Multi-Precision Quantized Neural Networks via Encoding Decomposition of {-1,+1}

Qigong Sun, Fanhua Shang, Kang Yang, Xiufang Li, Yan Ren, Licheng Jiao
Key Laboratory of Intelligent Perception and Image Understanding of Ministry of Education,
International Research Center for Intelligent Perception and Computation,
Joint International Research Laboratory of Intelligent Perception and Computation,
School of Artificial Intelligence, Xidian University, Xi'an, Shaanxi Province, 710071, China

Abstract

The training of deep neural networks (DNNs) requires intensive resources both for computation and for storage performance. Thus, DNNs cannot be efficiently applied to mobile phones and embedded devices, which seriously limits their applicability in industry applications. To address this issue, we propose a novel encoding scheme of using $\{-1, +1\}$ to decompose quantized neural networks (QNNs) into multi-branch binary networks, which can be efficiently implemented by bitwise operations (*xnor* and *bitcount*) to achieve model compression, computational acceleration and resource saving. Based on our method, users can easily achieve different encoding precisions arbitrarily according to their requirements and hardware resources. The proposed mechanism is very suitable for the use of FPGA and ASIC in terms of data storage and computation, which provides a feasible idea for smart chips. We validate the effectiveness of our method on both large-scale image classification tasks (e.g., ImageNet) and object detection tasks. In particular, our method with low-bit encoding can still achieve almost the same performance as its full-precision counterparts.

Introduction

Deep Neural Networks (DNNs) have been successfully applied in many fields, especially in image classification, object detection and natural language processing. Because of numerous parameters and complex model architectures, huge storage space and considerable power consumption are needed. Furthermore, with the rapid development of chip technology, especially GPU and TPU, the computing power has been greatly improved. In the rapid developing era of deep learning, researchers use multiple GPUs or computer clusters to contribute to the exploration of complex problems. Nevertheless, the energy consumption and limitation of computing resources are still significant factors in industrial applications, which are generally ignored in scientific research. In other words, breathtaking results of many DNNs algorithms under the condition of applying GPUs lag behind the demand of industry. DNNs can hardly be applied in mobile phones and embedded devices (as typical industrial applications) directly due to their limited memory and calculation resources. Therefore, the compression and acceleration of networks are especially important in future development and commercial applications.

In recent years, many solutions have been proposed to improve the energy efficiency of hardware, achieve model compression or computational acceleration, such as network sparse and pruning (Hassibi and Stork 1993; Wen et al. 2016; Han et al. 2015), low-rank approximation (Denton et al. 2014; Jaderberg, Vedaldi, and Zisserman 2014; Tai et al. 2015), architecture design (Howard et al. 2017; Sandler et al. 2018; Luo et al. 2018), model quantization (Hubara et al. 2017; Rastegari et al. 2016; Lin, Zhao, and Pan 2017), and so on. Network sparse and pruning can dramatically reduce the redundant connections, and thus reduce the computational load in the inference process without large accuracy drop. Tai et al. (2015) used low-rank tensor decomposition to remove the redundancy in the kernels which can be as a generic tool for speeding up. Since there is some redundant information in the networks, the most direct approach of cutting down those information is to optimize the structure and yield small networks (Ioffe and Szegedy 2015; Iandola et al. 2016). For example, Howard et al. (2017) proposed to use bitwise separable convolutions to build light networks for mobile applications. Most of those networks still utilize floating-point number representations (i.e., full-precision values). However, Gupta et al. (2015) discussed that the representation of the full-precision weights and activations in networks is not necessary during the training of DNNs, and a nearly identical or slightly better accuracy rate may be obtained under lower-precision representation and calculation.

Since non-differentiable discrete functions are applied in QNNs generally, there obviously exists the gradient mismatch problem in training process. Therefore, the backpropagation algorithm cannot be directly used to train QNNs. Many scholars (Mishra and Marr 2017; Polino, Pascanu, and Alistarh 2018; Tang, Hua, and Wang 2017; Wang et al. 2018) are devoted themselves to improving the performance (e.g., accuracy and compression ratio) of QNNs, but few researchers have studied their acceleration, which is an important reason for hindering industrial promotion. To the best of our knowledge, the accelerated method used in binarized

neural networks (BNNs) (Hubara et al. 2016) is the most efficient strategy at present. This strategy uses bitwise operations (*xnor* and *bitcount*) to replace full-precision matrix multiplication, and results 58× faster and 32× memory saving in CPU (Rastegari et al. 2016). As discussed in (Liang et al. 2017), it has a higher acceleration ratio on FPGA, which can speed up to about 705× in the peak condition compared with CPU and is 70× faster than GPU. In particular, they quantized activation values and weights to bits and used bitwise logic operations to achieve extreme acceleration ratio in inference process, but they could suffer significant performance degradation. However, most models were proposed for a fixed precision, and cannot extend to other precision models. They may easily fall into local optimal solutions and suffer from slow convergence speed in training process.

The representation capability of binary parameters is insufficient for many practical applications, especially for large-scale image classification (e.g., ImageNet) and regression tasks. In order to address various complex problems and take full advantage of bitwise operations, Lin, Zhao, and Pan (2017) used the linear combination of multiple binary parameters {-1, +1} to approximate full-precision weights and activations. Therefore, the complex full-precision matrix multiplication can be decomposed into some simpler operations. This is the first time to use binary networks for image classification on ImageNet. Guo et al. (2017) and Xu et al. (2018) used the same technique to accelerate the training of CNNs and RNNs. In addition, those methods not only increase the number of parameters many times, but also introduce a scale factor to transform the original problem into an NP-hard problem, which naturally makes the solution difficult and high complexity.

In order to bridge the gap between low-bit and full-precision and apply to many cases, we propose a novel encoding scheme of using $\{-1, +1\}$ to easily decompose trained QNNs into multi-branch binary networks. Therefore, the inference process can be efficiently implemented by bitwise operations (*xnor* and *bitcount*) to achieve model compression, computational acceleration and resource saving. Thus, our encoding mechanism can improve the utilization of hardware resources, and achieve parameter compression and computation acceleration. In our experiments, we not only validate the performance of our method for image classification on CIFAR-10 and large-scale datasets, e.g., ImageNet, but also implement object detection tasks. The advantages of our method are summarized as follows:

- We can directly use the high-bit model parameters to initialize a low-bit model for faster training. Hence, our networks can be trained in a short time, and only dozens of times fine-tuning are needed to achieve the accuracies in our experiments. Of course, we can get better performance if we continue training the network. Thus, our multi-precision quantized networks can be easily popularized and applied to engineering practices.
- We propose a range of functions (called MBitEncoder) to decompose activations (for example, we can use M functions to get the state $\{-1, +1\}$ of M encoded bits), which are used for inference computation. Therefore, those decomposed bits can be directly used in network computation without other judgments and mapping calculations.
- After the process of decomposition, instead of storing all encoding bits in data types, e.g., char, int, float or double, the parameters can be individually stored by bit vectors. Thus, the smallest unit of data in electronic equipments can be reduced to 1-bit from 8-bit, 16-bit, 32-bit or 64-bit, which raised the utilization rate of resources and compression ratio of the model. Then the data can be encoded, calculated and stored in various encoding precisions.

Related Work

QNNs can effectively implement model compression, even to 32× memory saving. Many researchers are focusing on the following three classes of methods: quantification methods, the methods of optimization in training process and acceleration computation in inference process.

Quantification methods play a significant role in QNNs, and determine the state and distribution of weights and activation values. Gupta et al. (2015) used the notation of integer and fractional to denote a 16-bit fixed-point representation, and proposed a stochastic rounding method to quantify values. Vanhoucke, Senior, and Mao (2011) used 8-bit quantization to convert weights into signed char and activation values into unsigned char, and all the values are integer. For multi-state quantification (8-bit to 2-bit), linear quantization is usually used in (Hubara et al. 2017; Wang et al. 2018; Zhuang et al. 2018). Besides, Miyashita, Lee, and Murmann (2016) proposed logarithmic quantization to represent data and used bitshift operation in log-domain to compute dot products. For ternary weight networks (Li, Zhang, and Liu 2017), the weights are quantized to $\{-\Delta^*, 0, +\Delta^*\}$, where $\Delta^* = 0.7 \cdot E(|W|)$. In (Zhu et al. 2017), the positive and negative states are trained together with other parameters. When the states are constrained to 1-bit, Hubara et al. (2016) applied the sign function to binarize weights and activation values {-1, +1}. In (Rastegari et al. 2016), the authors also used $\{-\alpha^*, +\alpha^*\}$ to represent the binary states, where $\alpha^* = \frac{1}{n}\|W\|_{l_1}$.

It is obvious that discrete functions, which are non-differentiable or have zero derivatives everywhere, need to quantize weights or activation values. The traditional gradient descent method is unsuitable for the training of deep networks. Recently, there are many researchers devoting themselves to addressing this issue. Li et al. (2017) divided optimization methods into two categories: quantizing pre-trained models with or without retraining (Lin, Talathi, and Annapureddy 2016; Zhou et al. 2017; Li, Zhang, and Liu 2017; Lin, Zhao, and Pan 2017) and directly training quantized networks (Courbariaux, Bengio, and David 2015; Hubara et al. 2016; Rastegari et al. 2016; Wang et al. 2018). (Hubara et al. 2016; 2017) used the straight-through estimator (STE) in (Bengio, Léonard, and Courville 2013) to train networks. STE uses the nonzero gradient to approximate the function gradient, which is not-differentiable or whose derivative is zero, and then applies the stochastic gradient descent (SGD) to update the parameters. Mishra and Marr; Polino, Pascanu, and Alistarh (2017; 2018) applied

knowledge distillation techniques, which use high-precision teacher network to guide low-precision student network to improve network performance. In addition, some networks as in (Guo et al. 2017; Lin, Zhao, and Pan 2017) use the linear combination of binary values to approximate the full-precision weights and activation values. They not only increase the number of parameters many times, but also introduce the scale factor to transform the original problem into an NP-hard problem, which naturally makes the solution difficult and high complexity. Xu et al. (2018) used the two valued search tree to optimize the scale factor and achieved better performance in the language model by using the quantized recurrent neural networks.

After quantizing, weights or activation values are represented in a low-bit form, which has the potential of acceleration computation and memory saving. Because the hardware implementation has a certain threshold, many scholars have avoided considering their engineering acceleration. This is also an important reason for hindering industrial promotion. The most direct quantization is to convert floating-point parameters into their fixed-point (e.g., 16-bit, 8-bit), which can achieve hardware acceleration for fixed-point based computation (Gupta et al. 2015; Vanhoucke, Senior, and Mao 2011). When the weight is extremely quantized to the binary weight {-1, +1} as in (Courbariaux, Bengio, and David 2015) or ternary weight {-1, 0, +1} as in (Li, Zhang, and Liu 2017), the matrix multiplication can be transformed into full-precision matrix addition and subtraction to accelerate computation. Especially when the weight and activation values are binarized, matrix multiplication operations can be transformed into highly efficient logical and bitcounting operations (Hubara et al. 2016; Rastegari et al. 2016). Guo et al.; Lin, Zhao, and Pan (2017; 2017) used a series of linear combinations of {-1, +1} to approach the parameters of full-precision convolution model, and then converted floating point operations into multiple binary weight operations to achieve model compression and computation acceleration.

Multi-Precision Quantized Neural Networks

In this section, we use the multiplication of two vectors to introduce the novel encoding scheme of using $\{-1, +1\}$ to decompose QNNs into multi-branch binary networks. In each branch binary network, we use -1 and +1 as the basic elements to efficiently achieve model compression and forward inference acceleration for QNNs. Different from fixed-precision neural networks (e.g., binary, ternary), our method can yield multi-precision networks and make full use of the advantage of bitwise operations to accelerate QNNs.

Model Decomposition

As the basic computation in most neural network layers, matrix multiplication costs lots of resources and also is the most time consuming operation. Modern computers store and process data in binary format, thus non-negative integers can be directly encoded by $\{0, 1\}$. We propose a novel decomposition method to accelerate matrix multiplication as follows: Let $x = [\mathrm{x}^1, \mathrm{x}^2, ..., \mathrm{x}^N]^T$ and $w = [\mathrm{w}^1, \mathrm{w}^2, ..., \mathrm{w}^N]^T$ be two vectors of non-negative integers, where $\mathrm{x}^i, \mathrm{w}^i \in \{0, 1, 2, ...\}$ for $i = 1, 2, ..., N$. The dot product of those two vectors can be represented as follows:

$$x^T \cdot w = [\mathrm{x}^1, \mathrm{x}^2, ..., \mathrm{x}^N][\mathrm{w}^1, \mathrm{w}^2, ..., \mathrm{w}^N]^T \tag{1}$$

$$= \sum_{n=1}^{N} \mathrm{x}^n \cdot \mathrm{w}^n. \tag{2}$$

All of the above operations consist of N multiplications and $(N-1)$ additions. Based on the above $\{0, 1\}$ encoding scheme, the vector x can be encoded to binary form using M bits, i.e.,

$$x = [\overbrace{\mathrm{c}_M^1 \mathrm{c}_{M-1}^1 ... \mathrm{c}_1^1}, \overbrace{\mathrm{c}_M^2 \mathrm{c}_{M-1}^2 ... \mathrm{c}_1^2}, ..., \overbrace{\mathrm{c}_M^N \mathrm{c}_{M-1}^N ... \mathrm{c}_1^N}]^T. \tag{3}$$

Then the right-hand side of (3) can be converted into the following form:

$$\begin{bmatrix} \mathrm{c}_M^1 & \mathrm{c}_M^2 & \cdots & \mathrm{c}_M^N \\ \mathrm{c}_{M-1}^1 & \mathrm{c}_{M-1}^2 & \cdots & \mathrm{c}_{M-1}^N \\ \vdots & \vdots & \cdots & \vdots \\ \mathrm{c}_1^1 & \mathrm{c}_1^2 & \cdots & \mathrm{c}_1^N \end{bmatrix} = \begin{bmatrix} c_M \\ c_{M-1} \\ \vdots \\ c_1 \end{bmatrix}, \tag{4}$$

where

$$\mathrm{x}^j = \sum_{m=1}^{M} 2^{m-1} \cdot \mathrm{c}_m^j, \quad \mathrm{c}_m^j \in \{0, 1\}, \tag{5}$$

$$c_i = [\mathrm{c}_i^1, \mathrm{c}_i^2, ..., \mathrm{c}_i^N]. \tag{6}$$

In such an encoding scheme, the number of represented states is not greater than 2^M. In addition, we encode another vector w with K-bit numbers in the same way. Therefore, the dot product of the two vectors can be computed as follows:

$$x^T \cdot w = \sum_{n=1}^{N} \mathrm{x}^n \cdot \mathrm{w}^n \tag{7}$$

$$= \sum_{n=1}^{N} \left(\sum_{m=1}^{M} 2^{m-1} \cdot \mathrm{c}_m^n \right) \cdot \left(\sum_{k=1}^{K} 2^{k-1} \cdot \mathrm{d}_k^n \right) \tag{8}$$

$$= \sum_{m=1}^{M} \sum_{k=1}^{K} 2^{m-1} \cdot 2^{k-1} \cdot c_m \cdot d_k^T. \tag{9}$$

From the above formulas, the dot product is decomposed into $M \times K$ sub-operations, in which every element is 0 or 1. Because of the restriction of encoding and without using the sign bit, the above representation can only be used to encode non-negative integers. However, it's impossible to limit the weights and the values of the activation functions to non-negative integers.

In order to extend encoding space to negative integer and reduce the computational complexity, we propose a new encoding scheme, which uses {-1, +1} as the basic elements of our encoder rather than $\{0, 1\}$. Except for the difference of basic elements, the encoding scheme is similar to the rules shown in the formula (5), and is formulated as follows:

$$x^i = \sum_{m=1}^{M} 2^{m-1} \cdot \mathrm{c}_m^i, \quad \mathrm{c}_m^i \in \{-1, 1\}. \tag{10}$$

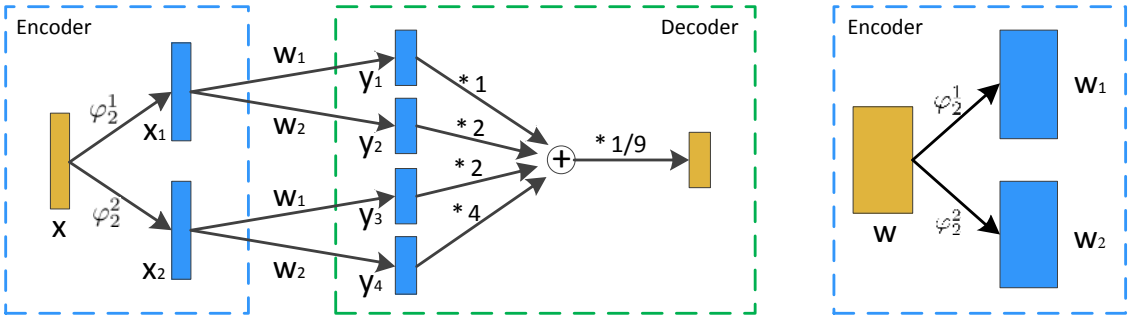

Figure 1: Architecture of fully connected layer by 2-bit encoding. We use 2BitEncoder ($\varphi_2^1(x)$ and $\varphi_2^2(x)$) to encode input data and weights in Encoder part and sum over those four results by fixing scale factors to achieve the final output in Decoder part.

where M denotes the number of encode bit, that can represent 2^M states. At this time, we can use multiple bitwise operations (*xnor* and *bitcount*) to effectively achieve the above vector multiplications. This operation mechanism is suitable for all vector/matrix multiplications.

In neural networks, matrix multiplication is the basic computation in both the fully connected and convolution layers. Based on the above decomposition mechanism of vector multiplication, we propose the following model decomposition method for quantized networks. We first use 2-bit encoding for fully connected layer as an example to introduce the mechanism of our model decomposition, the details are shown in Figure 1. x is the input data and w is the weight matrix. Here, we suppose the bias does not exist. We define an "Encoder" that can be used in the 2BitEncoder function ($\varphi_2^1(\cdot)$ and $\varphi_2^2(\cdot)$), which will be described in the next section, to encode input data. For example, x can be encoded by $x_1 \in \{-1, +1\}^N$ and $x_2 \in \{-1, +1\}^N$, where x_2 represents high bit data and x_1 represents low bit data. These variables meet the following formula: $x = x_1 + 2x_2$. In the same way, the weight w can be converted into $w_1 \in \{-1, +1\}^{M \times N}$ and $w_2 \in \{-1, +1\}^{M \times N}$. After cross multiplications, we get four intermediate variables $\{y_1, y_2, y_3, y_4\}$. Each multiplication can be considered as a binarized fully connected layer, whose elements are -1 or +1. This decomposition can result multi-branch layers, thus we call it as Multi-Branch Binary Networks (MBNs). For instance, we decompose the 2-bit fully connection operation into four branches binary operations, which can be accelerated by bitwise operations, and then sum over those four results by fixing scale factors to achieve the final output. This operation mechanism can be suitable for all vector/matrix multiplications. In addition to fully connected layers, convolution and deconvolution layers are also suit for neural networks.

M-bit Encoding Functions

As an important part in neural networks, activation functions can enhance the nonlinear characterization of networks. In our proposed model decomposition method, encoding function plays a critical role and can encode input data to multi-bits (-1 or +1). Those numbers represent the encoding of input data. For some other QNNs, several quantization functions have been given. However, it is not clear that what's the affine mapping between quantized numbers and encode bits. In this part, a list of M-bit encoding functions are proposed to produce the element of each bit that follows the rules for encoding data.

Table 1: Activation functions to limit input data to a fixed numerical range.

$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$HTanh(x) = \begin{cases} +1, & x > 1 \\ x, & -1 \leqslant x \leqslant 1 \\ -1, & -1 \leqslant x \end{cases}$
$Sigmoid(x) = \frac{1}{1+e^{-x}}$	$HReLU(x) = \begin{cases} +1, & x > 1 \\ x, & 0 \leqslant x \leqslant 1 \\ 0, & x \leqslant 0 \end{cases}$

Before encoding, the data should be limited to a fixed numerical range. Table 1 lists four activation functions. $HTanh(\cdot)$ brings the range of input data to [-1, +1], and it consists with sign function to achieve binary encoding of weights and activations (Hubara et al. 2016; Liang et al. 2017). Since the convergence of SGD obtained by using $ReLU(\cdot)$ is faster than other activation functions, we propose a new activation function $HReLU(\cdot)$ that retains the linear characteristics in the specific range and limits the range of input data to [0, 1]. Different from general activation functions mentioned above, the output of our M-bit encoding function defined below should be M numbers, which is -1 or +1. Those numbers represent the encoding of input data. Therefore, the dot product can be computed by the formula (9). In addition, at the above described experimental condition, when we use 2-bit to encode the data x and constrain to [-1, 1], there are 4 encoded states, as shown in Table 2. The affine mapping between quantized real numbers and their encoded states is given in the following table.

From the above results, we can see that there is a linear factor α between quantized real numbers and encoded states (e.g., α=3 for Table 2). When we use formula (9) to compute the multiplication of two encoded vectors, the value will be expanded α^2 times. Therefore, the result can mul-

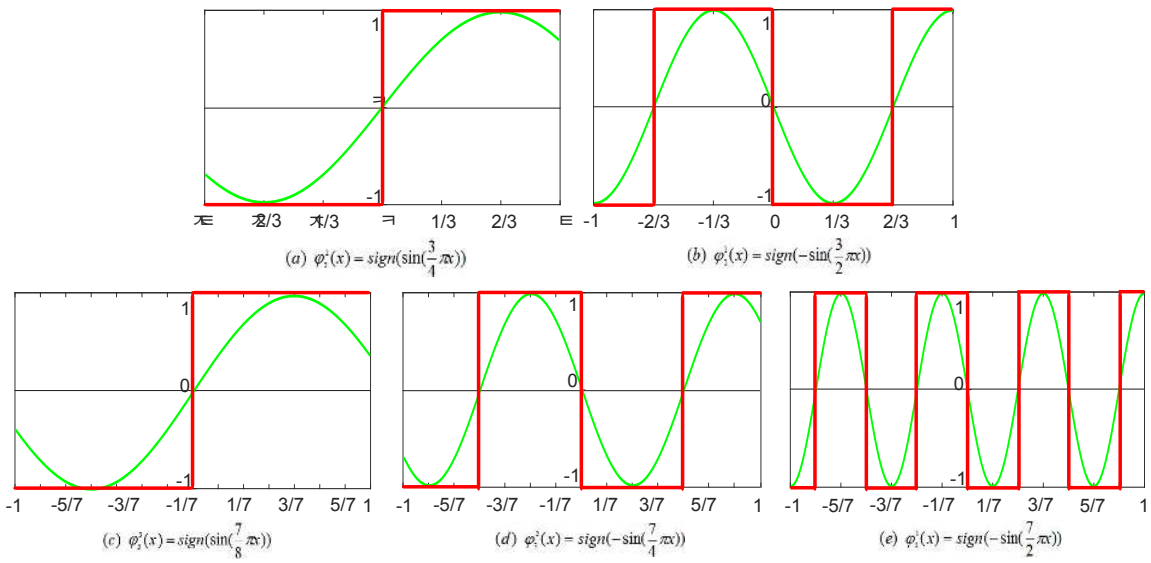

Figure 2: Encoding functions. (a) and (b) denote the encoding functions of the second bit and the first bit of 2BitEncoder. (c), (d) and (e) denote the encoding functions of the third bit, the second bit and the first bit of 3BitEncoder.

Table 2: Quantized real numbers and their Encoded states.

Quantized numbers	-1	-1/3	1/3	1
Encoded states	{-1,-1}	{-1,1}	{1,-1}	{1,1}

tiply its scale factor to get the final result, shown as $1/9$ in Figure 1. Figure 2 shows the illustration of 2-bit and 3-bit encoding functions, we can see that those encoding functions are required periodic, and each function has different periods. Naturally, we apply trigonometric functions as the basic encoder functions, which are signed as red lines. After all, we use sign function to hard divide to -1 or +1. The mathematical expression can be formulated as follows:

$$2BitEncoder(x) = \begin{cases} \varphi_2^2(x) : sign(sin(\frac{3}{4}\pi \cdot x)), \\ \varphi_2^1(x) : sign(-sin(\frac{3}{2}\pi \cdot x)), \end{cases} \tag{11}$$

where $\varphi_2^1(x)$ denotes the encoding function of the first bit (x_1^i) of 2BitEncoder, and $\varphi_2^2(x)$ represents the encoder function of the second bit (x_2^i) of 2BitEncoder. The periodicity is obviously different from others because it needs to denote more states.

Networks Training

QNNs face the problem that the derivative is not defined, thus traditional gradient optimization methods are not applicable. Hubara et al. (2016) presented the *HTanh* function to binary quantize both weights and activations, and also defined the derivative to support back-propagation (BP) training process. They used the loss computed by binarized parameters to update full precision parameters. Similarly, Rastegari et al. (2016) also proposed to update the weights with the help of the second parameters. Bengio, Léonard, and Courville (2013) discussed that using STE to train network models containing discrete variables can obtain faster training speed and better performance.

Multi-Branch Binary Networks Training

Generated by the decomposition of QNNs, MBNs need to use M-bit encoding functions to get the elements of each bit, which can be used by more efficient bitwise operations to replace arithmetic operations. We take the 2-bit encoding as an example to describe the optimization method of MBNs. The sign function of the encoder makes it difficult to implement the BP process. Thus, we approximate the derivation of the encoder function with respect to x as follows:

$$\frac{\partial \varphi_2^2(x)}{\partial x} = \begin{cases} \frac{3}{4}\pi cos(\frac{3}{4}\pi x), & -1 \leqslant x \leqslant 1 \\ 0 & \text{otherwise}, \end{cases} \tag{12}$$

$$\frac{\partial \varphi_2^1(x)}{\partial x} = \begin{cases} -\frac{3}{2}\pi cos(\frac{3}{2}\pi x), & -1 \leqslant x \leqslant 1 \\ 0 & \text{otherwise}. \end{cases} \tag{13}$$

Besides activations, all weights of networks also need to be quantized to binary values. We retain the real-valued weight w and binarized weight w_b in the training process, and apply w_b to compute loss and gradient, which is used to update w. w is constrained between -1 to +1 to avoid excessive growth. Different from weights, the binary function for w is not needed for the encoding function and directly defined as follows:

$$Binarize(x) = sign(HTanh(x)). \tag{14}$$

For this function, we have defined the gradient function of each component to constrain the search space. That is, the input of sign function can be constrained to [-1,+1]

by $HTanh(x)$, and it can also speed up the convergence. The parameters of the whole network are updated by *Adam* (Kingma and Ba 2014) in the condition for differentiability.

Quantized Networks Training

The above training scheme is proposed to optimize binary networks, which can be converted into multi-state networks. However, this converter can produce many times more parameters than the original network. If we optimize the binarized network, it may easily fall into local optimal solutions and face slow convergence speed. Based on the affine mapping between quantized numbers and fixed-point integers, we can directly optimize the quantized network and then use multi-branch binary operations in inference process. There are two quantization schemes usually applied in QNNs (Hubara et al. 2016; Zhou et al. 2016; Miyashita, Lee, and Murmann 2016), named linear quantization and logarithmic quantization. Due to the requirement of our encoding mechanism, linear quantization is used to quantize networks, and is defined as follows:

$$q_k(x) = 2\left(\frac{<(2^k-1)(\frac{x+1}{2})>}{2^k-1} - \frac{1}{2}\right), \tag{15}$$

where $<\cdot>$ denotes the rounding operation, which can quantize a real number $x \in [-1, +1]$ to a certainty state. We call it a hard ladder function, which can segment input codomain to multi-states. Table 2 lists the four states that quantized by formula (15). However, the derivative of this function is almost zero everywhere, it cannot be used in training process. Inspired by STE, we use the same technique to speed up computing process and yield better performance. We use the loss computed by quantized parameters to update full precision parameters. Note that for our encoding scheme with low-precision quantization (e.g., binary), we use *Adam* to train our model, otherwise stochastic gradient descent is used.

Experiments

Many scholars are devoted to improving the performance (e.g., accuracy and compression ratio) of QNNs, while very few researchers have studied their engineering acceleration, which is an important reason for hindering industrial promotion. Therefore, we mainly focus on an acceleration method, which is especially suitable for engineering applications. In this section, we compare the performance of our method with BWN (Courbariaux, Bengio, and David 2015), BNN (Hubara et al. 2016), XNOR-Net (Rastegari et al. 2016), TWN (Li, Zhang, and Liu 2017), and ABC-Net (Lin, Zhao, and Pan 2017) for image classification tasks on CIFAR-10 and ImageNet, and object detection tasks on Pascal VOC2007/2012 datasets.

Image Classification

CIFAR-10: CIFAR-10 is an image classification benchmark dataset, which has 50000 training images and 10000 testing images. All the images are 32×32 color images representing airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships and trucks.

We validated our method by different bit encoding schemes, in which activations and weights are equally treated, that is, both of them use the same bit-encoding. Table 3 lists the results of our method and several state-of-the-art models mentioned above. Here we use the same network architecture as in (Courbariaux, Bengio, and David 2015; Hubara et al. 2016) except for the encoding functions. We use *HTanh*$(\cdot)$ as the activation function and employ *Adam* to optimize all parameters of the network. From all the results, we can see that the representation capabilities of 1-bit and 2-bit are completely enough for small-scale datasets, e.g., CIFAR-10. Our method with low-precision encoding achieves nearly the same classification accuracy as high precision and full-precision models, while we can attain $\sim 16\times$ memory saving compared with its full-precision counterpart. When activations and weights are constrained to 1-bit, our network structure is similar to BNN (Hubara et al. 2016), and our method yields even better accuracy mainly because of our proposed encoding functions.

ImageNet: We further examined the performance of our method with different bit encoders on the ImageNet ILSVRC-2012 dataset (Russakovsky et al. 2015). This dataset consists of 1K categories images, and has over 1.2M images in the training set and 50K images in the validation set. We use Top-1 and Top-5 accuracies to report the classification performance. For large-scale training sets (e.g., ImageNet), it usually costs plenty of time and requires sufficient computing resources for classical full-precision models. It will be more hard to train quantized networks, thus the initialization of parameter values is particularly important. In this paper, we present *HReLU*$(\cdot)$ as the activation function to constraint activations. In particular, the full-precision model parameters activated by *ReLU*$(\cdot)$ can be directly used as initialization parameters for our 8-bit quantized network. After a little number of fine-tuning, 8-bit quantized networks can be well-trained. Similarly, we use the 8-bit model parameters as the initialization parameters to train 7-bit quantized networks, and so on. There has a special case, if we use *HReLU*$(\cdot)$ and 1BitEncoder function to encode activations, all the activations will be constrained to +1. Here, we use *HTanh*$(\cdot)$ as the activation function for 1-bit encoding. Note that we use *SGD* to optimize parameters when encoding bit is not less than 3, and the learning rate is set to 0.1. When the encode bit is 1 or 2, the convergent speed of *Adam* is faster than *SGD*, as discussed in (Hubara et al. 2016; Rastegari et al. 2016).

Table 3 lists the performance (e.g., accuracy, speedup ratio, memory saving ratio) of our method and several typical models mentioned above. Those results show that our method with 1-bit encoding performs much better than BNN (Hubara et al. 2016). Similarly, our method with 5-bit encoding significantly outperforms ABC-Net[5-bit] (Lin, Zhao, and Pan 2017). Moreover, our networks can be trained in such a short time, and to achieve the accuracies in our experiments only needs dozens of times fine-tuning. Of course, if we continue training the network, we can get better performance. Different from BWN and TWN, whose weights are only quantized rather than activation values, our method quantifies both weights and activation values simultane-

Table 3: Classification accuracies of Lenet on CIFAR-10 and ResNet-18 on ImageNet.

Method	CIFAR-10	ImageNet(Top-1)	ImageNet(Top-5)
BWN (Courbariaux, Bengio, and David 2015)	90.10%	60.80%	83.00%
BNN (Hubara et al. 2016)	88.60%	42.20%	67.10%
XNOR-Net (Rastegari et al. 2016)	-	51.20%	73.20%
TWN (Li, Zhang, and Liu 2017)	92.56%	61.80%	84.20%
ABC-Net[5-bit] (Lin, Zhao, and Pan 2017)	-	65.00%	85.90%
Full-Precision	91.40%	68.60%	88.70%
Encoded activations and weights			
MBN[M=K=1]	90.39%	47.10%	71.70%
MBN[M=K=2]	91.06%	56.30%	79.48%
MBN[M=K=3]	91.27%	58.69%	81.84%
MBN[M=K=4]	91.15%	59.57%	82.35%
MBN[M=K=5]	90.92%	65.09%	86.42%
MBN[M=K=6]	91.01%	67.04%	87.69%
MBN[M=K=7]	90.20%	68.37%	88.47%
MBN[M=K=8]	90.43%	68.63%	88.70%

Table 4: Comparison with different encoding bits for object detection.

Method	Full-Precision	MBN[M=K=8]	MBN[M=K=6]	MBN[M=K=5]
mAP	0.6392	0.6351	0.6131	0.5423

ously. Although BWN and TWN can obtain little higher accuracies than our method with 1-bit quantization model, our method obtains more speedup, and the speedup ratio of existing methods such as BWN and TWN is limited to $\sim 2\times$. Due to limited and fixed expression ability, existing methods (such as BWN, TWN, BNN, XNOR-Net) can not satisfy higher precision requirements. In particular, our method can provide 64 available encoding choices, and hence our encoded networks with different encoding precisions have different speedup ratios, memory requirements and experimental precisions.

Object Detection

We also use the trained ResNet-18 with the Single Shot MultiBox Detector (SSD) framework (Liu et al. 2016) to validate object detection, in which the coordinate regression task coexists with classification tasks. The normally regression task has higher requirement on value precision, therefore the application of object detection presents a new challenge for QNNs.

In this experiment, our model is trained on the VOC2007 and VOC2012 train/val set, and tested on the VOC2007 test set. ResNet-18 with the SSD framework (Liu et al. 2016) is used as the basic network. Here we use the trained model parameters in ImageNet classification to initialize SSD network parameters, after dozens of times fine-tuning the results are listed in Table 4. We use Mean Average Precision (mAP) as the criterion to evaluate the performance of our model. It is clear that our method with 8-bit encoding scheme can yield very similar performance as its full-precision counterpart. When we use 6-bit to encode parameters, the evaluation index dropped by 0.0261. If the number of encode bits is constrained to 5, the performance of this task has visibly deteriorated, while our method can achieve $\sim 5\times$ memory saving.

As the attempt in object detection tasks, our method yields good performance on the SSD framework. Similarly, it can be applied to other frameworks, e.g., R-CNN (Girshick et al. 2014), Fast R-CNN (Ren et al. 2015), SPP-Net (He et al. 2014) and YOLO (Redmon et al. 2016).

Discussion and Conclusion

$\{0, 1\}$ Encoding and $\{-1, +1\}$ Encoding

As described in (Zhou et al. 2016), there exists an affine mapping between quantized numbers and fixed-point integers. The quantized numbers are usually restricted to the closed interval [-1, +1]. For example, the mapping is formulated as follows:

$$\mathrm{x}^q = \frac{2}{2^M - 1}\mathrm{x}^{\{0,1\}} - 1, \tag{16}$$

where x^q denotes a quantized number and $\mathrm{x}^{\{0,1\}}$ denotes the fixed-point integer encoded by 0 and 1. We use a K-bit fixed-point integer to represent a quantized number w^q. The

product can be formulated as follows:

$$\mathrm{x}^q \cdot \mathrm{w}^q = \frac{4}{(2^M-1)(2^K-1)}\mathrm{x}^{\{0,1\}} \cdot \mathrm{w}^{\{0,1\}} - \frac{2}{2^M-1}\mathrm{x}^{\{0,1\}} - \frac{2}{2^K-1}\mathrm{w}^{\{0,1\}} + 1. \quad (17)$$

The right-hand side of (17) is a polynomial, which has four terms. And each term has its own scaling factor. The computation of $\mathrm{x}^{\{0,1\}} \cdot \mathrm{w}^{\{0,1\}}$ can be accelerated by bitwise operations, however, the polynomial and scaling factor will increase the computational complexity.

For our proposed quantized binary encoding scheme (i.e., $\{-1, +1\}$), the product of two numbers is defined as

$$\mathrm{x}^q \cdot \mathrm{w}^q = \frac{1}{(2^M-1)(2^K-1)}\mathrm{x}^{\{-1,1\}} \cdot \mathrm{w}^{\{-1,1\}}, \quad (18)$$

where $\mathrm{x}^{\{-1,1\}}$ and $\mathrm{w}^{\{-1,1\}}$ denote the fixed-point integers encoded by -1 and 1. Obviously, compared with the above encoding of $\{0, 1\}$, the product can be more efficiently calculated by using our proposed encoding scheme.

Linear Approximation and Quantization

As described in (Lin, Zhao, and Pan 2017; Guo et al. 2017; Xu et al. 2018), the weight w can be approximated by the linear combination of K binary subitems $\{\mathrm{w}_1, \mathrm{w}_2, ..., \mathrm{w}_K\}$ and $\mathrm{w}_i \in \{-1, +1\}^N$, which can replace arithmetic operations with more efficient bitwise operations. In order to obtain the combination, we need to solve the following problem

$$\min_{\{\alpha_i, \mathrm{w}_i\}_{i=1}^K} \left\| \mathrm{w} - \sum_{i=1}^{K} \alpha_i \mathrm{w}_i \right\|^2, \ \mathrm{w} \in \mathbb{R}^N. \quad (19)$$

When this approximation is used in neural networks, w_i can be considered as model weights. However, the scale factor α_i is introduced in this approximation, and such a scheme also expands the parameters K times. Therefore, this approximation can convert the original model to a complicated binary network, which is hard to train (Li et al. 2017) and easily falls into local optimal solutions.

For our method, we use the quantized parameters w^q to approximate w as follows:

$$\mathrm{w} \approx \frac{1}{2^K-1}\mathrm{w}^q, \ \mathrm{w} \in [-1, 1]^N, \quad (20)$$

where w^q is a positive or negative odd number, and its absolute value is not larger than 2^K-1. Different from the above linear approximation, our method can achieve the quantized weights, and directly get the corresponding encoding elements. Thus, our networks can be more efficiently trained via our quantization scheme than the linear approximation.

Conclusions

In this paper, we proposed a novel encoding scheme of using {-1, +1} to decompose QNNs into multi-branch binary networks, in which we used bitwise operations (*xnor* and *bitcount*) to achieve model compression, computational acceleration and resource saving. In particular, we can use the high-bit model parameters to initialize a low-bit model and achieve good results in various applications. Thus, users can easily achieve different encoding precisions arbitrarily according to their requirements (e.g., accuracy and speed) and hardware resources (e.g., memory). This special data storage and calculation mechanism can yield great performance in FPGA and ASIC, and thus our mechanism is a feasible idea for smart chips. Future works will focus on improving the hardware implementation and chip technology, and exploring some ways to automatically select proper bits for various network architectures (e.g., VGG and ResNet).

Acknowledgments

This work was partially supported by the State Key Program of National Natural Science of China (No. 61836009), Project supported the Foundation for Innovative Research Groups of the National Natural Science Foundation of China (No. 61621005), the National Natural Science Foundation of China (Nos. U1701267, 61871310, 61573267, 61502369, 61876220, 61876221, 61473215, and 61571342), the Fund for Foreign Scholars in University Research and Teaching Programs (the 111 Project) (No. B07048), the Major Research Plan of the National Natural Science Foundation of China (Nos. 91438201 and 91438103), the Program for Cheung Kong Scholars and Innovative Research Team in University (No. IRT_15R53), and the Science Foundation of Xidian University (No. 10251180018). Fanhua Shang is the corresponding author.

References

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Courbariaux, M.; Bengio, Y.; and David, J.-P. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, 3123–3131.

Denton, E. L.; Zaremba, W.; Bruna, J.; LeCun, Y.; and Fergus, R. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 1269–1277.

Girshick, R.; Donahue, J.; Darrell, T.; and Malik, J. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 580–587.

Guo, Y.; Yao, A.; Zhao, H.; and Chen, Y. 2017. Network sketching: Exploiting binary structure in deep CNNs. In *CVPR*, 4040–4048.

Gupta, S.; Agrawal, A.; Gopalakrishnan, K.; and Narayanan, P. 2015. Deep learning with limited numerical precision. In *ICML*, 1737–1746.

Han, S.; Pool, J.; Tran, J.; and Dally, W. J. 2015. Learning both weights and connections for efficient neural networks. In *NIPS*, 1135–1143.

Hassibi, B., and Stork, D. G. 1993. Second order derivatives for network pruning: Optimal brain surgeon. In *NIPS*, 164–171.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2014. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, 346–361.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2016. Binarized neural networks. In *NIPS*, 4107–4115.

Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research* 18(1):6869–6898.

Iandola, F. N.; Han, S.; Moskewicz, M. W.; Ashraf, K.; Dally, W. J.; and Keutzer, K. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*.

Ioffe, S., and Szegedy, C. 2015. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *ICML*, 448–456.

Jaderberg, M.; Vedaldi, A.; and Zisserman, A. 2014. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, H.; De, S.; Xu, Z.; Studer, C.; Samet, H.; and Goldstein, T. 2017. Training quantized nets: A deeper understanding. In *NIPS*, 5811–5821.

Li, F.; Zhang, B.; and Liu, B. 2017. Ternary weight networks. In *ICLR*.

Liang, S.; Yin, S.; Liu, L.; Luk, W.; and Wei, S. 2017. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* 275.

Lin, D.; Talathi, S.; and Annapureddy, S. 2016. Fixed point quantization of deep convolutional networks. In *ICML*, 2849–2858.

Lin, X.; Zhao, C.; and Pan, W. 2017. Towards accurate binary convolutional neural network. In *NIPS*, 345–353.

Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.-Y.; and Berg, A. C. 2016. SSD: Single shot multibox detector. In *ECCV*, 21–37.

Luo, J.-H.; Zhang, H.; Zhou, H.-Y.; Xie, C.-W.; Wu, J.; and Lin, W. 2018. ThiNet: Pruning CNN filters for a thinner net. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Mishra, A., and Marr, D. 2017. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*.

Miyashita, D.; Lee, E. H.; and Murmann, B. 2016. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*.

Polino, A.; Pascanu, R.; and Alistarh, D. 2018. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 525–542.

Redmon, J.; Divvala, S.; Girshick, R.; and Farhadi, A. 2016. You only look once: Unified, real-time object detection. In *CVPR*, 779–788.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 91–99.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115(3):211–252.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *CVPR*, 4510–4520.

Tai, C.; Xiao, T.; Zhang, Y.; Wang, X.; et al. 2015. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*.

Tang, W.; Hua, G.; and Wang, L. 2017. How to train a compact binary neural network with high accuracy? In *AAAI*, 2625–2631.

Vanhoucke, V.; Senior, A.; and Mao, M. Z. 2011. Improving the speed of neural networks on CPUs. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, volume 1, 4.

Wang, P.; Hu, Q.; Zhang, Y.; Zhang, C.; Liu, Y.; Cheng, J.; et al. 2018. Two-step quantization for low-bit neural networks. In *CVPR*, 4376–4384.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. In *NIPS*, 2074–2082.

Xu, C.; Yao, J.; Lin, Z.; Ou, W.; Cao, Y.; Wang, Z.; and Zha, H. 2018. Alternating multi-bit quantization for recurrent neural networks. *arXiv preprint arXiv:1802.00150*.

Zhou, S.; Wu, Y.; Ni, Z.; Zhou, X.; Wen, H.; and Zou, Y. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*.

Zhou, A.; Yao, A.; Guo, Y.; Xu, L.; and Chen, Y. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*.

Zhu, C.; Han, S.; Mao, H.; and Dally, W. J. 2017. Trained ternary quantization. In *ICLR*.

Zhuang, B.; Shen, C.; Tan, M.; Liu, L.; and Reid, I. 2018. Towards effective low-bitwidth convolutional neural networks. In *CVPR*, 7920–7928.