Symmetric Quantization Scheme and Approximate Batch Normalization for Quantized Neural Networks Training

Xiangyu Wu\textsuperscript{1*}, Zhisong Bie\textsuperscript{2}, Xuehong Lin\textsuperscript{3}

\textsuperscript{1,2,3}School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing, China.

*Corresponding author e-mail: wu_xiangyu@bupt.edu.cn

Abstract. With the advancements of circuit design and manufacturing processes of hardware, Deep Neural Networks (DNN) have been developed to be deeper and larger to achieve remarkable results in many aspects. Despite this, problems such as excessive memory usage and severe resource consumption are still unavoidable. We proposed a symmetric quantization scheme, which quantizes the weights from a certain kernel to be symmetric to zero along with a scaling factor and bias term in order to make full use of the whole quantization space and reduce the multiplication operations when convolution happens. We also came up with an Approximate Batch Normalization (ABN) method, which simplifies the traditional BN to linear operations lightly. Besides, activations and gradients are quantized in our scheme with fine distinctions, so that most floating-point calculations can be converted into fixed-point calculations and bit-shift operations in both training and inference phase. Our proposed scheme greatly alleviates the problems mentioned above theoretically with fewer computations and smaller storage theoretically. We carried out various experiments to prove its feasibility.

1. Introduction

In 2012, AlexNet [1] was proposed and got the first place in ImageNet Large Scale Visual Recognition Competition (ILSVRC) [2] with a great advantage, which showed that deep learning has great potential for visual tasks, and attracted more and more researchers to invest in this field. In recent years, deep learning has vigorously promoted the development in many domains, such as image classification, speech recognition, and natural language processing, etc., and is refreshing state-of-the-art results, even surpassing human performance. Despite this, obstacles like the huge memory requirements and high resource consumption faced by DNN during the training phase are still barriers to deep learning technology. Even in the testing phase, with tens or hundreds of megabytes of model size and hundreds of millions of floating-point Multiply-Accumulate (MAC) operations, it remains a problem on how to implement the inference on most small-scale intelligent devices.

In the past few years, there have been a lot of works devoted to reduce the size of DNN models or accelerate the inference, which can be roughly divided into the following branches. One is pruning, which compresses the models by cutting out the weights [3] or the filters [4] which have less impact on the performance of the models. The second approach is to introduce more compact models, such as SqueezeNet [5], MobileNet [6], etc. These networks are designed to have the concise structures to reduce the model scale or the computation costs. The third method is to utilize low-rank decomposition [7]. It takes the sparsity of the model parameters into account, and decompose the
models with low-rank matrixes, which reduces the redundancy of the models. Last but not least, the parameters quantization method, which quantizes the full-precision floating-point values into fixed-point values represented by low bits, is also an intuitively effective and important way to compress the model size and accelerate the networks.

In this paper, we proposed a novel quantization scheme that quantized the weights symmetric about zero filter-wise in low bits, concretely, 8-bit quantization is used considering the deployment of hardware. We claimed that it only calls for a half accumulator registers and multiplication operations when we implement QNN on a certain hardware compared with current scheme [8]. An approximate batch normalization is introduced to simplify the traditional BN operations by replacing the variance with the range of inputs, which is quite cheaper in computations while comparable accuracy can be obtained. To the best of our knowledge, we are the first one to come up with such a symmetrical quantization scheme and the approximate BN method in neural networks quantization field. Gradients are also quantized in back propagation process with the similar method proposed by DoReFa-Net [9], but more meticulous. We carried out various experiments with our scheme, and all the parts have been proved to be effective when tested separately or together in our experiments.

2. Related Work
Quantized Neural Networks (QNN) can be categorized into two main groups from the perspective of whether pre-trained models are used [10].

For the first scheme, the quantized neural networks are trained by fine-tuning pre-trained models. Generally, the fine-tuning method can make the networks converge in less iterations, and get very limited accuracy loss after quantization. The quantized models can be compressed to generate relatively small ones, thus solving the problem of oversized models in the inference phase. But in the pre-training stage, the traditional training methods are used with full-precision values, which does not solve the problem of huge memory occupancy and high resource consumption in the training phase.

For the second scheme, quantization happens directly in the training phase. Some of these methods deploy simple quantization strategies to convert network weights or activations into binary or ternary values, represented by Binary Weight Networks (BWN) [11] and Ternary Weights Networks (TWN) [12]. BWN quantizes the network into {-1, 1} according to their signs, while TWN introduces a threshold to quantize the network into{-1, 0, 1}. Because of the simplifying of the weights representation, the models are quantized to be very small, and the application of bit manipulation makes great improvements speeding up the forward propagation. In some networks with relatively large scale of parameters like AlexNet, the results are comparative to the benchmarks, but in some others like ResNet [13], server accuracy loss will occur. Besides, the difficulty of convergence in the training phase should not be ignored. There are also some methods to quantize the gradients in the back-propagation process. DoReFa-Net, advised to uses 1-bit, 2-bits and 4-bits to quantify network weights, activations and gradients respectively. Though good results are achieved, it employs lots of high complexity operations such as traditional BN and hyperbolic tangent, which make the overall operations still at a high level. Some other methods are proposed to derive the quantization strategy as a convex optimization problem. Back Propagation (BP) method [14], Newton method [15] and ADMM method [16] are applied to solve the problem. Although they are mathematical logical and have beautiful close forms, they also bring enormous extra computation costs, hinder such methods to be implemented on common hardware.

3. Quantization Scheme

3.1. Symmetrical Weights Quantization
For n-bit (n≥2) fixed-point weights quantization, weights are usually mapped into two types of sets about whether uniform quantization interval is adopted. In our work, we take the equidistant quantization form which is easier for weights quantization. Different with the scheme which maps the quantized values to [0, 2^n − 1] [17], our proposed symmetric method maps them to [−2^{n−1} + 1, 2^{n−1} − 1]. This idea was came up for the reason that it only calls for a half accumulator registers
when implemented on certain hardware like FPGA. This is one of the motivation of our work, and will be described in detailed in section 3.1.

Given a certain convolution kernel $W_{H \times W \times C}$, $H, W, C$ is its 3-dimension size, and $w_{H \times W \times C}$ is the full-precision weight, where $h \in H, w \in W, c \in C$, our weights quantization rules are detailed as follow:

$$s_w = w_{\max} - w_{\min} ; \#(1)$$
$$b = \frac{w_{\max} + w_{\min}}{2} ; \#(2)$$
$$\hat{w} = clamp(round\left(\frac{w - b}{s_w} \times 2^n\right), -2^{n-1} + 1, 2^{n-1} - 1). \#(3)$$

Here, $w_{\max}$ and $w_{\min}$ is the maximum and minimum value of weights of $W_{H \times W \times C}$, $s_w$ is the range between $w_{\max}$ and $w_{\min}$, $b$ is the bias term, representing the gap between the center of $W_{H \times W \times C}$ and zero. (3) contains the quantization and clamp operations, after all, the quantized $\hat{w}$ is ranging in $(-127, 127)$ together with a scaling factor and a bias. Such method makes the maximum use of the quantization space, allowing much value precision to take effect in QNN. Note that when training is undergoing, weights should restore to its original form by reverse operations as follow:

$$W_{H \times W \times C} = s_w \times \hat{W}_{H \times W \times C} \times 2^n + b \times I_{H \times W \times C} . \#(4)$$

In (4), $I_{H \times W \times C}$ is a kernel with the same size of $W$ consisting of “1”. (4) is the final form of quantized $W_{H \times W \times C}$. In this way, a certain convolution kernel $W_{H \times W \times C}$ can be represented by $H \times W \times C$ int-8 values and two 32-bit fixed-point values, which means nearly a quarter storage space is acquired compared with the original ones.

Quantization for weights of full connected (FC) layers is with the same method.

### 3.2. Activations Quantization with Approximate Batch Normalization

Batch normalization [18] has been proved to be effective for DNN training on many experiments. When back to why BN works, the author explained with solving the problem of “internal covariate shift”, whose essence is to make the inputs from different layers obey the similar distribution, which makes sense statistically to enable convergence to be more stable.

Given a layer with $N$-batch and $D$-dimensional inputs, written as $A = (A_1, A_2, ..., A_D)$, and for $d \in \{1, ..., D\}$, $A_d$ is consisted of $N$ samples. For traditional BN, its form written as:

$$\bar{A}_d = \alpha \cdot \frac{A_d - \mu_d}{\sqrt{\text{Var}(A_d)}} + \beta . \#(5)$$

Here, $\mu_d$ is the mean value of $A_d$, $\text{Var}(A_d)$ is the variance of $A_d$. $\alpha$ and $\beta$ are scale and shift parameters learnable. The computations of (5) are quite complex and a bit tricky meanwhile for the reason that in fact, distributions with same expectation and variance are not same distributions. Since original BN is not rigorous statistically, we searched for a new representation for BN operations.

In our substitution tests, we found that when replacing the variance term with a combination of simple range together with batch size would take effect as well. The expression of approximate BN is as follow:

$$\overline{A}_d = \alpha \cdot c * \sqrt{\log(N)} \cdot \frac{A_d - \mu_d}{R(A_d)} + \beta . \#(6)$$

Here, $R(A_d)$ refers to the range between the maximum and minimum values of $A_d$, $c$ is a constant coefficient which is not fixed mathematically. Actually, (6) is not a rigorous close form, the results may change slightly when the coefficient changes, and in our experiments, we found the results relatively good when $c$ was set to “2”. We called this approximate batch normalization method ABN in this paper.

After ABN, activations quantization is processed, we simply fused the ReLU activation function into the activations quantization process by quantizing negative values to zero directly. Activations quantization operations are as follow:

$$s_a = \bar{a}_{\max} = 2^{\left\lfloor \log_2 a_{\max} \right\rfloor} ; \#(7)$$
\[ \hat{a} = \begin{cases} \text{clamp} \left( \text{round} \left( \frac{a}{s_a} \times 2^n \right), 0, 255 \right), & a \leq 0; \\ 0, & a > 0. \end{cases} \]  
\[ A := \frac{s_a \times \hat{A}}{2^n}. \]  
(8)  
(9)

Here, \( \hat{a} \) is the quantized activation. The upper bound is set in (7) for two aspects of consideration: Firstly, use powers-of-2 values as denominator is for the easy computations of (8) when inference. Secondly, according to central limit theorem, activations contribute is approximate Gaussian contribution, large activations are rare and will produce large gradients to the update of weights, leaving them inactivated, so the floor function is introduced as saturation function to avoid this problem.

### 3.3. Scalable Gradients Quantization

Tough many quantization schemes have been proposed for weights and activations quantization, works on gradients quantization are relatively fewer. The main issue is that gradients vary greatly during the back propagation procedure and there is not a certain bound for them. Besides, the precision loss of gradients may influence the convergence process and harm the final performance severely. In our work, we shared the idea proposed in DoReFa-Net, which use a scaling factor to represent the quantized weights. The gradients quantization is demonstrated as follow:

\[ s_{g_l} = \max(\text{abs}(g_l)); \]  
(10)  
\[ \hat{g} = s_{g_l} \times \text{round} \left( \frac{g}{s_{g_l}} \times 2^{n-1} \right)/2^{n-1}. \]  
(11)

Here, \( g_l \) is the back-propagated gradient for the \( l \)th layer, and \( s_{g_l} \) is the scaling factor, referring to the maximum absolute value of \( g_l \) for a certain issue in a batch. As many previous works adopted, we use stochastic round instead of deterministic round. Clamp are also introduced, though not described here.

### 4. Training for Quantized Neural Networks

#### 4.1. Convolution Operation Simplification

Given \( A_{H,W,C}^l \) is the outputs of the \( l-1 \)th layer, where \( l \in \{2,3,...,L\} \), \( L \) is the number of layers for a certain neural network. Since activation function is fused into the activations quantization, \( A_{H,W,C}^l \) can be taken as the inputs of \( l \)th layer meanwhile. Given \( W_{H,W,C}^l \) is the corresponding convolution kernel of the \( l \)th layer, the convolution operation between them can be depicted as:

\[ O_d^l = W_{H,W,C}^l \odot A_{H,W,C}^l. \]  
(12)

Here, \( \odot \) indicates the kernel convolution operation. When proposed quantization is used, this procedure is rewritten as:

\[ O_d^l := \left( \frac{S_w \times W_{H,W,C}^l}{2^n} + b \times I_{H,W,C}^l \right) \odot \left( \frac{S_a \times \hat{A}_{H,W,C}^l}{2^n} \right) \]  
\[ = \frac{s_w \times s_a \times (W_{H,W,C}^l \odot \hat{A}_{H,W,C}^l)}{2^n} + b \times s_a \times (I_{H,W,C}^l \odot \hat{A}_{H,W,C}^l). \]  
(13)

Now we turn to describe \( W_{H,W,C}^l \odot \hat{A}_{H,W,C}^l \) in detailed. This process carries on multiplication and addition operations for \( H \times W \times C \) times respectively. Since the weights and activations are quantized into 8-bit, we can simply reduce the multiplication operations to \( 2^n \) with accumulators, and this is the widely-shared idea if the weights are quantized into uint-8 values. What’s different with symmetric quantization scheme is that, we quantized the weights to int-8 values, which means, the number of accumulator registers and multiplication operations can be further reduced to \( 2^{n-1} \), by rational use of the symmetric relation. When activations multiply with the weights with same absolute value, we may put them into one accumulator, switching the addition operations to the subtraction operations when the corresponding weights are originally negative. So it can be written as:
\[ W_{H=W+C}^T \otimes A_{H=W+C}^T = \sum_{i=0}^{127} \sum_{heH} \sum_{weW} \sum_{c=0}^{ceC} \left( \pm a_{h,w,c}^i \right) \left( w_{h,w,c}^i = \pm i \right). \] (#14)

For the later convolution operation in (13), \( B_{H=W+C}^T \otimes A_{H=W+C}^T \) can be simply achieved with one-time multiplication as follows, which is also computational-saving as follow:

\[ B_{H=W+C}^T \otimes A_{H=W+C}^T = b \sum_{heH} \sum_{weW} \sum_{c=0}^{ceC} \tilde{a}_{h,w,c}^T. \] (#15)

### 4.2. Gradients Propagation

In our scheme, weights updating process will be conducted with full precision. It is to say, only the gradients propagating over the neural networks will be quantized. This limitation is based on the fact that small gradients at each iteration should not be ignored.

In order to deal with the problem that the derivatives will be zero everywhere when then variables are discrete, as many previous works have adopted, we use the Straight-Through Estimator (STE) [19] method for gradients computation which can be summarized as:

\[ \frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial A}. \] (#16)

Here, \( \mathcal{L} \) refers to the loss function from the forward propagation. STE method sets the incoming gradients to a threshold function equal to its outgoing gradients. So the gradients propagated to the \( l \)th layer can be computed recursively from the last layer by:

\[ g_l = \frac{\partial \mathcal{L}}{\partial A_l}. \] (#17)

In (17), gradients are quantized as section 2.3 refers. We use full precision for weights update as common neural networks do as follows:

\[ g_{wl} = g_l \ast A_l^T, \] (#18)

\[ w_{l+1} = w_l - \eta \ast g_{wl}. \] (#19)

Such idea was came up except for the reason has been mentioned at the beginning of this section, from another view of point, it is also feasible because gradients is quantized for quick propagation, and only \( g_l \) is needed in this procedure. Whilst, \( g_{wl} \) is just for weights updating, which does not participate in the procedure of back propagation, full precision will not limit the speed or consume much resource.

### 5. Experiment Results

Our experiments are carried out from two aspects. On the one hand, we verified the effeteness of our approximate batch normalization upon BN-AlexNet and ResNet-18 with ImageNet; on the other hand, we tested our quantized scheme upon SVHN, CIFAR-10 and ImageNet with several neural networks. Both of them are described as follow.

#### 5.1. Experiments on Approximate Batch Normalization

Approximate BN is introduced in Section 2.2, in order to prove its effectiveness, we performed controlled experiments between approximate BN and traditional BN upon BN-AlexNet and ResNet-18 on ImageNet database using tensorpack [20] API.

![Figure 1](image_url) Results of controlled experiments results of ImageNet database.
Full precision is used in this experiment and the configure parameters are simply followed the tensorpack settings, we make no change except for modifying the batch normalization function. For BN-AlexNet, 20.50% top-5 error rate and 42.33% top-1 error rate are reached with traditional BN, while the corresponding results with approximate BN are 20.90% and 42.79%. The results are comparable and both greatly surpass the benchmark without BN of a top-1 error worse than 45%. For ResNet-18, we got top-1 error rate with 29.38% and 29.53%, and top-5 error rate with 10.33% and 11.71%, the losses from ABN are also acceptable to an extent. The detailed results with validation dataset are shown as follow:

5.2. Experiments on Quantized Neural Network
We further carried on serval experiment for the proposed quantization scheme. For SVHN dataset, which is relatively small, as can be seen form figure 2(a), the final results are almost the same among several configures. It turned out that our quantization scheme worked well for simple classification tasks. We then tested this scheme in CIFAR-10 dataset with ResNet-18, the results are shown in figure 2(b). The accuracy of the networks kept on about 93% with different training schemes, and our proposed scheme seemed to perform even better than the benchmark, though the increment was quite limited. Which is not shown here is that the error rate of training data was decrease to below $1 \times 10^{-4}$ in our experiments, we think it was another evident to prove our proposed scheme.

(a) Validation Error Rate with BN-AlexNet  (b) Validation Error Rate with ResNet-18

Figure 2. Results of controlled experiments results of ImageNet database.

Since the SVHN dataset and CIFAR-10 dataset may be thought too small to verify our scheme, the results on Image-Net with AlexNet are shown in figure 3(a) and figure 3(b) with 43.58% top-1 validation error and 21.46% top-5 validation error respectively, which are both close to the benchmark already shown in section 4.1. Besides, we can conclude from figure 2 and figure 3 that the convergence process is stable with the proposed scheme.

We will test the performance for resource occupation and energy costs on FPGA in further work to verify our scheme more comprehensive.

(a) Top-1 Error Rate of ImageNet  (b) Top-5 Error Rate of ImageNet

Figure 3. Results of ImageNet dataset with AlexNet.

6. Conclusion
In this paper, we proposed a symmetric quantization scheme, which enables the quantized weights to make full use of the whole quantization space and cuts down the accumulator registers and multiplication computations to a half compared with current scheme. We also proposed the scheme of
approximate batch normalization, which simplifies the complex square computations to be substituted by a linear range. Activations and gradients are also quantized. We described the training process with forward and backward propagation in detailed and performed several experiments of our scheme, and the results on SVHN, Cifar-10 and ImageNet with different models showed that our scheme can achieve competitive accuracy with the full precision models and state-of-the-art quantization methods, and is suitable for various neural networks. For the reason that most of the float-point computations can be replaced by fixed-point computations with low bits, the proposed scheme is quite hardware-friendly, which alleviates the memory and resource problem to a certain extent. Future work will be carried out on actual hardware, aiming to design and implement a neural network work platform.

References
[1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.
[2] Deng, Jia, et al. "Imagenet large scale visual recognition competition 2012 (ILSVRC2012)." See net.org/challenges/LSVRC (2012).
[3] Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149 (2015).
[4] Jaderberg, Max, Andrea Vedaldi, and Andrew Zisserman. "Speeding up convolutional neural networks with low rank expansions." arXiv preprint arXiv:1405.3866 (2014).
[5] Iandola, Forrest N., et al. "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size." arXiv preprint arXiv:1602.07360 (2016).
[6] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).
[7] Sainath, Tara N., et al. "Low-rank matrix factorization for deep neural network training with high-dimensional output targets." Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.
[8] Jacob, Benoit. "gennmlowp: a small self-contained low-precision GEMM library.(2017)." (2017).
[9] Zhou, Shuchang, et al. "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients." arXiv preprint arXiv:1606.06160 (2016).
[10] Cheng, Yu, et al. "A survey of model compression and acceleration for deep neural networks." arXiv preprint arXiv:1710.09282 (2017).
[11] Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In European Conference on Computer Vision (pp. 525-542). Springer, Cham. 2016.
[12] Li, F., Zhang, B., & Liu, B. Ternary weight networks. arXiv preprint arXiv:1605.04711 (2016).
[13] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
[14] Zhu, Chenzhuo, et al. "Trained ternary quantization." arXiv preprint arXiv:1612.01064 (2016).
[15] Hou, Lu, and James T. Kwok. "Loss-aware Weight Quantization of Deep Networks." arXiv preprint arXiv:1802.08635 (2018).
[16] Leng, Cong, et al. "Extremely low bit neural network: Squeeze the last bit out with admm." arXiv preprint arXiv:1707.09870(2017).
[17] Wu, Yonghui, et al. "Google's neural machine translation system: Bridging the gap between human and machine translation." arXiv preprint arXiv:1609.08144 (2016).
[18] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).
[19] Bengio, Yoshua, Nicholas Léonard, and Aaron Courville. "Estimating or propagating gradients through stochastic neurons for conditional computation." arXiv preprint arXiv:1308.3432 (2013).
[20] Wu, Y., et al.: Tensorpack. https://github.com/tensorpack/ (2016)