Simultaneously Optimizing Weight and Quantizer of Ternary Neural Network using Truncated Gaussian Approximation

Zhezhi He
University of Central Florida
Orlando, FL 32816
Elliot.He@Knights.ucf.edu

Deliang Fan
University of Central Florida
Orlando, FL 32816
dfan@ucf.edu

Abstract

In the past years, deep convolution neural network has achieved great success in many artificial intelligence applications. However, its enormous model size and massive computation cost have become the main obstacle for the deployment of such powerful algorithm in the low power and resource-limited mobile systems. As the countermeasure to this problem, deep neural networks with ternarized weights (i.e., -1, 0, +1) have been widely explored to greatly reduce the model size and computational cost, with limited accuracy degradation. In this work, we propose a novel ternarized neural network training method which simultaneously optimizes both weights and quantizer during training, differentiating from prior works. Instead of fixed and uniform weight ternarization, we are the first to incorporate the thresholds of weight ternarization into a closed-form representation using truncated Gaussian approximation, enabling simultaneous optimization of weights and quantizer through back-propagation training. With both of the first and last layer ternarized, the experiments on the ImageNet classification task show that our ternarized ResNet-18/34/50 only has ∼3.9/2.52/2.16% accuracy degradation in comparison to the full-precision counterparts.

1. Introduction

Artificial intelligence is nowadays one of the hottest research topics, which has drawn tremendous efforts from various fields in the past years. While computer scientists have succeeded to develop Deep Neural Networks (DNN) with transcendent performance in the domains of computer vision, speech recognition, big data processing and etc. [13]. The state-of-the-art DNN evolves into structures with larger model size, higher computational cost and denser layer connections [8, 25, 24, 11]. Such evolution brings great challenges to the computer hardware in terms of both computation and on-chip storage [10], which leads to great research effort on the topics of model compression in recent years, including channel pruning [9, 29], weight sparsification [7], weight quantization [6] and etc [10].

Weight ternarization, as a special case of weight quantization technique to efficiently compress DNN model, mainly provides three benefits: 1) it converts the floating-point weights into the ternary format (i.e., -1, 0, +1), which can significantly reduce the model size by $16\times$. With proper sparse encoding technique, such model compression rate can be further boosted. 2) Besides the model size reduction, the ternarized weight enables elimination of hardware-expensive floating-point multiplication operations, while replacing with hardware friendly addition/subtraction operations. Thus, it could significantly reduce the inference latency. 3) The ternarized weights with zero values intrinsically prune network connections, thus the computations related to those zero weights can be simply skipped.

In the previous low bit-width quantization works, such as TTN [15], TTQ [28] and BNN [5], they do re-train the models' weights but a fixed weight quantizer is used and not properly updated together with other model parameters, which leads to accuracy degradation and slow convergence of training. In this work, we have proposed a network ternarization method which simultaneously update both weights and quantizer (i.e. thresholds) during training, where our contributions can be summarized as:

- We propose a fully trainable DNN ternarization method that can jointly train the quantizer threshold, layer-wise scaling factor, and weights to minimize the accuracy degradation caused by the model compression.
- Rather than utilizing the fixed and uniform ternarizer, we are the first to incorporate the thresholds of weight ternarization into a closed-form expression using truncated Gaussian approximation, which can be optimized through back-propagation together with network's other parameters through the end-to-end training.

- We further optimize the widely used Straight-Through-Estimator (STE) [2, 5] with gradient correctness technique. It gives better gradient approximation for the non-differential staircase ternarization function, which leads to faster convergence speed and higher inference accuracy.
- In order to validate the effectiveness of our proposed methods, we apply the proposed model ternarization method on CIFAR-10 and ImageNet datasets for object classification task.

The rest of this paper is organized as follows. We first give a brief introduction to the related works regarding the topics of model compression. Then the proposed network ternarization method and the applied tricks are explained in details. In the following section, experiments are performed on both small and large scale dataset with the various DNN architectures, to evaluate the effectiveness of our proposed method. After that, the conclusion is drawn in the end.

2. Related Works

Recently, model compression on deep convolutional neural network has emerged as one hot topic in the hardware deployment of artificial intelligence. There are various techniques, including network pruning [17], knowledge distillation [18], weight sparsification [7], weight quantization [6] and etc. [22], to perform network model compression.

As one of the most popular technique, weight quantization techniques are widely explored in many related works which can significantly shrink the model size and reduce the computation complexity [10]. The famous deep compression technique [6] adopts the scheme that optimizing weight quantizer using K-means clustering on the pre-trained model. Even though the deep compression technique can achieve negligible accuracy degradation with 8-bit quantized weight, its performance on low-bit quantized case is non-ideal. Thereafter, many works are devoted to quantize the model parameters into binary [5, 20] or ternary formats [28], not only for its extremely model size reduction ($16\times \sim 32\times$), but also the computations are simplified from floating-point multiplication (i.e. *mul*) operations into addition/subtraction (i.e. *add/sub*). BinaryConnect [4] is the first work of binary CNN which can get close to the state-of-the-art accuracy on CIFAR-10, whose most effective technique is to introduce the gradient clipping. After that, both BWN in [20] and DoreFa-Net [27] show better or close validation accuracy on ImageNet dataset. In order to reduce the computation complexity, XNOR-Net [20] binarizes the input tensor of convolution layer which further converts the `Add/Sub` operations into bit-wise `Xnor` and `bit-count` operations.

Besides weight binarization, there are also recent works proposing to ternarize the weights of neural network using trained scaling factors [28]. Leng *et al.* employ ADMM method to optimize neural network weights in configurable discrete levels to trade off between accuracy and model size [14]. ABC-Net in [16] proposes multiple parallel binary convolution layers to improve the network model capacity and accuracy, while maintaining binary kernel. All the aforementioned aggressive DNN binarization or ternarization methods sacrifice inference accuracy, in comparison with the full precision counterpart, to achieve large model compression rate and computation cost reduction.

3. Methodology

3.1. Problem Definition

As for weight quantization of neural networks, the state-of-the-art work [26] typically divides it into two sub-problems: 1) minimizing the quantization noise (i.e., Mean-Square-Error) between floating-point weights and quantized weights, and 2) minimizing the inference error of DNN w.r.t the defined objective function of DNN inference. In this work, instead of optimizing two sub-problems separately, we mathematically incorporate the thresholds of weight quantizer into neural network forward path, thus enabling the simultaneous optimization of weights and thresholds through back-propagation method. In this work, given the vectorized input $\boldsymbol{x}$ and target $\boldsymbol{t}$, the network optimization problem can be described as:

$$\begin{aligned} \underset{\{\boldsymbol{w}_l, S_l, \Delta_l^{\pm}\}_{l=1}^{L}}{\arg\min} \quad & \mathcal{L}(f(\boldsymbol{x}; \{\boldsymbol{w}_l'\}_{l=1}^{L}), \boldsymbol{t}) \\ s.t.\ \boldsymbol{w}_l' &= Tern(\boldsymbol{w}_l, S_l, \Delta_l^{\pm}) \end{aligned} \tag{1}$$

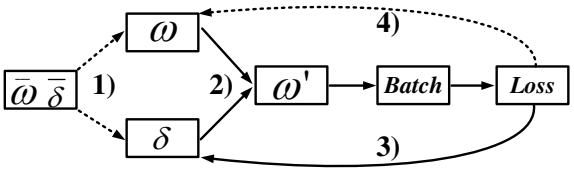
where $f(\boldsymbol{x}; \{\boldsymbol{w}_l'\}_{l=1}^{L})$ calculates the outputs of DNN, which is parameterized by the ternarized weight, w.r.t the input $\boldsymbol{x}$. $\{\boldsymbol{w}_l'\}_{l=1}^{L}$. L is the number of layers in DNN. $\boldsymbol{w}_l$ is the floating-point weights in l-th layer, before ternarization. $\mathcal{L}(\cdot, \cdot)$ is the defined loss function. The ternarization function $Tern()$ in Eq. (1) is parameterized by the ternarized value S_l and thresholds $\Delta_l^{\pm}$, where the equation detail is given as Eq. (2) in the following section.

3.2. Trainable ternarization under Gaussian approximation

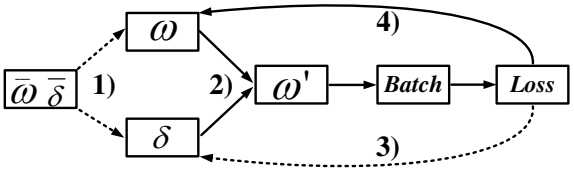
In this subsection, we will first introduce our weight ternarization methodology. Then, our proposed method to incorporate ternarization thresholds into neural network inference path, which makes it trainable through back-propagation, is discussed particularly.

3.2.1 Network Ternarization:

For the sake of obtaining a DNN with ternarized weight and minimized accuracy gap w.r.t to its full-precision coun-



(a) processes to only update thresholds $\boldsymbol{\delta}$



(b) processes to only update weights $\boldsymbol{w}$

Figure 1. The flowchart of network ternarization, where solid/dashed line indicate activate/inactive step transition. 1)⇒2)⇒3)⇒2)⇒4) steps are iteratively operated during training.

terpart, the training scheme for one iteration (as shown in Fig. 1) can be generally enumerated as four steps:

1) Initialize the weight with full-precision pre-trained model. Previous works have experimentally demonstrated that fine-tuning the pre-trained model with small learning rate normally generates a quantized model with higher accuracy. More importantly, with the pre-trained model as parameter initialization, much less number of training epochs is required to get model converged in comparison to training from scratch.

2) Ternarize the full-precision weight $w_{l,i}$ w.r.t the layer-wise thresholds $\Delta_l^{\pm}$ and quantized value S_l (*aka.* scaling coefficient) in real-time. The weight ternarization function can be described as:

$$\begin{aligned} w'_{l,i} &= S_l \cdot Tern(w_{l,i}, \Delta_l^{\pm}) \\ &= S_l \cdot \begin{cases} +1 & w_{l,i} > \Delta_l^{+} \\ 0 & \Delta_l^{-} \leq w_{l,i} \leq \Delta_l^{+} \\ -1 & w_{l,i} < \Delta_l^{-} \end{cases} \end{aligned} \tag{2}$$

Note that, the scaling coefficient S_l can be written in a closed-form function with threshold, which is key to incorporate the the quantizer optimization into DNN training without modifying the loss function. The formula derivation of S_l will be specified in Section 3.2.2. Moreover, since we propose to use symmetric thresholds centered by μ_l for weight ternarization, thus we reformat $\Delta_l^{\pm} = \mu_l \pm \delta_l$, where μ_l is the statistical mean of $\boldsymbol{w}_l$.

3) With one given input batch, this step only updates the thresholds $\{\delta_l\}_{l=1}^{L}$ through back-propagation. Meanwhile, the update of weight is suspended in the current step.

4) With the identical input batch, it repeats step-2 to synchronize the ternarized weights $\{\boldsymbol{w}'_l\}_{l=1}^{L}$ w.r.t the updated thresholds $\{\delta_l\}_{l=1}^{L}$ in step-3. Then, it suspends the update of thresholds and only allows full-precision weight base to be updated [1]. Since the staircase ternarization function ($Tern(\cdot)$ in Eq. (2)) is non-differential owing to its zero derivatives almost everywhere, we adopt the method of Straight-Through-Estimator (STE) [2] similar as previous network quantization works [28]. It is noteworthy that we propose and apply the gradient correctness technique on STE, which is critical to improving the convergence speed for weight retraining (see details in Section 3.3).

Now with the ternarized weights, the major computation of DNN is converted from the computational expensive floating-point Multiplication-and-Accumulation (MAC) to more efficient and less complex addition and subtraction (`Add/Sub`). The computation can be expressed as[2]:

$$\boldsymbol{x}_l^{\mathrm{T}} \cdot \boldsymbol{w}'_l = \boldsymbol{x}_l^{\mathrm{T}} \cdot (S_l \cdot Tern(\boldsymbol{w}_l)) = S_l \cdot (\boldsymbol{x}_l^{T} \cdot Tern(\boldsymbol{w}_l)) \tag{3}$$

where $\boldsymbol{x}_l$ and $\boldsymbol{w}'_l$ are the vectorized input and ternarized weight of l-th layer respectively. In the state-of-the-art DNN architectures, convolution/fully-connected layers normally follows a batch-normalization layer [12] (i.e., Affine function) or ReLU, where both of them perform element-wise multiplication on their input tensor (i.e., $\boldsymbol{x}_l^{\mathrm{T}} \cdot \boldsymbol{w}'_l$). Therefore, the element-wise scaling with S_l in Eq. (3) can be emitted and integrated with the following batch-norm/RELU layer in the forward path. In addition to the above description, we formalize the operations in Algorithm 1 as well for clarification.

3.2.2 Trainable thresholds utilizing truncated Gaussian distribution approximation:

It has been discussed in previous works [3, 1] that the weighted distributions of spatial convolution layers and fully-connected layers are intending to follow Gaussian distribution, whose histogram is in bell-shape, owing to the regularization effect of L^2-norm weight penalty. For example, in Fig. 2, we have shown the weight distributions and their corresponding Probability Density Function (PDF) using the calculated mean and standard deviation for each parametric layer (i.e., convolution and fully-connected layers) in ResNet-18b [8]. Meanwhile, the Shapiro-Wilk normality test [23] is conducted to identify whether the weight sample originated from Gaussian distribution quantitatively. The given test statistic $\mathcal{W}_s$ of Shapiro-Wilk normality test indicate a good normally distribution match with minimum

[1] During the training, ternarized weights are calculated from the full-precision weight base in real-time, thus the weight update is performed on the full-precision weight instead of its ternarized counterpart.

[2] For simplicity, we neglect the bias term.

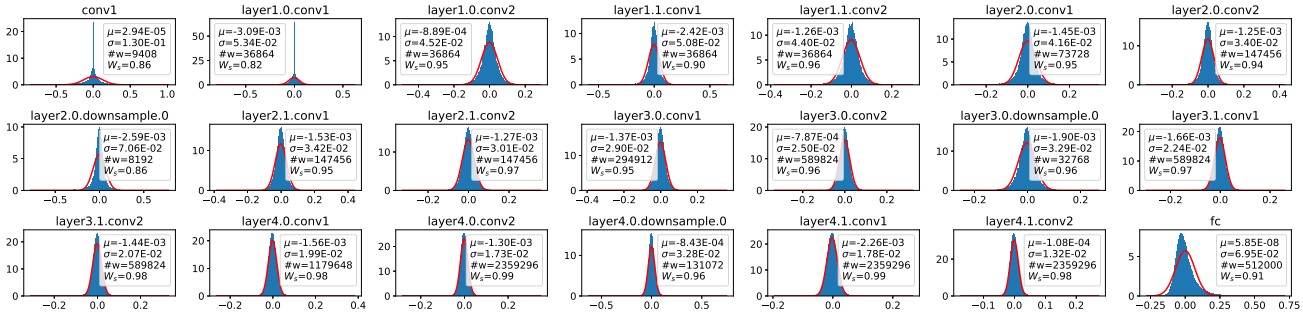


Figure 2. The histogram of weights $\boldsymbol{w}_l$ (blue shadow) along with the PDF curve (red line) of Gaussian distribution $\mathcal{N}(\mu_l, \sigma_l^2)$, for each convolution, fully-connected and residual layers in ResNet-18b [8]. μ_l and σ_l^2 are the statistical mean and variance of $\boldsymbol{w}_l$, respectively. For layers with more number of weights ($\#w$), the weights distribution conforms to the Gaussian distribution more precisely.

0.82 value. Note that, the asymmetry (i.e., Skewness) of the last fully-connected layer is due to the existence of bias term. In this work, we consider the weight of parametric layers (i.e., convolution and fully-connected layers) approximately following Gaussian distribution, then we perform the weight ternarization based on such approximation.

In order to make the thresholds $\{\delta_l\}_{l=1}^{L}$ as trainable parameters that can be updated through back-propagation, there are two criteria that have to meet:

- Thresholds $\{\delta_l\}_{l=1}^{L}$ have to be parameters within the DNN inference path in a closed-form expression.
- Such closed-form expression is differentiable w.r.t the thresholds.

Hereby, we first make the assumption that:

Assumption 1 *the weights of designated layer l are approximately following Gaussian distribution (i.e., $\boldsymbol{w}_l \sim \mathcal{N}(\mu_l, \sigma_l^2)$), where μ_l and σ_l are the calculated mean and standard deviation of the weight sample $\boldsymbol{w_l}$.*

where such assumption is the key to incorporate the thresholds into DNN inference path in a differentiable closed-form.

For the quantizer design of either uniformly or non-uniformly distributed data, the centroid is normally taken as the quantized value to minimize the quantization error [19]. Thus, for weight ternarization, the layerwise scaling coefficient (i.e., quantized value) can be described as:

$$\begin{aligned} S_l(\boldsymbol{w_l}, \Delta_l^{\pm}) &= \int_{-\infty}^{\Delta_l^-} \phi_c(x) \cdot x dx + \int_{\Delta_l}^{+\infty} \phi_c(x) \cdot x dx \\ &= E(|\boldsymbol{w}_{l,i}| | (\boldsymbol{w}_{l,i} > \Delta_l^+) \cup (\boldsymbol{w}_{l,i} < \Delta_l^-)) \end{aligned} \tag{4}$$

where $\phi_c(x)$ is the conditional PDF under the condition of $(x > \Delta_l^+) \vee (x < \Delta_l^-)$. In this work, by setting $\Delta_l^{\pm} = \mu_l \pm \delta_l$, we can approximate the Eq. (4) and reformat it into:

$$S_l(\mu_l, \sigma_l, \delta_l) = \int_{a=\mu_l+\delta_l}^{b=+\infty} \frac{\phi(x|\mu_l, \sigma_l)}{\Phi(b|\mu_l, \sigma_l) - \Phi(a|\mu_l, \sigma_l)} \cdot x dx \tag{5}$$

where $\phi(x|\mu_l, \sigma_l)$ and $\Phi(x|\mu_l, \sigma_l)$ are the PDF and CDF for Gaussian distribution $\mathcal{N}(\mu_l, \sigma_l^2)$. Such calculation can directly utilize the closed-form expression of mathematical expectation for truncated Gaussian distribution with lower bound a and upper bound b. Thus, we finally obtain a closed-form expression of scaling factor embedding trainable thresholds δ_l:

$$\alpha = \frac{a - \mu_l}{\sigma_l} = \frac{\delta_l}{\sigma_l}; \quad \beta = \frac{b - \mu_l}{\sigma_l} = +\infty \tag{6}$$

$$\begin{aligned} S_l(\mu_l, \sigma_l, \delta_l) &= \mu_l - \sigma_l \cdot \frac{\phi(\beta|0,1) - \phi(\alpha|0,1)}{\Phi(\beta|0,1) - \Phi(\alpha|0,1)} \\ &= \mu_l + \sigma_l \cdot \frac{\phi(\alpha|0,1)}{1 - \Phi(\alpha|0,1)} \end{aligned} \tag{7}$$

where $\phi(\cdot|0,1)$ and $\Phi(\cdot|0,1)$ are PDF and CDF of standard normal distribution $\mathcal{N}(0,1)$.

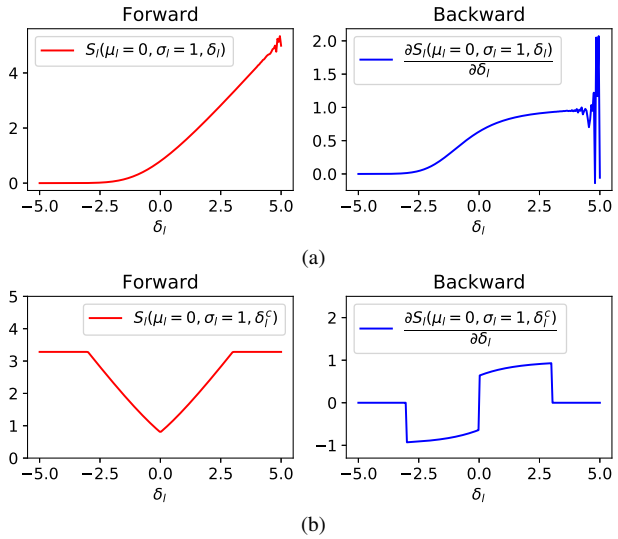


Figure 3. The forward and backward curves for (a) $S_l(\mu_l, \sigma_l, \delta_l)$ and (b) $S_l(\mu_l, \sigma_l, \delta_l^c)$ w.r.t δ_l, where δ_l^c is δ_l with clipping constraints . Note that, we choose $\mu_l = 0$ and $\sigma_l = 1$ as the example for visualization.

As shown in Fig. 3a, we plot the function of S_l in the forward and backward paths w.r.t the variation of δ_l for visualization. Since most of the popular deep learning frameworks using

numerical method (e.g., Monte-Carlo method) for distribution related calculation, there will be error for calculating S_l and $\partial S_l / \partial \delta_l$ at the tail of distribution (i.e., $\delta_l > 3\sigma_l$). For ensuring the correctness of S_l in both forward and backward path and prevent the framework convergence issue, we perform the clipping on δ_l, thus $|\delta_l| \in (0, 3\sigma_l)$. Such clipping operation is functionally equivalent as propagating δ_l through the hard-tanh function, which is piecewise linear activation function with upper-limit j and lower-limit k, then the trainable thresholds with clipping constraints can be expressed as:

$$\text{hardtanh}(x, j, k) = \text{Clip}(x, j, k) = \max(j, \min(x, k)) \tag{8}$$

$$\delta_l^c = \text{hardtanh}(\text{abs}(\delta_l), 0, 3\sigma_l) \tag{9}$$

After the substitution of δ_l with its clipped version δ_l^c, the forward and backward function of S_l is transformed from Fig. 3a to Fig. 3b. Beyond that, since the weight decay tends to push the trainable threshold of δ_l close to zero which biases the ternary weight representation towards the binary counterpart, thus we do not apply weight decay on threshold δ_l during training.

In summary, we finalize the scaling factor term and weight ternarization function to substitute the original full-precision weight in the forward propagation path:

$$S_l(\mu_l, \sigma_l, \delta_l) = \mu_l + \sigma_l \cdot \frac{\phi(\delta_l^c/\sigma_l|0,1)}{1 - \Phi(\delta_l^c/\sigma_l|0,1)} \tag{10}$$

$$Tern(w_{l,i}, \mu_l, \delta_l) = \begin{cases} +1 & w_{l,i} > \mu_l + \delta_l^c \\ 0 & \mu_l - \delta_l^c \leq w_{l,i} \leq \mu_l + \delta_l^c \\ -1 & w_{l,i} < \mu_l - \delta_l^c \end{cases} \tag{11}$$

3.3. STE with Gradient Correctness

Almost for any quantization function which maps the continuous values into discrete space, it has encountered the same problem that such stair-case function is non-differentiable. Thus, a widely adopted solution is using the so-called Straight-Through-Estimator (STE) to manually assign an approximated gradient to the quantization function. We take the STE in famous binarized neural network [5] as an example to perform the analysis (Fig. 4a), where the forward and backward of binarization function are defined as:

$$\textbf{Forward}: \quad r_o = \text{sgn}(r_i) \tag{12}$$

$$\textbf{Backward}: \quad \frac{\partial \mathcal{L}}{\partial r_o} \overset{\text{STE}}{=} \frac{\partial \mathcal{L}}{\partial r_i}\bigg|_{|r_i|\leq 1} \Longrightarrow \frac{\partial r_o}{\partial r_i}\bigg|_{|r_i|\leq 1} = 1 \tag{13}$$

where $\mathcal{L}$ is the DNN inference loss. The rule behind such STE setup is that the output of quantization function r_o can effectively represent the full-precision input value r_i. Thus, Sign$(\cdot)$ performs the similar function as $f(r_i) = r_i$ whose derivative is $\partial f(r_i)/\partial r_i = 1$. However, the rough approximation in Eq. (12) and Eq. (13) leads to significant quantization error and hamper the network training. When r_i is either too large or too small ($r_i << 1$ or $r_i >> 1$), the gradients of r_i will be stationary if binarized value r_o is not changed.

In order to encounter the drawback of naive STE design discussed above, we propose a method called *gradient correctness* for better gradient approximation. For our weight ternarization case, the full-precision weight base $\boldsymbol{w}_l$ is represented by $S_l(\mu_l, \sigma_l, \delta_l) \cdot Tern(\boldsymbol{w}_l)$, where both terms can pass back gradients to update the embedding parameters. For assigning a proper gradient to the $Tern(w_{l,i})$, we follow STE design rule which leads to the following expression:

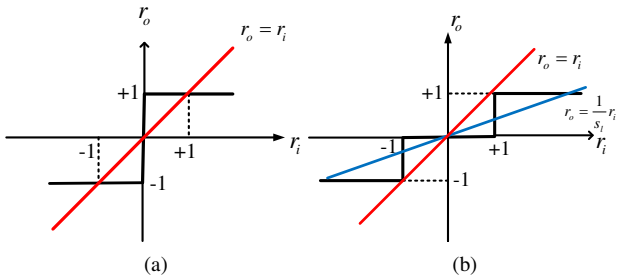


Figure 4. Analysis about the quantizer's straight-through-estimator design for (a) $r_o = sign(r_i)$ for [5] and (b) $r_o = Tern(r_i)$ in this work.

$$\frac{\partial w'_{l,i}}{\partial w_{l,i}} = \frac{\partial S_l \cdot Tern(w_{l,i})}{\partial w_{l,i}} = S_l \frac{\partial Tern(w_{l,i})}{\partial w_{l,i}} = 1 \tag{14}$$

Thus, the STE for ternarization function can be derived Eq. (14) as:

$$\frac{\partial Tern(w_{l,i})}{\partial w_{l,i}} = \frac{1}{S_l} \tag{15}$$

As seen in Eq. (15), instead of simply assigning the gradient as 1, we scale the $\partial Tern(w_{l,i})/\partial w_{l,i}$ w.r.t the value of $S_l(\mu_l, \sigma_l, \delta_l)$ in real time. As shown in Fig. 4b, STE could better approximate the gradient with adjustable gradient correctness term.

4. Experiment and Result Evaluation

4.1. Experiment setup

In this work, we evaluate our proposed network ternarization method for object classification task with CIFAR-10 and ImageNet datasets. All the experiments are performed under Pytorch deep learning framework using 4-way NVIDIA Titan-XP GPUs. For clarification, in this work, both the first and last layer are ternarized during the training and test stage.

CIFAR-10 contains 50 thousands training samples and 10 thousands test samples with 32×32 image size. The data augmentation method is identical as used in [8]. For fine-tuning, we set the initial learning rate as 0.1, which is scheduled to scale by 0.1 at epoch 80, 120 respectively. The mini-batch size is set to 128. In order to provide a more comprehensive experimental results on large dataset, we examine our model ternarization techniques on image classification task with ImageNet [21] (ILSVRC2012) dataset. ImageNet contains 1.2 million training images and 50 thousands validation images, which are labeled with 1000 categories. For the data preprocessing, we choose the scheme adopted by ResNet [8]. Augmentations applied to the training images can be sequentially enumerated as: 224×224 randomly resized crop, random horizontal flip, pixel-wise normalization. All the reported classification accuracy on validation dataset is single-crop result. The mini-batch size is set to 256.

Algorithm 1 Training both the weights and thresholds of ternarized network under the assumption that weights are following Gaussian distribution.

Require: : a mini-batch of inputs $\boldsymbol{x}$ and its corresponding targets $\boldsymbol{y}_t$, number of layers N, full-precision pretrained weights $\bar{\boldsymbol{w}}$, initial thresholds $\boldsymbol{\delta}$ full-precision weight base $\boldsymbol{w}^t$ and layer-wise thresholds $\boldsymbol{\delta}^t$ from last training iteration t, learning rate η, network inference function $f(\cdot)$.

Ensure: for current iteration index of $t+1$, updated full-precision weights $\boldsymbol{w}^{t+1}$, updated layer-wise thresholds $\boldsymbol{\delta}^{t+1}$.

{Step-1. Initialization:}
if $t = 0$ **then** ▷ This is the first training iteration
  $\boldsymbol{w} \leftarrow \bar{\boldsymbol{w}}; \boldsymbol{\delta} \leftarrow \bar{\boldsymbol{\delta}}$ ▷ load pretrained model
else
  $\boldsymbol{w} \leftarrow \boldsymbol{w}^t; \boldsymbol{\delta} \leftarrow \boldsymbol{\delta}^t$ ▷ load from last iteration
end if
{Step-2. Weight ternarization:}
for $l := 1$ to N **do**
  $\mu_l \leftarrow \boldsymbol{w}_l$.mean(); $\sigma_l \leftarrow \boldsymbol{w}_l$.std()
  $\boldsymbol{w}'_l \leftarrow S_l(\mu_l, \sigma_l, \delta_l) \cdot Tern(\boldsymbol{w}_l, \mu_l, \delta_l)$ ▷ Eqs. (10) and (11)
end for
{Step-3. Update thresholds $\boldsymbol{\delta}$ only:}
$\boldsymbol{y} \leftarrow f(\boldsymbol{x}, \boldsymbol{w}')$ ▷ forward propagation, Eq. (3)
$\mathcal{L} \leftarrow Loss(\boldsymbol{y}, \boldsymbol{y}_t)$ ▷ get inference error
for $l := N$ to 1 **do**
  $g_{\delta_l} \leftarrow \partial\mathcal{L}/\partial\delta_l$ ▷ back-propagate for gradients
  $\delta_l \leftarrow$ Update$(\delta_l, g_{\delta_l}, \eta)$ ▷ Using vanilla SGD
end for
{Repeat Step-2: from op-6 to op-10} ▷ important step!
{Step-4. Update weights $\boldsymbol{w}$ only:}
$\boldsymbol{y} \leftarrow f(\boldsymbol{x}, \boldsymbol{w}')$
$\mathcal{L} \leftarrow Loss(\boldsymbol{y}, \boldsymbol{y}_t)$
for $l := N$ to 1 **do**
  $g_{\boldsymbol{w}_l} \leftarrow \partial\mathcal{L}/\partial\boldsymbol{w}_l$ ▷ back-propagate for gradients
  $\boldsymbol{w}_l \leftarrow$ Update$(\boldsymbol{w}_l, g_{\boldsymbol{w}_l}, \eta)$ ▷ Using SGD/Adam
end for
return $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}; \boldsymbol{\delta}^{t+1} \leftarrow \boldsymbol{\delta}$

4.2. Ablation studies

In order to exam the effectiveness of our proposed methods, we have performed the following ablation studies. The experiments are conducted with ResNet-20 [8] on CIFAR-10 dataset, where the differences are significant enough to tell the effectiveness.

4.2.1 Gradient Correctness

We compare the accuracy curve convergence speed between the STE with or without the gradient correctness. As shown in Fig. 5, the network training speed with gradient correctness is much faster in comparison with the case without gradient correctness. The main reason cause the convergence speed degradation is that when layer-wise scaling factor is less than 1, without gradient correctness, the gradient of the loss function w.r.t the weights is scaled by the scaling factor due to the chain-rule. Thus, weights are updated with a much smaller step-size in comparison to the thresholds, when optimized are set up with identical parameters (e.g., learning rate, etc.).

Table 1. Ablation study of proposed method using ResNet-20 on CIFAR-10 dataset.

Configurations	Accuracy
full-precision (baseline)	91.7%
w/ gradient correctness	90.39%
w/o gradient correctness	87.89%
vanilla SGD	90.39%
Adam	56.31%
Initialize with $\delta_l = 0.05\max(\lvert\boldsymbol{w}_l\rvert)$	89.96%
Initialize with $\delta_l = 0.1\max(\lvert\boldsymbol{w}_l\rvert)$	90.24%
Initialize with $\delta_l = 0.15\max(\lvert\boldsymbol{w}_l\rvert)$	90.12%

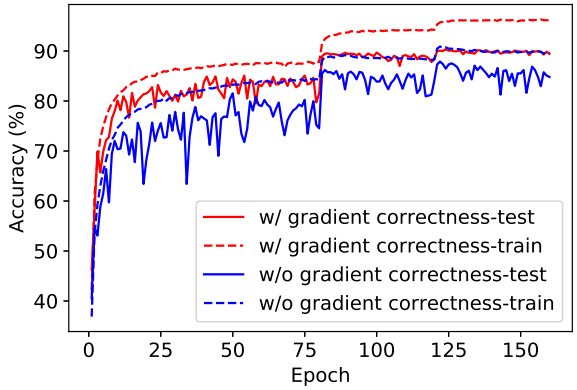


Figure 5. The accuracy evolution curve for train and test for the cases w/ or w/o gradient correctness.

4.2.2 Optimizer on thresholds

The vanilla SGD and Adam are two most adopted optimizers for quantized neural network training. Hereby, we took those two optimizers as an example to show the training evolution. Note that, since weights and thresholds are iteratively updated for each input mini-batch, we can use different optimizer for weights and thresholds. In this experiment, we use SGD for weight optimization, while using SGD and Adam on thresholds. The result depicted in Fig. 6 shows that it is better to use the same SGD optimizers to achieve higher accuracy.

4.2.3 Thresholds Initialization

In order to exam how the threshold initialization affects the network training, we initialize the threshold as $\delta_l = \{0.05, 0.1, 0.15\} \cdot \max(|w_l|)$ for all the layers. The experimental results reported in Fig. 7 shows that the initialization does not play

Table 2. Validation accuracy (top1/top5 %) of ResNet-18/34/50b [8] on ImageNet using various model quantization methods.

	Quan. scheme	First layer	Last layer	Accuracy (top1/top5)	Comp. rate
ResNet-18b					
Full precision	-	FP	FP	69.75/89.07	1×
BWN[20]	Bin.	FP	FP	60.8/83.0	∼32×
ABC-Net[16]	Bin.	FP*	FP*	68.3/87.9	∼6.4×
ADMM[14]	Bin.	FP*	FP*	64.8/86.2	∼32×
TWN[15, 14]	Tern.	FP	FP	61.8/84.2	∼16×
TTN[28]	Tern.	FP	FP	66.6/87.2	∼16×
ADMM[14]	Tern.	FP*	FP*	67.0/87.5	∼16×
APPRENTICE[18]	Tern.	FP*	FP*	68.5/-	∼16×
this work	Tern.	FP	FP	68.09/87.90	∼16×
this work	Tern.	Tern	Tern	65.83/86.68	∼16×
ResNet-34b					
Full precision	-	FP	FP	73.31/91.42	1×
APPRENTICE[18]	Tern.	FP*	FP*	72.8/-	∼16×
this work	Tern.	Tern	Tern	70.79/89.89	∼16×
ResNet-50b					
Full precision	-	FP	FP	76.13/92.86	1×
APPRENTICE[18]	Tern.	FP*	FP*	74.7/-	∼16×
this work	Tern.	Tern	Tern	73.97/91.65	∼16×

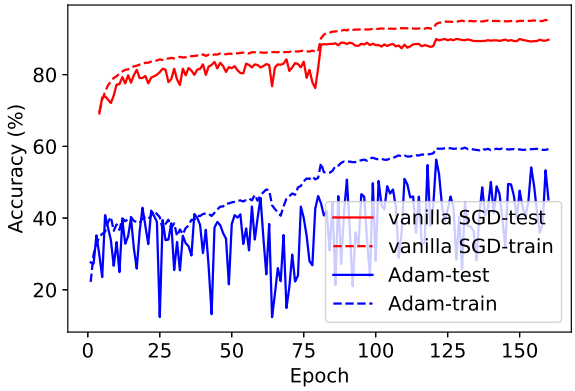


Figure 6. The accuracy evolution curve for train and test for the cases with vanilla SGD and Adam optimizer

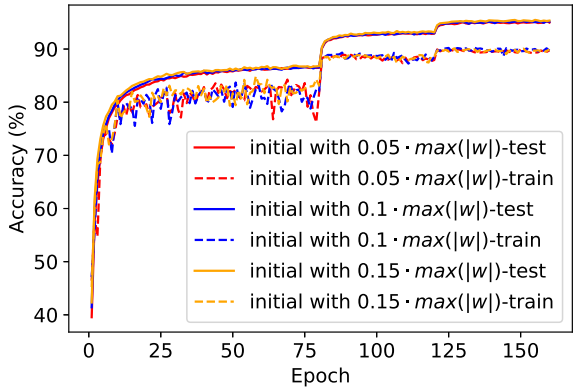


Figure 7. The accuracy evolution curve for train and test for the cases with various threshold initialization.

an important role for network ternarization in our case. The reason of that may comes to twofolds: 1) on one hand, all the layer-wise ternarization thresholds are initialized with small values where the difference is not significant. 2) on the other hand, all the thresholds are fully trainable which will mitigate the difference during training.

4.3. Performance on ImageNet dataset

Beyond the ablation studies we performed on the CIFAR-10 dataset, we also conduct the experiment on large scale ImageNet dataset with ResNet-18/34/50 (type-b residual connection) network structures. The experimental results are listed in Table 2 together the methods adopted in related works. Since for the realistic case that neural network operating on the specifically designed hardware, it is expected that all the layers are ternarized. The results shows that, our result can achieve the state-of-the-art results. The layer-wise thresholds are initialized as $\delta_l = 0.1 \times |max(\boldsymbol{w}_l)|$. We use the full-precision pre-trained model for weight initialization as described in Fig. 1. The learning rate starts from 1e-4, then change to 2e-5, 4e-6, 2e-6 at epoch 30, 40, 45 correspondingly.

5. Conclusion and future works

In this work, we have proposed a neural network ternarization method which incorporate thresholds as trainable parameter within the network inference path, thus both weights and thresholds are updated through back-propagation. Furthermore, we explicitly discuss the importance of straight-through-estimator design for approximating the gradient for staircase function. In general, our work is based on the assumption that the weight of deep neural network is tend to following Gaussian distribution. It turns out that such assumption somehow successfully returns a abstract model for network ternarization purpose.

Acknowledgement: This work is supported in part by the National Science Foundation under Grant No. 1740126 and Semiconductor Research Corporation nCORE.

References

[1] C. Baskin, E. Schwartz, E. Zheltonozhskii, N. Liss, R. Giryes, A. M. Bronstein, and A. Mendelson. Uniq: uniform noise injection for the quantization of neural networks. *arXiv preprint arXiv:1804.10969*, 2018. 3

[2] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 2, 3

[3] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015. 3

[4] 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. 2

[5] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016. 1, 2, 5

[6] 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. 1, 2

[7] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015. 1, 2

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 3, 4, 5, 6, 7

[9] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017. 1

[10] Z. He, B. Gong, and D. Fan. Optimize deep convolutional neural network with ternarized weights and high accuracy. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 913–921. IEEE, 2019. 1, 2

[11] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3, 2017. 1

[12] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 3

[13] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015. 1

[14] C. Leng, H. Li, S. Zhu, and R. Jin. Extremely low bit neural network: Squeeze the last bit out with admm. *arXiv preprint arXiv:1707.09870*, 2017. 2, 7

[15] F. Li, B. Zhang, and B. Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016. 1, 7

[16] X. Lin, C. Zhao, and W. Pan. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, pages 344–352, 2017. 2, 7

[17] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*, 2017. 2

[18] A. Mishra and D. Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*, 2017. 2, 7

[19] J. G. Proakis, M. Salehi, N. Zhou, and X. Li. *Communication systems engineering*, volume 2. Prentice Hall New Jersey, 1994. 4

[20] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016. 2, 7

[21] 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. 5

[22] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 2

[23] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965. 3

[24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 1

[25] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017. 1

[26] D. Zhang, J. Yang, D. Ye, and G. Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. *arXiv preprint arXiv:1807.10029*, 2018. 2

[27] 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. 2

[28] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016. 1, 2, 3, 7

[29] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 875–886, 2018. 1