speed, our method is at least 21% faster than the most accurate quantization method [7](whose VGG16 accuracy is lower than ours for $>$ 2%).

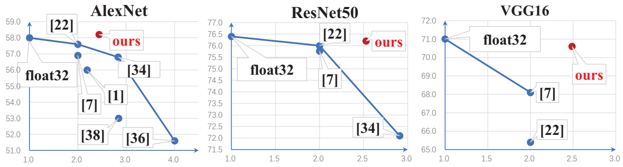


Figure 10: Accuracy (vertical axis) and estimated speed ratio against float32 (horizontal axis) on ImageNet.

.

Table 3: Layer-wise training speedup of AlexNet

	conv0	conv1	conv2	conv3	conv4
CPU Forward	2.03	3.89	6.2	4.44	4.28
CPU Backward	1.91	1.71	1.78	2.21	2.07
	fc0	fc1	fc2	Overall	
CPU Forward	4.09	6.42	4.41	3.98	
CPU Backward	4.41	4.97	2.03	2.07	

6. Training Acceleration

Intel Xeon Gold 6154 supports vector int8/int16 operations with AXV2 instruction set. Table. 3 shows the speedup of our method compared with float32 in training. Specifically, we use 100 iterations' average acceleration ratio of each layer in forward-pass and backward-pass for AlexNet with 256 batch size[11]. Our approach can achieve 2.52 times speedup over float32 training on CPU.

7. Conclusion and Future Work

With the novel training scheme presented in this paper, AI processors no longer need to equip expensive floating-point hardware units to support deep learning training tasks. Instead, lightweight fixed-point hardware units, which consumes much smaller chip area and power, have been sufficient for training. Therefore, given the same budget of chip area/power, training performance of an AI processor could be made much higher than before, which is definitely a good news for chip designers and users. Recent chip products of Cambricon, including MLU220, MLU270 and the upcoming MLU290, can efficiently support the adaptive fixed-point training scheme proposed in this paper.

[11] Xeon Gold 6154 can only support multiplication between equal bit-width fixed-point numbers, so in this experiment int16 × int8 is implemented as int16 × int16.

Authorized licensed use limited to: Cornell University Library. Downloaded on August 15,2020 at 14:00:39 UTC from IEEE Xplore. Restrictions apply.

References

[1] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. In *NeurIPS*, pages 5145–5153, 2018.

[2] Han Cai and Zhu. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.

[3] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 40(4):834–848, 2017.

[4] Shangyu Chen, Wenya Wang, and Sinno Jialin Pan. Deep neural network quantization via layer-wise optimization using limited training data. In *AAAI*, 2019.

[5] Tianqi Chen, Mu Li, and Yutian Li. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv:1512.01274*, 2015.

[6] Jungwook Choi and Zhuo Wang. Pact: Parameterized clipping activation for quantized neural networks. *arXiv:1805.06085*, 2018.

[7] Dipankar Das, Naveen Mellempudi, Mudigere, Avancha, et al. Mixed precision training of convolutional neural networks using integer operations. In *ICLR*, 2018.

[8] Jacob Devlin and Chang. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186, 2019.

[9] Mark Everingham and Luc Van Gool. The pascal visual object classes (voc) challenge. *IJCV*, 88(2):303–338, 2010.

[10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010.

[11] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *ICML*, pages 1737–1746, 2015.

[12] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

[13] Kaiming He and Xiangyu Zhang. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

[14] Kaiming He and Xiangyu Zhang. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[15] Felix Hieber and Tobias Domhan. Sockeye: A Toolkit for Neural Machine Translation. *arXiv:1712.05690*, Dec. 2017.

[16] Benoit Jacob and Skirmantas Kligys. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, pages 2704–2713, 2018.

[17] Sangil Jung and Changyong Son. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *CVPR*, pages 4350–4359, 2019.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1097–1105, 2012.

[19] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *ICML*, pages 2849–2858, 2016.

[20] Tsung-Yi Lin and Michael Maire. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755. Springer, 2014.

[21] Wei Liu and Dragomir Anguelov. Ssd: Single shot multibox detector. In *ICCV*, pages 21–37. Springer, 2016.

[22] Sharan Narang, Gregory Diamos, Elsen, et al. Mixed precision training. *ICLR*, 2018.

[23] NVIDIA. Nvidia tesla v100 gpu architecture. 2017.

[24] Andres Rodriguez, Eden Segal, et al. Lower numerical precision deep learning inference and training. *Intel White Paper*, 2018.

[25] Olga Russakovsky, Jia Deng, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015.

[26] Charbel Sakr and Naresh Shanbhag. Per-tensor fixed-point quantization of the back-propagation algorithm. In *ICLR*, 2019.

[27] Tao Sheng and Chen Feng. A quantization-friendly separable convolution for mobilenets. In *EMC2*, pages 14–18. IEEE, 2018.

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

[29] Christian Szegedy and Vincent Vanhoucke. Rethinking the inception architecture for computer vision. In *CVPR*, pages 2818–2826, 2016.

[30] Stefan Uhlich, Lukas Mauch, Kazuki Yoshiyama, Fabien Cardinaux, Javier Alonso García, Stephen Tiedemann, Thomas Kemp, and Akira Nakamura. Differentiable quantization of deep neural networks. *arXiv preprint arXiv:1905.11452*, 2019.

[31] Yaman Umuroglu, Lahiru Rasnayake, and Magnus Själander. Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing. In *FPL*, pages 307–3077. IEEE, 2018.

[32] Ashish Vaswani and Shazeer. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[33] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *CVPR*, pages 8612–8620, 2019.

[34] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *NeurIPS*, pages 7675–7684, 2018.

[35] Bichen Wu, Yanghan Wang, and Peizhao Zhang. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv:1812.00090*, 2018.

[36] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. In *ICLR*, 2018.

[37] Jiwei Yang and Xu Shen. Quantization networks. In *CVPR*, June 2019.

[38] Shuchang Zhou, Yuxin Wu, and Zekun Ni. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv:1606.06160*, 2016.

[39] Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard. Adaptive quantization for deep neural network. In *AAAI*, 2018.

Authorized licensed use limited to: Cornell University Library. Downloaded on August 15,2020 at 14:00:39 UTC from IEEE Xplore. Restrictions apply.