Two-Step Quantization for Low-bit Neural Networks

Peisong Wang[1,2], Qinghao Hu[1,2], Yifan Zhang[1,2], Chunjie Zhang[1,2], Yang Liu[4], and Jian Cheng[1,2,3*]
[1]Institute of Automation, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]Center for Excellence in Brain Science and Intelligence Technology, CAS, Beijing, China
[4]Alibaba Group, Hangzhou, China

Abstract

Every bit matters in the hardware design of quantized neural networks. However, extremely-low-bit representation usually causes large accuracy drop. Thus, how to train extremely-low-bit neural networks with high accuracy is of central importance. Most existing network quantization approaches learn transformations (low-bit weights) as well as encodings (low-bit activations) simultaneously. This tight coupling makes the optimization problem difficult, and thus prevents the network from learning optimal representations. In this paper, we propose a simple yet effective Two-Step Quantization (TSQ) framework, by decomposing the network quantization problem into two steps: code learning and transformation function learning based on the learned codes. For the first step, we propose the sparse quantization method for code learning. The second step can be formulated as a non-linear least square regression problem with low-bit constraints, which can be solved efficiently in an iterative manner. Extensive experiments on CIFAR-10 and ILSVRC-12 datasets demonstrate that the proposed TSQ is effective and outperforms the state-of-the-art by a large margin. Especially, for 2-bit activation and ternary weight quantization of AlexNet, the accuracy of our TSQ drops only about 0.5 points compared with the full-precision counterpart, outperforming current state-of-the-art by more than 5 points.

1. Introduction

Recently, deep neural networks (DNNs) have been widely studied for a variety of applications including computer vision [20, 25], speech recognition, natural language processing and so on. By learning a hierarchical representation, DNNs have achieved state-of-the-art performance among many of these tasks. However, the computational complexity of DNNs is also increasing drastically. This presents a significant challenge to the network deployment on resource limited devices, such as mobile phones and tablets.

To make DNNs less resource-intensive, several approaches have been developed by the community, such as network pruning [22, 10], low-rank approximation [7, 17, 18], architecture design [15, 12, 35], etc. A recent work [9] shows that full-precision representations are not necessary for networks to achieve high performance. The network storage can be dramatically reduced if low-bit weights are utilized. In [5, 4], it is also shown that, the internal representations of deep neural networks can also be turned into low-bit format. In this way, both the weights and activations are quantized, and the expensive floating-point multiply-accumulate operations (MACs) can be replaced by low-bit multiply-accumulate operations, or even without multiplications (in the case of binary or ternary quantization). Thus both storage and computational complexity can be reduced using low-bit quantized neural networks.

Low-bit representation can also benefit hardware accelerators like FPGAs and neural network oriented chips. Under this circumstance, the bit-width is of vital importance, i.e., lowering one bit usually means saving lots of hardware resources as well as chip area. It also dramatically simplifies the hardware design when using fewer bits. However, the problem is that when extremely-low-bit representations are utilized, the accuracy of the quantized neural network drops a lot compared with the full-precision counterpart. This is due to the noise introduced during the network quantization stage, which makes the gradient descent method hard to converge. The problem can be more severe when both weights and activations are quantized using extremely-low-bit representations. In this case, the quantized network has to learn the optimal hidden layer encodings as well as the transformation functions between adjacent layers simultaneously.

In this paper, we propose a Two-Step Quantization (TSQ) framework for low-bit quantized neural networks: code learning and transformation function learning based

*The corresponding author.

on the learned codes. Our motivation is to decouple the weight quantization from activation quantization. In the code learning step, the parameters of the transformation functions are continuous, which makes the network more stable to converge using stochastic gradient descent. In the second step of transformation function learning, the codes are already known, making the problem into a non-linear least square regression problem with low-bit constraints. The proposed framework allows some of the existing low-bit quantization approaches to be placed in context. To obtain higher accuracy, we further present new code learning and transformation function learning methods.

For the code learning, we propose the sparse quantization method. Network sparsity plays an important role in network compression and acceleration. However, very few works deal with activation sparsity. We find that activation sparsity has a profound impact on the code learning stage of quantized neural networks. Another benefit of sparse quantization is that the increased sparsity makes the network more efficient on dedicated hardware.

After the first step, we can assume that the learned codes are optimal. Thus the objective of the second step is to learn the transformation function from the codes of previous layer to the codes of current layer. We propose a general method to learn the transformation function for different bit-width. The main contributions can be summarized as follows:

- We propose a Two-Step Quantization (TSQ) framework for learning low-bit neural networks, which decomposes the learning problem into two steps: code learning and transformation function learning.
- For the low-bit code learning, we propose the sparse quantization method, which outperforms previous activation quantization methods. A novel general iterative method is proposed to solve the transformation function learning problem for different bit-width.
- Extensive experiments on ImageNet demonstrate that our TSQ method achieves more than 5% higher top-1 accuracy than current state-of-the-arts, and only has about 0.5% top-1 accuracy drop compared with the full-precision baseline.

2. Related Work

Network acceleration and compression have become a hot topic in the deep learning community. Many great methods have been developed, such as low-rank matrix/tensor approximation [7, 17, 18, 31, 33], network pruning [22, 10], network approximation [11, 26, 40] and many others [3]. Most of these methods still utilize floating-point number representations.

Recently, it is shown that full-precision is not necessary during the training of deep neural networks [9]. Using low-bit representation of weights, the network storage can be dramatically reduced, especially when extremely-low-bit numbers are used, like binary/ternary weights. In the work of [9], a 16-bit fixed-point number representation is used to train the network. In the work of [8], it is shown that using 8-bit number representation can speed up the parallel training of deep networks while maintaining the performance. In [6] and [13], the authors show that deep networks can be trained using binary weights, which may even outperform the floating-point baseline in some cases. The Ternary Weight Network (TWN) proposed in [24] is among the first methods which can achieve good results on large dataset like ImageNet [27]. Ternary weights are also investigated in the work of [23, 39]. The Incremental Network Quantization (INQ) method proposed in [37] trains networks using logarithmic weights, in an incremental manner. Trained Ternary Quantization proposed in [39] learns both ternary values and ternary assignments. Fixed-point Factorized Networks (FFN) proposed in [32] propose to use fixed-point factorization to ternarize the weights of networks. These methods can achieve comparable accuracy with full-precision counterparts on ImageNet, however, only the weights are quantized, leaving the activations in floating-point format.

Besides weight quantization, activation quantization is also widely investigated. By turning both weights and activations into low-bit format, the network computation can be conducted using only fixed-point operations, which is more efficient and resource saving, especially on dedicated hardware. Binarized Neural Networks (BNN) proposed in [14] achieves comparable accuracy on small dataset like CIFAR-10. In the work of [24], another network named XNOR-net, is proposed to binarize both weights and activations. The XNOR-net is more accurate than BNN on large dataset like ImageNet, however, the accuracy still drops by a big step. The DOREFA-net proposed in [38] investigate the effect of different bit-with for weights, activations as well as gradients. A more recent work [2] makes use of batch normalization layer and presents the Half-wave Gaussian Quantization (HWGQ) to quantize both weights and activations. Compared with weight quantization, activation quantization usually causes much higher accuracy drop. For the quantization of large networks such as AlexNet and VGG-16, the accuracy drop of these methods can be more than 5 points, or ever more than 10 points. Thus how to quantize both weights and activations using extremely low-bit representation is still a challenging problem.

3. Two-Step Quantization

Considering a typical deep neural network of L layers, given a set of training examples A_0 with ground-truth labels y and the loss function $\mathcal{L}$, the training problem can be

formulated as

$$\begin{aligned} \underset{\{W_l\}}{\text{minimize}} \quad & \mathcal{L}(Z_L, y) \\ \text{subject to} \quad & Z_l = W_l A_{l-1} \\ & A_l = \psi(Z_l), \text{ for } l = 1, 2, \cdots L \end{aligned} \tag{1}$$

For convolutional layers, each row of W_l corresponds to a convolutional kernel. The task of network low-bit quantization is to turn all weights W_l and activations A_l into low-bit format, through two quantization functions Q_w and Q_a:

$$\begin{aligned} \underset{\{W_l\}}{\text{minimize}} \quad & \mathcal{L}(Z_L, y) \\ \text{subject to} \quad & \hat{W}_l = Q_w(W_l) \\ & Z_l = \hat{W}_l \hat{A}_{l-1} \\ & A_l = \psi(Z_l) \\ & \hat{A}_l = Q_a(A_l), \text{ for } l = 1, 2, \cdots L \end{aligned} \tag{2}$$

Here Q_w and Q_a are usually step functions, which are non-differentiable and only have discrete outputs. The non-differentiable problem can be approximated by Straight-Through Estimator [1]. However, when both weights and activations are quantized, the discrete outputs of Q_w will cause problems for the stochastic gradient descent (SGD) procedure. The difficulty is that a tiny change of the weights W (caused by one step of SGD) could not immediately change weights $\hat{W}$ which are actually used during the forward and backward computation. The slow reaction in $\hat{W}$, coupled with the high variance gradient signal caused by the Straight-Through estimation of Q_a, will make the SGD process hard to converge.

The motivation behind our approach is to decouple the quantization of weights from the quantization of activations. We decompose the learning process of quantized neural networks into two steps: the code learning step and the transformation function learning step. For the first step, all weights are full-precision values and we use the proposed sparse quantization method to quantize all activations into low-bit format. After the first step, only the leaned codes A_l are kept while the learned weights are discarded, hence the name of code learning. For the second step, we will learn the transformation function from A_{l-1} to A_l, with low-bit constraints. We show that this optimization problem can be solved by the proposed iterative method. It is also shown that by a small modification, the transformation function learning has the error correction ability by taking the quantization error of previous layers into consideration. We will discuss these two steps in details in section 3.1 and section 3.2 respectively.

3.1. Sparse Quantization for Code Learning

In the code learning step, all weights are of full-precision, only activations are quantized. To obtain more efficient codes, we present a novel sparse quantization method.

Weights sparsity plays an important role in network compression and acceleration. However, there are few works deal with activation sparsity. One reason is that ReLU (Rectified Linear Units) activation function can already result in about 50% sparsity. However, we find that activation sparsity has a profound impact on the code learning of quantized neural networks. In deep neural network, large activations are usually more important than small activations, which is the foundation of attention mechanism. By turning a portion of the small positive activations into zeros, the quantization function can pay more attention to large values. Another benefit of sparse quantization is that the increased sparsity makes the network more efficient on dedicated hardware. Here we first give several previous quantization methods, and then present our sparse quantization approach in detail.

A n-bit uniform quantizer maps the full-precision activations into 2^n discrete numbers in the set of $\{0, \Delta, 2\Delta, \cdots, (2^n - 1)\Delta\}$, according to the following function.

$$Q(x) = \begin{cases} q_i & x \in (t_i, t_{i+1}], \\ 0 & x \leq 0 \end{cases} \tag{3}$$

Here $q_i \in \mathbb{R}^+$ and $q_{i+1} - q_i = \Delta$ for $i = 1, \cdots, 2^n - 1$, and $t_i \in \mathbb{R}^+$ defines the quantization intervals. The main problem is how to determine the step value Δ and quantization intervals defined by t_i.

The Half-Wave Gaussian Quantizer (HWGQ) proposed in [2] tries to alleviate this problem by resorting to batch normalization [16]. After batch normalization, the output distribution of each layer is close to Gaussian with zero mean and unit variance. Thus the optimal step value and quantization intervals for all layers are the same, which can be determined by Lloyd's algorithm by solving the following optimization function

$$Q^*(x) = \underset{Q}{\text{argmin}} E_{x \sim \mathcal{N}(0,1), x>0}[(Q(x) - x)^2] \tag{4}$$

In this paper, we explore the sparse quantization, where instead of quantize the whole positive values after ReLU, we explore to only quantize important values while set other unimportant values to zeros. This idea had been studied in network pruning [10], by turning unimportant weights to zeros. Here we explore the sparsity of activations. Like in [10], we assume big activations are more important than small activations. Thus the sparsity is introduced by setting all activations below a threshold to zeros. Formally, given a sparse threshold ϵ, the quantization function becomes

$$Q_\epsilon(x) = \begin{cases} q'_i & x \in (t'_i, t'_{i+1}], \\ 0 & x \leq \epsilon \end{cases} \tag{5}$$

Accordingly, the step value and quantization intervals can be determined by the following optimization function

$$Q_\epsilon^*(x) = \underset{Q_\epsilon}{\text{argmin}} E_{x\sim\mathcal{N}(0,1),x>\epsilon}[(Q_\epsilon(x)-x)^2] \quad (6)$$

Another problem of sparse quantization is how to determine the sparse threshold ϵ. Here we make use of the batch normalization [16]. After batch normalization, the output distribution of each layer is close to the standard normal distribution. Thus, given a sparse ratio θ, the sparse threshold ϵ can be decided by solving the following equation

$$\Phi(\epsilon) = P(x <= \epsilon) = \theta \quad (7)$$

where $\Phi(x)$ is the cumulative distribution function of standard normal distribution.

Based on the above analysis we can find that the only parameter in our sparse quantization is the sparse ratio θ we want to achieve. Here for 2-bit activation quantization of different sparse ratios ranging 50% to 75%, we give the optimal sparse threshold ϵ using Eq.7 and the optimal step value Δ using Eq.6 in Table 1.

Table 1. The optimal thresholds and step values given different sparse ratios for 2-bit sparse quantization.

θ	50%	56.25%	62.5%	68.75%	75%
ϵ	0.00	0.16	0.32	0.49	0.68
Δ	0.5388	0.5914	0.6487	0.7139	0.7889

3.2. The Transformation Function Learning

After the first step, we assume that the learned codes are optimal. Thus the objective of the second step is to learn the transformation function from the codes of previous layer $\hat{A}_{l-1}$ to the codes of current layer $\hat{A}_l$ (i.e., non-linear activation approximation). By denoting the codes of previous layer $\hat{A}_{l-1}$ and current layer $\hat{A}_l$ as X and Y, we can convert the transformation function learning problem into the following non-linear least square regression problem

$$\begin{aligned} &\underset{\Lambda,\hat{W}}{\text{minimize}} \quad \| Y - Q_\epsilon(\Lambda\hat{W}X) \|_F^2 \\ =&\underset{\{\alpha_i\},\{\hat{w}_i^T\}}{\text{minimize}} \quad \sum_i \| y_i^T - Q_\epsilon(\alpha_i \hat{w}_i^T X) \|_2^2 \end{aligned} \quad (8)$$

where $\hat{W}$ is the low-bit weights to be learned. Note that for simplicity, we discard the low-bit symbol $\hat{\bullet}$ for X and Y, because it makes no difference whether they are full-precision values or low-bit values. Like previous low-bit quantization methods [21, 2], we introduce a floating-point scaling factor α_i for each convolutional kernel $\hat{w}_i$, which is organized into a nonnegative diagonal matrix Λ. y_i^T and $\hat{w}_i^T$ denote the i-th row of Y and $\hat{W}$.

To solve the above problem, we can alternatively solve multiple subproblems as:

$$\underset{\alpha,\hat{w}}{\text{minimize}} \quad \| y - Q_\epsilon(\alpha X^T \hat{w}) \|_2^2 \quad (9)$$

where the elements of $\hat{w}$ are low-bit values.

The problem of Eq. 9 is challenging due to the discrete function Q_ϵ and the low-bit constraints of $\hat{w}$. To solve this problem, we introduce an auxiliary variable z and relax Eq 9 as:

$$\underset{\alpha,\hat{w},z}{\text{minimize}} \quad \| y - Q_\epsilon(z) \|_2^2 + \lambda \| z - \alpha X^T \hat{w} \|_2^2 \quad (10)$$

Here λ is a penalty parameter. The solution to Eq. 10 will converge to the solution of Eq. 9 when $\lambda \to \infty$. The above optimization problem can be solved in an alternating manner, i.e., solve α and $\hat{w}$ when z is fixed and vice versa.

Solving α and $\hat{w}$ with z fixed. When z is fixed, the optimization problem of 10 becomes to

$$\underset{\alpha,\hat{w}}{\text{minimize}} \quad J(\alpha,\hat{w}) = \| z - \alpha X^T \hat{w} \|_2^2 \quad (11)$$

By expanding Eq. 11, we have

$$J(\alpha,\hat{w}) = z^T z - 2\alpha z^T X^T \hat{w} + \alpha^2 \hat{w}^T X X^T \hat{w} \quad (12)$$

By setting $\partial J/\partial\alpha = 0$, the optimal value of α is given by

$$\alpha^* = \frac{z^T X^T \hat{w}}{\hat{w}^T X X^T \hat{w}} \quad (13)$$

Substituting α^* in equation 12, we can get

$$\hat{w}^* = \underset{\hat{w}}{\text{argmax}} \frac{(z^T X^T \hat{w})^2}{\hat{w}^T X X^T \hat{w}} \quad (14)$$

If $\hat{w}$ is a m-dimensional vector with n-bit values, the integer program of Eq. 14 has 2^{nm} feasible points, thus it is impractical to get the optimal solution using exhaustive search. We choose to use an alternating method to obtain the approximate solution to the problem. During each iteration, we only update one element of $\hat{w}$ but fix all the other elements. In this way, we only need to check $m2^n$ possibilities, where the bit number n is usually quite small.

Solving z with α and $\hat{w}$ fixed. When α and $\hat{w}$ are fixed, there is no coupling between the elements of the vector z, so the optimization problem can be turned into solving many one-dimensional problems as follows:

$$\underset{z_i}{\text{minimize}} \quad (y_i - Q_\epsilon(z_i))^2 + \lambda(z_i - v_i)^2 \quad (15)$$

where $v = \alpha X^T \hat{w}$ is a known vector. This problem can be solved in closed form. To further simplify this optimization problem, we can relax Q_ϵ to $\tilde{Q}_\epsilon$ as follows:

$$\tilde{Q}_\epsilon(x) = \begin{cases} M & x > M, \\ x & 0 < x \leq M, \\ 0 & x \leq 0 \end{cases} \tag{16}$$

where $M = (2^n - 1)\Delta$ is the maximum low-bit number. By this relaxation, the optimal solution to Eq. 15 can be obtained by discussing $z_i \leq 0$, $0 < z_i \leq M$ and $z_i > M$. In this way, we can get

$$z_i^{(0)} = min(0, v_i) \tag{17}$$

$$z_i^{(1)} = min(M, max(0, \frac{\lambda v_i + y_i}{1 + \lambda}) \tag{18}$$

$$z_i^{(2)} = max(M, v_i) \tag{19}$$

Thus, the optimal z_i is the one which minimizes Eq. 15.

Initialization of α and $\hat{w}$ using Optimal Ternary Weights Approximation (OTWA). During the transformation function learning step, the learnt weights are constrained to be low-bit values. We can find a good initialization for ternary quantization (2-bit quantization), which is adopted to evaluate our proposed method in the experiments part. Here we utilize weights approximation to find the initial values for α and $\hat{w}$, as follows

$$\begin{aligned} \underset{\alpha, \hat{w}}{\text{minimize}} \quad & \| w - \alpha \hat{w} \|_2^2 \\ \text{subject to} \quad & \alpha > 0 \\ & \hat{w} \in \{-1, 0, +1\}^m. \end{aligned} \tag{20}$$

where w is the learned full-precision weights during the code learning step, and m is the dimension of w.

By expanding Eq 20, we can get the optimal solution

$$\begin{aligned} \alpha^* &= \frac{w^T \hat{w}}{\hat{w}^T \hat{w}} \\ \hat{w}^* &= \underset{\hat{w}}{\text{argmax}} \frac{(w^T \hat{w})^2}{\hat{w}^T \hat{w}} \end{aligned} \tag{21}$$

Note that $\hat{w} \in \{-1, 0, +1\}^m$, thus $\hat{w}^T \hat{w}$ equals to the number of nonzeros in $\hat{w}$, and $0 \leq \hat{w}^T \hat{w} \leq m$. Assuming that $\hat{w}$ has exactly r nonzeros, then the solution of minimizing equation 20 is given by

$$\hat{w}_j = \begin{cases} sign(w_j) & abs(w_j) \text{ in top } r \text{ of } abs(w) \\ 0 & \text{others} \end{cases} \tag{22}$$

where $sign$ is the sign function and abs is the absolute value function. When r traverses from 0 to m, we can get the global optimum $\hat{w}^*$ for equation 20. We summarize our proposed ternary weight approximation method in algorithm 1.

Algorithm 1 OTWA weight approximation

Input: weight matrix $W \in R^{k \times m}$
Output: $\hat{W} \in \{+1, 0, -1\}^{k \times m}$
Output: k floating point scaling factors $\{\alpha_i\}_{i=1}^k$

```
J(r) ← 0, for r = 1, ··· , m
for i = 1, ··· , k do
    v ← abs(w_i)
    sort v in decreasing order
    for r = 1, ··· , m do
        s ← Σ_{j=1}^{r} v(j)
        J(r) ← s^2 / r
    end for
    r* ← argmax_r J(r)
    get ŵ_i* according to equation 22
    get α_i* according to equation 21
end for
```

Asymmetric Transformation Function Learning. From Eq. 8, we can see that the transformation function learning step can be conducted simultaneously for all layers. There is no coupling between different layers. However, because of the quantization error during the function learning step, when different layers are quantized independently, the quantization error can be accumulated across layers. This is a common problem for layer-wise weight compression methods like [36, 34]. We can see that in our transformation function learning, this problem can be easily solved by a small modification to the optimization problem of Eq. 8, as follows:

$$\underset{\Lambda, \hat{W}}{\text{minimize}} \quad \| Y - Q_\epsilon(\Lambda \hat{W} \tilde{X}) \|_F^2 \tag{23}$$

where $\tilde{X}$ represents the activations (codes) of previous layer from the quantized network. In other word, the asymmetric transformation function learning tries to learn the mapping from the approximate codes of previous layer (from the quantized network) to the optimal codes of current layer learned at the first step. Using this layer-wise quantization scheme, the quantization error of all previous layers can be considered during the quantization of current layer, thus preventing quantization errors from accumulating across layers.

4. Experiments

In this section, we evaluate our proposed two-step quantization method against other fixed-point quantization methods on ImageNet [27] and CIFAR-10 [19] image classification benchmarks. Experiments are conducted on two of the mostly used CNN models, i.e., AlexNet [20] and VGG-16 [29].

4.1. Implementation Details

For the experiments on ImageNet, training images are first resized to 256 pixels at the smaller dimension. We randomly crop a 224×224 (227×227 for AlexNet) image patch from an image or its horizontal reflection. No other data augmentation such as multi-scale is utilized. At test time, only the central 224×224 crop is used for prediction. Since our method relies on batch normalization, we add batch normalization layer after each convolutional or fully-connected layer. In all experiments, weight decay is set to 0.0005 and the momentum is set to 0.9. We use polynomial learning rate and the base learning rates are set to 0.05 and 0.1 for AlexNet and VGG-16. When activations are quantized, the iterations for training are 480K and 640K with a batchsize of 256 for AlexNet and VGG-16 respectively. Just as previous works [21, 2], the first and last layers are not quantized. To evaluate each part of our proposed method and to compare our method with other state-of-the-art methods, we report top-1 and top-5 classification accuracy on ImageNet.

4.2. Sparse Quantization Results

First, we want to verify the effectiveness of the proposed sparse quantization method (i.e., low-bit activation quantization). Here we mainly compare our sparse quantization (SQ) method with the HWGQ [2] method, which is the current state-of-the-art for activation quantization. The results on AlexNet are shown in Table 2. We report the results for different activation sparsity ranging from 56.25% to 75% (models denoted as SQ-i for $i = 1 \cdots 4$) with parameters shown in Table 1. Note that when the sparsity of our sparse quantization is 50%, it will become the same as HWGQ.

Table 2. Two-bit activation quantization comparison. Our sparse quantization method, denoted by SQ, is conducted under different sparsity.

Model	Sparsity (%)	Top-1 (%)	Top-5 (%)
AlexNet	50.00	58.5	81.5
HWGQ [2]	50.00	55.8	78.7
SQ-1	56.25	58.2	80.7
SQ-2	62.50	59.0	81.3
SQ-3	68.75	58.9	80.8
SQ-4	75.00	57.9	79.8

From Table 2, we can see that our sparse quantization method can achieve much higher accuracy than HWGQ. For the sparsity of 62.5%, our method only has a 0.2% top-5 accuracy drop compared with the full-precision model. And our sparse quantization method outperforms the HWGQ by a large margin (3.2% top-1 accuracy and 2.6% top-5 accuracy). Even when the sparsity is 75%, our sparse quantization method still outperforms the HWGQ by 2.1% top-1 accuracy, with only half of the computation compared with HWGQ.

Table 3. The accuracy of our two-bit activation and ternary weight quantization models before and after fine-tuning. Results are reported under different activation sparsity.

Model	Before Fine-tune		After Fine-tune	
	Top-1	Top-5	Top-1	Top-5
TFL-SQ-1	52.7	76.6	-	-
TFL-SQ-2	55.1	78.4	58.0	80.5
TFL-SQ-3	54.7	78.1	-	-
TFL-SQ-4	54.3	77.4	56.7	79.0

Another finding from Table 2 is that, when the sparsity rises from 50% to 62.5%, the accuracy will keep increasing. This result verifies that larger values are more important than smaller values. By dropping the smaller values, our sparse quantization method can better approximation the larger values, thus boosting the accuracy. However, when the sparsity keeps increasing further, the accuracy will stop increasing and begin to drop. This is because of the information loss during the sparse quantization. At a very high sparsity, most of activations become zeros and only little useful information is left for quantization.

4.3. Transformation Function Learning Results

In this section, we thoroughly evaluate the proposed transformation function learning method. Our experiments are mainly conducted on AlexNet. Table 3 shows the Transformation Function Learning (TFL) results of our method. To fully evaluate the transformation function learning ability, results under different activation sparsity are reported. Our models are denoted as TFL-SQ-i where i denotes the index of sparsity, the same as in Table 2.

By comparing the results of Table 3 and Table 2, we can conclude that our transformation function learning method is very effective and only small accuracy drop is shown. Note that even *before fine-tuning*, our method can achieve much higher accuracy than other state-of-the-art methods (Table 4). After fine-tuning, even when the activation sparsity is set to 75%, our proposed method (denoted by TFL-SQ-4) still outperforms previous state-of-the-art by a large margin.

Effeciency analysis for each part of our proposed transformation function learning method. To further show the effect of each part of our proposed method, we have conducted extensive experiments based on the SQ-2 model. The results are shown in Figure 1. We summarize the controlled models used for comparison as follows:

- **OTWA**: Weight ternarization using OTWA;
- **TFL-rand**: Asymmetric transformation function learning initialized by random ternary variables;

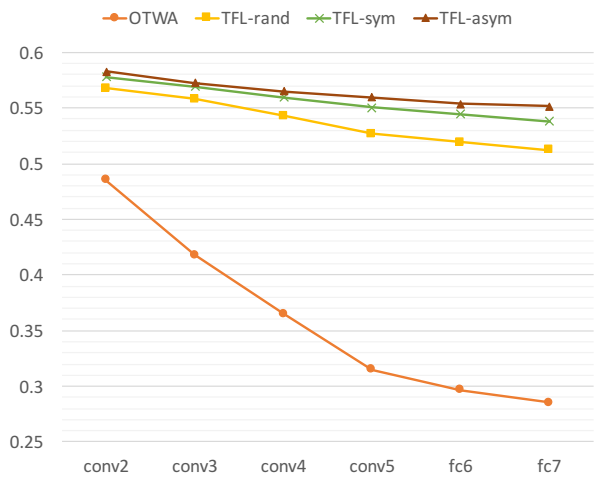

Figure 1. Top-1 accuracy after each layer is quantized for different transformation function learing settings. The results are conducted based on the SQ-2 AlexNet mode without fune-tuning.

- **TFL-sym**: Symmetric transformation function learning, initialized by OTWA;
- **TFL-asym**: Asymmetric transformation function learning, initialized by OTWA;

From Figure 1, we can see that our proposed OTWA alone can achieve about 28.5% top-1 accuracy. Thus the OTWA can serve as an initialization for our transformation function learning method. This is further confirmed by comparing the results of random initialization (TFL-rand) with that of initialization using OTWA (TFL-asym). Even without error correction, i.e., when multi-layers are processed independently, our method (TFL-sym) can still achieve very good results. This shows the effectiveness of our proposed transformation function learning method. By using asymmetric learning, our method (TFL-asym) can outperform the symmetric counterpart by about 1.3% top-1 accuracy.

4.4. Comparison with the state-of-the-art

In this section, we compare our proposed Two-Step Quantization (TSQ) method with full-precision networks as well as the current state-of-the-art low-bit quantization methods. Table 4 and Table 5 show the classification accuracy of AlexNet and VGG-16 on ImageNet dataset. For the VGG-16-BN model, we use a similar training strategies as [28]. Note that the accuracy of our implemented VGG-16-BN model in Table 5 is a bit lower than the original VGG-16 [29], which may be caused by fewer training iterations and no further data augmentation.

From the results, it is easy to conclude that the accuracy of our quantized networks is very close to the accuracy of the full-precision counterparts. Our two-step quantization method achieves negligible accuracy drop compared with the full-precision networks. For the top-1 accuracy on AlexNet, our method (denoted by TSQ) outperforms the best results by 5.3%. The gap to the full-precision model is only 0.5%. The results on VGG-16 is similar, only 2.0% top-1 accuracy drop is shown compared with the original VGG-16 model [29]. These results show that our proposed method can achieve comparable accuracy with full-precision baselines, and dramatically outperforms current state-of-the-art methods.

Table 4. Comparison with the state-of-the-art low-bit quantization methods on AlexNet. The accuracy gap to the full-precision model is also reported.

Model	Top-1	Top-5	Top-1 gap	Top-5 gap
AlexNet[2]	58.5	81.5	0	0
XNOR[24]	44.2	69.2	-12.4	-12.3
BNN[30]	46.6	71.1	-11.9	-10.4
DOREFA[38]	47.7	-	-8.2	-
HWGQ[2]	52.7	76.3	-5.8	-5.2
TSQ (ours)	58.0	80.5	-0.5	-1.0

Table 5. Comparison between our quantized VGG-16 model and the full-precison counterparts.

Model	Top-1	Top-5	Top-1 gap	Top-5 gap
VGG-16 [29]	71.1	89.9	0	0
VGG-16-BN	69.6	89.6	-1.5	-0.3
TSQ (ours)	69.1	89.2	-2.0	-0.7

4.5. Results on CIFAR-10

To compare with other methods, we have also conducted experiments on the CIFAR-10 dataset [19]. We adopt the same network architecture (VGG-small) and training strategies as [2]. Table 6 shows the results on CIFAR-10. From the results, we can see that our two-step quantization method outperforms other quantization methods by a large margin. Our TSQ method even outperforms the full-precison model by a little bit, which may be resulted from regularization ability of our low-bit quantization method.

Table 6. Comparison with the state-of-the-art low-bit quantization methods on CIFAR-10. The bit-width for activations and weights are given.

Activation	Weights	Method	error (%)
Full	Full	VGG-Small	6.82
Full	Binary	BinaryConnect [6]	8.27
Full	Ternary	TWN [23]	7.44
Binary	Binary	BNN [14]	10.15
2-bit	Binary	HWGQ [2]	7.49
2-bit	Ternary	TSQ (ours)	6.51

5. Conclusion

In this paper, we present a simple and effective network quantization framework named Two-Step Quantization (TSQ). Using TSQ, the network quantization problem can be decomposed into two steps: the code learning step and the transformation function step. For the code learning, we propose the sparse quantization method to learn both sparse and low-bit codes. The second step of our approach can be formulated as a non-linear least square regression problem with low-bit constraints, which can be solved efficiently in an iterative manner. The proposed Two-Step Quantization method is shown to dramatically outperform previous state-of-the-art low-bit quantization methods.

Acknowledgement. This work was supported in part by National Natural Science Foundation of China (No.61332016 and No.61572500) and Youth Innovation Promotion Association CAS. The authors would like to thank NVIDIA for support within the NVAIL program.

References

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

[2] Z. Cai, X. He, J. Sun, and N. Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. July 2017.

[3] J. Cheng, P. Wang, G. Li, Q. Hu, and H. Lu. Recent advances in efficient computation of deep convolutional neural networks. *Frontiers of Information Technology & Electronic Engineering*, 19(1):64–77, Jan 2018.

[4] M. Courbariaux and Y. Bengio. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[5] M. Courbariaux, Y. Bengio, and J.-P. David. Low precision arithmetic for deep learning. *arXiv preprint arXiv:1412.7024*, 2014.

[6] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3123–3131, 2015.

[7] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

[8] T. Dettmers. 8-bit approximations for parallelism in deep learning. *arXiv preprint arXiv:1511.04561*, 2015.

[9] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1737–1746, 2015.

[10] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[11] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

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

[13] Q. Hu, P. Wang, and J. Cheng. From hashing to cnns: Training binary weight networks via hashing. In *AAAI*, February 2018.

[14] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

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

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

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

[18] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[19] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[21] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[22] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In *Advances in Neural Information Processing Systems*, volume 89, 1989.

[23] F. Li, B. Zhang, and B. Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[24] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV (4)*, volume 9908, pages 525–542. Springer, 2016.

[25] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[26] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

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

[28] M. Simon, E. Rodner, and J. Denzler. Imagenet pre-trained models with batch normalization. *arXiv preprint arXiv:1612.01452*, 2016.

[29] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

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

[31] P. Wang and J. Cheng. Accelerating convolutional neural networks for mobile applications. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 541–545. ACM, 2016.

[32] P. Wang and J. Cheng. Fixed-point factorized networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[33] P. Wang, Q. Hu, Z. Fang, C. Zhao, and J. Cheng. Deepsearch: A fast image search framework for mobile devices. *Acm Transactions on Multimedia Computing Communications & Applications*, 14(1):1–22, 2018.

[34] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[35] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.

[36] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2015.

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

[38] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[39] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[40] G. Zhu, J. Wang, P. Wang, Y. Wu, and H. Lu. Feature distilled tracking. *IEEE Transactions on Cybernetics*, PP(99):1–13, 2017.