RTN: Reparameterized Ternary Network

Yuhang Li,[1*] **Xin Dong,**[2*] **Sai Qian Zhang,**[2] **Haoli Bai,**[3] **Yuanpeng Chen,**[1] **Wei Wang**[4]
[1]University of Electronic Science and Technology of China, [2]Harvard University
[3]The Chinese University of Hong Kong, [4]National University of Singapore
{loafyuhang, haolibai, chenyuanpengcyp}@gmail.com, {xindong, zhangs}@g.harvard.edu, wangwei@comp.nus.edu.sg

Abstract

To deploy deep neural networks on resource-limited devices, quantization has been widely explored. In this work, we study the extremely low-bit networks which have tremendous speed-up, memory saving with quantized activation and weights. We first bring up three omitted issues in extremely low-bit networks: the squashing range of quantized values; the gradient vanishing during backpropagation and the unexploited hardware acceleration of ternary networks. By reparameterizing quantized activation and weights vector with full precision scale and offset for fixed ternary vector, we decouple the range and magnitude from direction to extenuate above problems. Learnable scale and offset can automatically adjust the range of quantized values and sparsity without gradient vanishing. A novel encoding and computation pattern are designed to support efficient computing for our reparameterized ternary network (RTN). Experiments on ResNet-18 for ImageNet demonstrate that the proposed RTN finds a much better efficiency between bitwidth and accuracy and achieves up to 26.76% relative accuracy improvement compared with state-of-the-art methods. Moreover, we validate the proposed computation pattern on Field Programmable Gate Arrays (FPGA), and it brings $46.46\times$ and $89.17\times$ savings on power and area compared with the full precision convolution.

1 Introduction

Deep neural networks have achieved significant improvement for various real-world applications. However, the large memory cost, computational burden, and energy consumption prohibit the massive deployment of deep neural networks on resource-limited devices. A number of methods are proposed to compress and accelerate deep neural networks, including pruning (Han, Mao, and Dally 2015), tensor decomposition (Zhang et al. 2015), and quantization (Rastegari et al. 2016).

Among these methods, low-bit network quantization is particularly helpful in network acceleration and size reduction. Binary neural networks (Courbariaux, Bengio, and David 2015) raise the attention of quantization neural networks. However, binary networks usually suffer from a large drop in terms of accuracy due to limited expressiveness. To enhance the model capacity, various multi-bit quantization methods are proposed (Zhou et al. 2016), which significantly improve the performance of quantized models but enjoy less size reduction and speed acceleration.

As a compromise between binary networks and *N*-bit networks, ternary neural networks convert full-precision parameters into merely three values and bring large memory savings with acceptable accuracy degradation. Despite ternary networks are popularly investigated (Li, Zhang, and Liu 2016; Zhu et al. 2016) in recent years, three major issues are mostly overlooked: 1) *The squashing behavior of the forward quantization function.* Most existing activation quantization methods (Cai et al. 2017; Rastegari et al. 2016) squash full precision activation into a narrow and fixed range, which could affect the expressiveness of the quantized network. 2) *The saturating behavior of the backward quantization function.* The clipped Straight-Through Estimator (STE) (Bengio, Léonard, and Courville 2013) is widely adopted in training a quantized network. Nevertheless, the gradient becomes zero when entering the saturating zone of the STE estimator. Moreover, as the network depth increases, the training could suffer from the severe problem of gradient vanishing. 3) *Hardware customization for ternary neural networks.* For networks with ternary weights and activation, the computation on most modern hardware can only be performed when the ternary values are aligned up to 2-bit representation. Compared with 2-bit quantization, it is yet less explored to utilize some nice properties of ternary values to design a more efficient computation pattern and save more energy.

In this paper, we propose a reparameterized ternary network (RTN) to resolve the three issues. Specifically, in RTNs both weights and activation are ternarized, followed by a reparameterization with scale and offset parameters. The reparameterization can easily alleviate the first two issues mentioned above. Specifically, in order to avoid the squashing behavior of quantization function during the forward pass, the learnable scale and offset parameters on network parameters enable dynamic adjustment of the quantization

[*] Equal Contribution

range and thereon enhances the capacity of the ternary network. To tackle the saturating behavior of the clipped STE function, with the chain rule of derivatives we can decompose the gradient of activation after reparameterization with respect to that of activation before reparameterization, as well as the gradients of scale and offset parameters. Consequently, even though the gradient of activation before reparameterization saturates, the optimization can still proceed as a result of learning the reparameterization parameters.

Finally, to address the third issue, we build a customized hardware prototype on FPGA for the reparameterized ternary network. We design an efficient encoding and computation pattern to conduct dot products between two ternary vectors, saving extra energy compared to the previous implementation of 2-bit networks.

Experimental results on large scale tasks like ImageNet indicate that our proposed method significantly improves the capacity of the ternary network, and achieves up to 26.76% relative improvement of accuracy on ResNet-18 against state-of-the-art binary and low-bit networks. Moreover, our hardware prototype on FPGA achieves $3.43\times$ and $4.17\times$ savings on power and area respectively comparing to traditional implementations of the 2-bit network.

2 Related Work

Recent work on network compression shows that full precision computation is not necessary for the training and inference of DNNs (Gupta et al. 2015). To achieve higher compression and acceleration ratio, extremely low-bit like binary weights (Rastegari et al. 2016) have been studied. (Li, Zhang, and Liu 2016; Zhu et al. 2016) further improve the performance by ternarizing weights to achieve higher representation ability. TWN minimizes the Euclidean distance between ternary weights and the full precision weights. Instead of the symmetric ternarization, TTQ uses an asymmetric ternarization to achieve higher performance but less hardware convenience.

Substantial speed up requires further quantization for activation, which is generally more challenging than weights quantization (Cai et al. 2017). (Courbariaux et al. 2016) uses $+1$ and -1 to represent both weights and activation and XNOR-Net (Rastegari et al. 2016) further adds scaling factors for binary weights to improve accuracy. Higher-order Residual Quantization (Li et al. 2017a) uses two 1-bit tensors to approximate the full precision activation, but the computation speed would reduce to half. To take advantage of ReLU (Nair and Hinton 2010) and introduce sparsity in quantized activation, (Cai et al. 2017) uses Half-wave Gaussian Quantization to approximate ReLU. The quantized activation function has the form of a step-wise function, which always has zero gradients with respect to its input. To circumvent this problem, Straight-Through Estimator (STE) (Bengio, Léonard, and Courville 2013) is adopted. STE approximates backward function of arbitrary functions with (clipped) identity function, and several studies (Liu et al. 2018; Zhou et al. 2016) attempt to reduce this mismatch between forward and backward to improve performance. (Choi et al. 2018; Baskin et al. 2018) propose to learn the clipping parameters and achieve better results. (Gong et al. 2019) leverage $tanh$ function to approximate the gradient of quantization, however, there is still a large accuracy gap between extremely low-bit and full precision models.

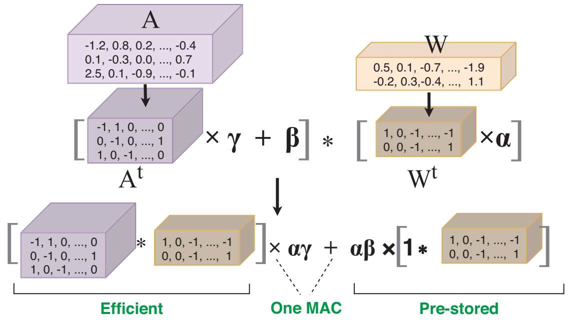

Figure 1: The overall workflow of our proposed reparameterized ternary quantization. The weights and activation are reparameterized by a scale (and an offset) factor to achieve higher network capacity. The convolution is efficient with our specially designed computation pattern.

3 Methodology

We first introduce some basic notations for our proposed method. Consider one weight filter in a convolution layer, it can be denoted by $\boldsymbol{W} \in \mathbb{R}^{c \cdot k \cdot k}$, where c and k are input channels and kernel size, respectively. Suppose one instance is fed to the network, and the corresponding feature map is denoted by $\boldsymbol{A} \in \mathbb{R}^{c \cdot k \cdot k}$. Then the output of one unit in the next layer can be computed by the dot product[*] as $z = \phi(\boldsymbol{W}^T \boldsymbol{A})$, where $\phi(\cdot)$ is the Rectified Linear Units (Nair and Hinton 2010).

Our proposed reparameterized ternary network (RTN) consists of the linear transformations on both weights and activation of the network. The reparameterization allows the dynamic adjustment of the quantization range, and avoid the issue of gradient vanishing during the quantized training. Besides, we also customize hardware implementations for RTN by leveraging the nice properties of ternary networks. The overall workflow of RTN is shown in Figure 1.

3.1 Reparameterized Ternarization

Activation Ternarization Previous work (Wan et al. 2018; Deng et al. 2018) on ternary networks argue that the degradation of quantized network mainly comes from the limited quantization levels. However, it is rarely observed that the quantization function they adopt usually squashes the input into fixed ranges and therefore harms the network expressiveness significantly. In a ternary neural network, the quantization function is applied to both weights and activation, which highly restricts the capacity of the quantized model. Therefore, in this paper, we propose a reparameterized quantizer to enhance the model expressiveness. First,

[*] For convolutional layers, this can be done by the im2col operation.

the ternarization function is given by:

$$\boldsymbol{A}_i^t = Q(\boldsymbol{A}_i) = \begin{cases} \mathrm{sign}(\boldsymbol{A}_i) & \text{if } |\boldsymbol{A}_i| > 0.5 \\ 0 & \text{otherwise} \end{cases}, \tag{1}$$

Since activation function in RTN is ReLU, the output of this function is always non-negative, which means activation can never be quantized to -1. We apply Batch Normalization (BN) after ReLU to recreate negative activation. As a result, the quantization values can be made full use of.

After normalizing the inputs of each layer, BN usually applies an affine transformer to increase model capacity. Here we use

$$\bar{\boldsymbol{A}} = k\boldsymbol{A} + b \tag{2}$$

to denote the transformation. With BN transformation, consequently the quantization function can be formulated as follows:

$$\boldsymbol{A}_i^t = Q(\bar{\boldsymbol{A}}_i) = \begin{cases} 1 & \text{if } \boldsymbol{A}_i > \frac{0.5-b}{k} \\ -1 & \text{if } \boldsymbol{A}_i < -\frac{0.5+b}{k} \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where the learnable BN parameters k and b can adaptively adjust the quantization threshold (0.5) in Equation 1. In spite of the quantization threshold is learnable, however, the ternary activation $\boldsymbol{A}_i^t$ only contains fixed ternary values (i.e. $\boldsymbol{A}^t \in \{+1, 0, -1\}^n$). We consider a ternary activation $\bar{\boldsymbol{A}}^t \in \{-\gamma+\beta, \beta, \gamma+\beta\}$ and we further reparameterize $\bar{\boldsymbol{A}}^t$ by

$$\bar{\boldsymbol{A}}^t = \gamma \cdot \boldsymbol{A}^t + \beta, \tag{4}$$

where γ is the magnitude scale factor and β is the offset. With γ and β, the reparameterized ternary activation can dynamically change the squashing range, improving the network capacity with little increase of model size and computation. Here, we refer to $\boldsymbol{A}^t$ as fixed ternary activation, because the ternary values are fixed and it only controls the direction of activation.

Our method can be reduced to a number of previous methods by taking different β and γ. For example, when $\beta = \gamma$, we squash the activation into range $[0, \beta+\gamma]$, which is equivalent to HWGQ (Cai et al. 2017). When $\gamma = \mathbb{E}_{|\boldsymbol{A}|>0.5}(|\boldsymbol{A}|)$ and $\beta = 0$, the Euclidean distance from $\bar{\boldsymbol{A}}^t$ to full precision activation $\boldsymbol{A}$ is minimized, and our approach resembles XNOR-Net (Rastegari et al. 2016). Note that the scale factor mentioned in XNOR-Net is different from ours, their scale factor has to be calculated from full precision activation for each forward pass as a running variable, which is not practical.[†] A similar idea on decoupling the vector magnitude from its direction for full precision weights can also be found in (Salimans and Kingma 2016). Note that these factors are designed in a layerwise pattern, so value ranges may change across different layers depending on γ and β.

Weight Ternarization In a similar spirit to activation ternarization, we first apply linear transformation for network weights that resembles the BN layer in activation to obtain learnable quantization thresholds. For each weight filter $\boldsymbol{W} \in \mathbb{R}^{c \cdot k \cdot k}$, the weights transformers can be defined as follows:

$$\bar{\boldsymbol{W}} = k_{\boldsymbol{W}} \boldsymbol{W} + b_{\boldsymbol{W}}, \tag{5}$$

where $k_{\boldsymbol{W}}$ and $b_{\boldsymbol{W}}$ are learnable parameters. Then the transformed weights $\bar{\boldsymbol{W}}$ are quantized by the same function in Equation 1, i.e $\boldsymbol{W}^t = Q(\bar{\boldsymbol{W}})$. As a consequence, the weights can adjust its quantization threshold like activation. To obtain flexible quantized values, we follow a similar way to reparameterize $\boldsymbol{W}^t$ by $\bar{\boldsymbol{W}}^t = \alpha \boldsymbol{W}^t \in \{-\alpha, 0, \alpha\}$, where α is the scale factor. Note that the offset is not included under the consideration of additional computation overhead.

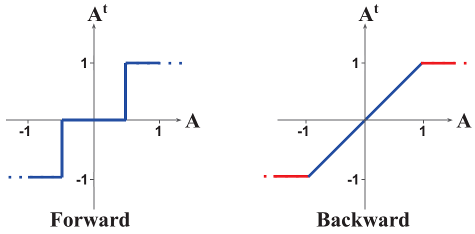

Figure 2: The forward and backward functions for fixed ternary activation $\boldsymbol{A}^t = Q(\boldsymbol{A})$. The line in red is referred to as saturating zone whose gradient is always zero.

3.2 Backward Update in Reparameterized Ternarization

A typical approach to propagate the gradients through the quantized activation is the clipped Straight-Through Estimator (STE): $\frac{\partial \boldsymbol{A}^t}{\partial \bar{\boldsymbol{A}}} = \mathbf{1}_{|\bar{\boldsymbol{A}}| \le 1}$, which is exactly the gradients of *hard tanh*. Despite being successfully used in previous methods (Rastegari et al. 2016), *hard tanh* suffers from the saturating problem. When $|\bar{\boldsymbol{A}}_i| \ge 1$, the gradient of $\bar{\boldsymbol{A}}_i$ becomes zero, which enters the saturating zone as shown in the red part of Figure 2 . The saturating behavior of STE can cause gradient vanishing for weights as the depth of the network increases, which slows down and even hurts the convergence of the model. Furthermore, once activation falls into the saturating zone, they will get stuck and barely find a way out because both $\boldsymbol{W}, \boldsymbol{W}^t$ and $\boldsymbol{A}, \boldsymbol{A}^t$ remain unchanged.

Fortunately, our reparameterized ternary activation can alleviate this problem easily. Consider L as the loss function, the derivative w.r.t. to $\bar{\boldsymbol{A}}^t$ can be written as

$$\frac{\partial L}{\partial \gamma} = \boldsymbol{A}^t \frac{\partial L}{\partial \bar{\boldsymbol{A}}^t}, \; \frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial \bar{\boldsymbol{A}}^t}. \tag{6}$$

It can be observed that since we decouple the scale γ and offset β from fixed ternary activation $\boldsymbol{A}^t$, even when $|\bar{\boldsymbol{A}}| \ge 1$ the reparameterized ternary activation $\bar{\boldsymbol{A}}^t$ can still be optimized as a result of learning γ and β. Consequently, the entire network can converge faster and reach a better optimum in the loss landscape.

Furthermore, our reparameterized ternarization has another benefit that can dynamically adjust the learning rate of network parameters. Consider the gradients w.r.t to the

[†] As a result they abandon this scale factor for quantized activation in officially released implementation.

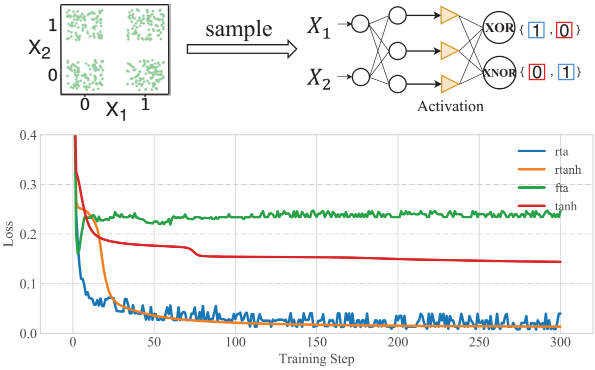

Figure 3: The XOR-XNOR toy model architecture and the training curve for the toy model.

activation,

$$\frac{\partial L}{\partial \bar{\boldsymbol{A}}} = \frac{\partial L}{\partial \bar{\boldsymbol{A}}^t} \frac{\partial \bar{\boldsymbol{A}}^t}{\partial \bar{\boldsymbol{A}}} = \gamma \mathbf{1}_{|\boldsymbol{A}| \leq 1} \frac{\partial L}{\partial \bar{\boldsymbol{A}}^t}, \tag{7}$$

where γ can be absorbed in learning rate as a multiplier, making the training of the network robust to the value of the learning rate. Learnable scale factor also has been studied in (Salimans and Kingma 2016; Zhu et al. 2016), in which they claim a similar effect as well.

3.3 The XOR-XNOR Toy Problem

To demonstrate how the reparameterized ternarization improves the capacity of the quantized model, we give a toy example on a two-layer neural network, as is shown in Figure 3.

The 2-layer network is designed to learn two logical functions, XOR$(\mathbf{x}_1, \mathbf{x}_2)$ and XNOR$(\mathbf{x}_1, \mathbf{x}_2)$ respectively. 4 different kinds of activation function are compared: fixed ternary activation (fta), reparameterized ternary activation (rta), the hyperbolic tangent activation ($tanh$) and the reparameterized hyperbolic tangent activation ($rtanh$). Inputs are sampled from a Bernoulli distribution pulsing a uniform noise, $\{(\mathbf{x}_1 = \mathbf{z}_1 + \epsilon_1, \mathbf{x}_2 = \mathbf{z}_2 + \epsilon_2) | \mathbf{z}_1, \mathbf{z}_2 \sim \mathcal{B}(p = 0.5), \epsilon \sim \mathcal{U}(-0.3, 0.3)\}$. Outputs are either 0 or 1. The network has a hidden layer consisting of 3 neurons without bias term. To better observe behaviors of quantized activation, we keep the weights as full precision numbers. We report the mean square error (MSE) during training. More implementation details are in the Appendix.

The training curve is shown in Figure 3. Compared with fixed ternary activation (*fta*) and reparameterized ternary activation (*rta*), hyperbolic tangent (*tanh*) is a full precision function with a fixed squashing range $[-1, +1]$, which is supposed to have better representation ability than ternary activation. However, our *rta* achieves lower MSE than *tanh*, because the scale and offset factors alleviated the squashing issue. Similarly, if we reparameterize *tanh*, *rtanh* can achieve even lower MSE than *rta* and *tanh*. Moreover, the scale and offset factors of *rtanh* are $\gamma^* = 0.46$ and $\beta^* = 2.02$ respectively, which substantially changes the squashing range from $[-1, 1]$ into $[1.56, 2.48]$. As we observed from the empirical result, the range of activation is at least as important as the number of quantization levels.

Table 1: This table shows the bit encoding scheme of our ternary values and 2-bit quaternary values.

2-bit Representation		Our Ternary	2-bit Network
1st bit	2nd bit	True Value	True Value
0	0	0	0
0	1	0	1
1	0	-1	2
1	1	+1	3

3.4 Efficient Computation Pattern

How to Compute Dot Product between Two Ternary Vectors To support our ternary network (ternary weights + ternary activation), in this section, we propose an efficient way to compute the dot product between the ternary weights and activation vectors, which is the core operation for both convolution and linear layers. A special bit encoding scheme is adopted for the ternary weights and activation. More specifically, we use two bits to represent the ternary weights and activation, where the first bit indicates that whether this number is zero or not, and the second bit indicates the sign of this number. Table 1 shows the detailed encodings for all ternary values -1, 0 and $+1$. Under this encoding scheme, zero can be represented by either 00 or 01.

Now, we have $\boldsymbol{W}^t, \boldsymbol{A}^t \in \{+1, 0, -1\}^{ck^2}$ and we will encode them into 2-bit vector representations. Suggest $\boldsymbol{W}_1 \in \{0, 1\}^{ck^2}$ is a vector contains the first bit of all entries in ternary weights. $\boldsymbol{W}_2$ contains the second bit and we define $\boldsymbol{A}_1, \boldsymbol{A}_2$ in a similar way to represent the activation. The dot product can be computed using bit-wise operations.

$$(\boldsymbol{W}^t)^T \boldsymbol{A}^t = bC\left(\mathbf{c}\right) - 2 \times bC\left(\left(\boldsymbol{W}_2 \oplus \boldsymbol{A}_2\right) \wedge \mathbf{c}\right), \tag{8}$$

where $\mathbf{c} = \boldsymbol{W}_1 \wedge \boldsymbol{A}_1$, and $bC(.), \wedge$ and $\oplus$ are bitCount, AND, XOR bit-wise operations. Specifically, $bC(.)$ returns the number of 1 (logic high) in a vector. As indicated by Equation 8, the convolution can be computed efficiently via simple Boolean operations.

Figure 4(a) shows the hardware design for the vector multiplication shown in Equation 8. Given the two input vectors, the circuit computes the scale product between each pair of elements of the two vectors, the partial results $bC\left(\mathbf{c}\right)$ and $bC\left(\left(\boldsymbol{W}_2 \oplus \boldsymbol{A}_2\right) \wedge \mathbf{c}\right)$ are saved inside two 32-bit counters. The multiplication with two shown in Equation 8 can be easily achieved by shifting the partial results to the left by 1 bit. A substractor is used to perform the substraction operation shown in Equation 8.

For comparison, we compute the dot product of 2-bit quaternary weights and activation because they share the same size of our model. We use the computation pattern introduced in DoReFa-Net (Zhou et al. 2016) to compute the dot product.

The quaternary vector multiplication can be computed by performing AND between each bit of inputs. Figure 4(b) shows the hardware design for the vector multiplication

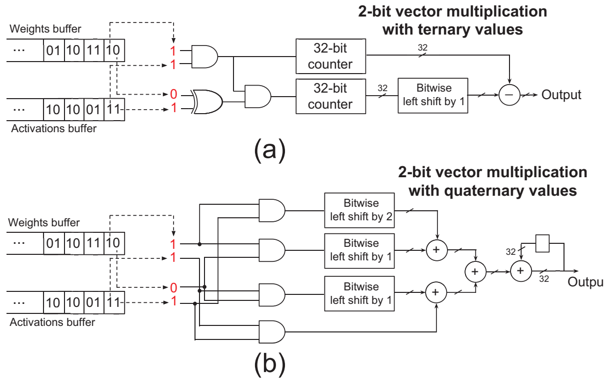

Figure 4: This figure illustrates the hardware implementation of the dot product between, (a) ternary weights and ternary activation (b) 2-bit weights and 2-bit activation.

in (Zhou et al. 2016). The circuit takes the two input vectors and calculates the scale product between each pair of elements. The multiplication with power-of-two is implemented by using bitwise shift operations. Four adders are employed to sum the partial results. We also evaluate the performances of the two designs shown in Figure 4 in terms of power consumption, computation latency, and area in the next section.

How To Deal With γ and β Actually, our reparameterized ternary activation has two extra parameters, scale γ and offset β, besides fixed ternary activation. We demonstrate that it only introduces negligible computation complexity. With the quantized ternary weights $\alpha \boldsymbol{W}^t$ and reparameterized ternary activation $\bar{\boldsymbol{A}}^t$, the input of the next layer can be computed by

$$\begin{aligned} z &= \phi\left(\alpha \boldsymbol{W}^t * \bar{\boldsymbol{A}}^t\right) = \phi\left(\alpha \boldsymbol{W}^t * (\gamma \boldsymbol{A}^t + \beta)\right) \\ &= \phi\left(\alpha\gamma(\boldsymbol{W}^t \otimes \boldsymbol{A}^t) \ + \ C\right), \quad C = \alpha\beta(\mathbf{1} \otimes \boldsymbol{W}^t), \end{aligned} \tag{9}$$

where ϕ is the ReLU function, $\otimes$ is the dot product between ternary vectors, $\mathbf{1}$ denotes the matrix with all elements equal 1. As a matter of fact, the second terms C in Equation 9, is a constant, which can be pre-stored in the cache. As shown in Figure 1, when performing the convolution, we first calculate the ternary value convolution efficiently with Boolean operations, then we only need to conduct one multiply-accumulate (MAC) operations to get the final results.

Reparameterized Ternary Activation Can Adjust Sparsity Automatically Interestingly, we can modify the expression of Equation 9 and fold the second term into ReLU to make it more hardware friendly,

$$z = \phi\left(\alpha\gamma(\boldsymbol{W}^t \otimes \boldsymbol{A}^t) \ + \ C\right) = \alpha \cdot \phi_T\left(\gamma(\boldsymbol{W}^t \otimes \boldsymbol{A}^t)\right), \tag{10}$$

where $\phi_T(\boldsymbol{x}) = \max(0, \boldsymbol{x} + T)$ is the ReLU parameterized by the sparsity threshold $T = \beta(\mathbf{1} \otimes \boldsymbol{W}^t)$.

Apparently, T controls the sparsity of z. This reveals another effect of our reparameterized ternary activation. It can control the sparsity of the activation. The sparsity of activation has been studied in (Wang et al. 2018), in which they find that sparsity has a profound impact on accuracy. However, (Wang et al. 2018) manually sets the sparsity threshold to increase the sparsity, in which they believe the quantization error can be reduced and larger activation is more important based on the attention mechanism. In our method, the sparsity threshold is given by $\beta(\mathbf{1} \otimes \boldsymbol{W}^t)$, which can be dynamically tuned during the training by offset factor β for every layer. We give the sparsity record in Section 4.2 to show that our method concurs with (Wang et al. 2018).

4 Experiments

In this section, we first present some empirical evaluations of the reparameterized ternary network (RTN) on two real-world datasets: ImageNet-ILSVRC2012 (Russakovsky et al. 2015) and CIFAR-10 (Krizhevsky, Hinton, and others 2009), then we evaluate the performance of the hardware implementation for RTN in terms of power consumption and area.

We adopt a number of popular neural architectures for evaluation: ResNet (He et al. 2016), AlexNet (Krizhevsky, Sutskever, and Hinton 2012), MobileNet (Howard et al. 2017) and Network-In-Network (NIN) (Lin, Chen, and Yan 2013). Two sets of strong baselines are chosen for comparison: 1) quantizing weights only: BWN (Rastegari et al. 2016), TWN (Li, Zhang, and Liu 2016), and TTQ (Zhu et al. 2016); 2) quantizing both weights and activation: XNOR (Rastegari et al. 2016), Bi-Real (Liu et al. 2018), TBN (Wan et al. 2018), HWGQ (Cai et al. 2017), DoReFa-Net (Zhou et al. 2016), PACT (Choi et al. 2018) and HORQ (Li et al. 2017b).

For our methods, we compare results and with reparameterization on weights and activation, denoted as RTN-F (fixed) and RTN-R (reparameterized) respectively. We also evaluate our method when only quantizing weights.

We highlight substantial accuracy improvement (up to 13% absolute improvement compared with XNOR-Net) of our RTN for ResNet-18 on ImageNet. Such improvement mainly comes from: 1) zero is introduced into quantized activation to get the fixed ternary activation $\{-1, 0, +1\}$, 2) dynamically adjusting the quantization range of weights and activation by transformers in Equations (2) and (5), and 3) learnable scale and offset are adopted for the fixed ternary activation to get the reparameterized ternary activation $\{\gamma+\beta, \beta, -\gamma+\beta\}$ which has much better representation ability with negligible computation overhead.

Compared with several 2-bit models, RTN has the lowest degradation from full precision models and achieves comparable accuracy. In Section 4.4, we implement our ternary multiplication circuit and other 2-bit multiplication circuit used in (Zhou et al. 2016; Choi et al. 2018; Li et al. 2017b) and show that the circuit for multiplication with ternary values significantly outperforms that for multiplication with 2-bit values in terms of power and area.

4.1 Implementation

We follow the implementation setting of other extremely low-bit quantization networks (Rastegari et al. 2016) and do not quantize the weights and activation in the first and the last layers. See Appendix for more details of our implementation.

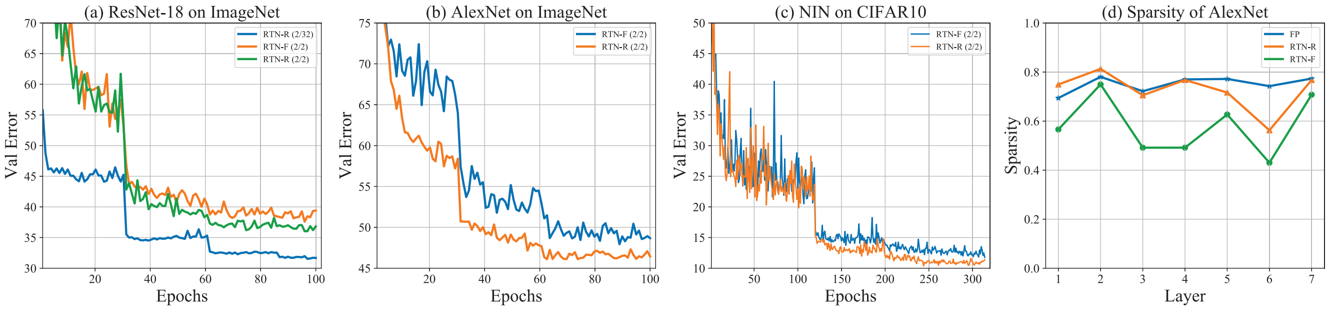

Figure 5: The validation error rate plots during training with/without reparameterization and the sparsity comparison of each layer in AlexnNet.

Initialization could be vitally important for quantization neural networks. We first train a full-precision model from scratch and initialize the RTN by minimizing the Euclidean distance between quantized and full precision weights like TWN (Li, Zhang, and Liu 2016). For example, the initial γ is set to $\mathbb{E}_{|\boldsymbol{A}|>0.5}(|\boldsymbol{A}|)$ and β is set to 0.

4.2 Results on ImageNet

The validation error on ImageNet is plotted in Figure 5. We can see that RTN-R has a lower error rate than RTN-F. Especially, AlexNet plot, Figure 5(b), shows that RTN-R has a relatively smooth curve and better convergence speed and may be the result of automatic adjustment of learning rate via gamma in Equation 7.

The overall results on ImageNet are shown in Table 2 with several strong extremely low-bit models. Note that we swap the order of BN and ReLU, so we report the full precision models' accuracy as a reference and compare the degradation from full precision models (the last column in the table). We first compare our RTN with models that only quantize weights like BWN, TWN, and TTQ. Our RTN not only achieves state-of-the-art accuracy but also has the smallest gap between full precision models. In addition, compared with TTQ's asymmetric quantization, our RTN uses symmetric quantization which is naturally harder for training but more friendly for hardware implementation.

Quantizing the activation is more challenging compared with weights (Cai et al. 2017), and there is still a large margin between full precision models and extremely low-bit models. We compare several models that quantize both weights and activation with our proposed model (denoted by RTN-R). For the ablation study, we also report the performance of our ternary network with fixed ternary activation (denoted by RTN-F) to show the effectiveness of scale and offset.

According to Table 2, we can conclude that, **1)** RTN-R outperforms almost every models. So, though it is a trade-off between the number of bits and accuracy, the ternary network finds a better balance between them. **2)** In spite of PACT has comparable performance, especially on AlexNet, our RTN-R also shares a small gap with full precision models. Note that RTN is, furthermore, better for hardware implementation on mobile and embedded devices. **3)** With learnable scale factors and offset, RTN-R has higher accuracy than the RTN-F, which validates the improvement of representation ability from our reparameterization design.

Sparsity Comparison According to our analysis in Section 3.4, the offset β can adjust the sparsity of z automatically. Generally, changing of sparsity concurs with observation in (Wang et al. 2018), in which they believe the optimal sparsity is slightly higher than 50% based on the foundation of attention mechanism. Figure 5 (d) shows the sparsity comparison between RTN-F, RTN-R, and full precision models. Our reparameterized ternary activation can adjust the sparsity automatically, and the sparsity of RTN-R is close to FP. Compared with (Wang et al. 2018), Our RTN can adjust sparsity automatically without any manual settings.

Analysis of Reparameterization We report the value of scale γ and offset β for activation and the mean value of scale $\bar{\alpha}$ for weights of each layer in ResNet-18 (See Appendix). We can see that the activation distribution has changed a lot among layers. This means that each layer learns its optimal range and magnitude thus increasing the representational ability. Interestingly, we found that the activation and weights in the downsample residual layer only change slightly. This situation may result from the special 1×1 filter in this layer.

4.3 Results on CIFAR10

For CIFAR10, we mainly compare our method with XNOR-Net on NIN. We use the PyTorch implementations of XNOR-Net (Rastegari et al. 2016)[‡]. See more implementations details in Appendix.

Results for NIN on CIFAR10 can be found in Table 2. Our RTN almost reboots full precision accuracy (only 0.2% absolute gap) without bells and whistles. This performance may result from the scale and offset that significantly changes the range of ternary activation and weights.

Ablation Study There are two learnable parameters in the reparameterized ternary activation, the scale factor, and the offset factor. We evaluate the effect of these two parameters by only applying one of them in the RTN. We denote

[‡] https://github.com/jiecaoyu/XNOR-Net-PyTorch

Table 2: Overall comparison of various extremely low-bit quantized models on ImageNet and CIFAR10. We compare top-1 accuracy and degradation from full precision for a fair comparison. T Denotes network uses ternary values instead of quaternary for 2 bits representation; ‡ denotes results of ResNet-18B where the filter number in each block is 1.5×.

ResNet-18 (ImageNet)				
Methods	# bits(W/A)	FP ref.	Accuracy	Degrad.
BWN	1 / 32	69.3	60.8	8.5
TWN	T / 32	69.3	61.8	7.5
TWN‡	T / 32	69.3	65.3	4.0
TTQ‡	T / 32	69.6	66.6	3.0
RTN-R	T / 32	69.2	**68.5**	**0.7**
XNOR	1 / 1	69.3	51.2	18.1
Bi-Real	1 / 1	68.0	56.4	11.6
TBN	1 / T	69.3	55.6	13.7
DoReFa	1 / 2	70.2	53.4	16.8
HWGQ	1 / 2	69.6	56.1	13.5
HORQ	2 / 2	69.3	55.9	13.4
DoReFa	2 / 2	70.2	62.6	7.6
PACT	2 / 2	70.2	64.4	5.8
RTN-F	T / T	69.2	62.4	6.8
RTN-R	T / T	69.2	**64.5**	**4.7**
AlexNet (ImageNet)				
XNOR	1 / 1	56.6	44.2	12.4
TBN	1 / T	57.2	49.7	7.5
DoReFa	1 / 2	55.9	49.8	6.1
HWGQ	1 / 2	55.7	50.5	5.2
PACT	2 / 2	57.2	**55.0**	**2.2**
RTN-F	T / T	58.7	52.6	6.1
RTN-R	T / T	58.7	53.9	4.8
MobileNet (ImageNet)				
PACT	2 / 2	69.9	56.1	13.8
RTN-R	T / T	69.9	**56.9**	**13.0**
NIN (CIFAR10)				
XNOR	1 / 1	89.8	86.4	3.4
RTN-F	T / T	89.8	88.2	1.6
RTN-S	T / T	89.8	88.5	1.3
RTN-O	T / T	89.8	89.1	0.7
RTN-R	T / T	89.8	**89.6**	**0.2**

the RTN-S as the activation with the scale factor and RTN-O as the activation with the offset only. Implementation on CIFAR10 is kept the same as before.

The results are shown in Table 2. Apparently, when we only add the scale factor, the improvement can be trivial. This is because the ReLU does not impact the scale of the activation (i.e. $\phi(\gamma \boldsymbol{A}) = \gamma\phi(\boldsymbol{A})$), and BN can eliminate the effect of the scale factor. We refer to this effect as *scale invariance of activation*. However, according to Equation 10, the offset factor can change the sparsity threshold in ReLU, thus greatly affect the activation. Therefore, RTN-O has higher performance than RTN-S. Note that in our RTN-R, there is no *scale invariance of activation* when we apply scale and offset factors together, which can both change the distribution of activation.

Table 3: Hardware performances of the circuits for the vector multiplication operations shown in Figure 4.

Circuits	Power consumption	Area
2-bit vector multiplication with ternary values	**22.17**uW	**199.43**um^2
2-bit vector multiplication with quaternary values	76.09uW	831.62um^2
vector multiplication with floating-point (32 bits) values	1.03mW	17783um^2

4.4 Hardware Implementation

We compare the hardware performances of the two circuits for vector multiplication operation shown in Figure 4. We further implement the circuit for floating-point values (32 bits) vector multiplication. We synthesize our design with Xilinx Vivado Design Suite (viv) and use Xilinx VC707 FPGA evaluation board for power measurement. For the comparison on circuit area and computation latency, we utilize the Synopsys Design Compiler (syn) with 45nm NanGate Open Cell Library (nan).

As shown in the Table 3, the circuit for ternary values (Figure 4(a)) outperforms that for the 2-bit values (quaternary values) (Figure 4(b)) and floating-point values in terms of both power (3.43×, 46.46×) and area $(4.17\times, 89.17\times)$. These differences result from the fact that more adders and bitwise shifters are used by the circuit for quaternary value multiplication. From Table 3, we notice that the circuit for quaternary value multiplication is $4\times$ larger than that of ternary value multiplication. That is to say, for a fixed size of circuit area and a settled clock frequency, the circuit for ternary value multiplication has $4\times$ less latency than the circuit for quaternary value multiplication, since we can make four ternary value multiplier works in parallel. Moreover, our circuit can be easily deployed as a building block of any large-scale parallel computing framework such as systolic array (Kung 1982) for efficient matrix multiplication.

5 Conclusion

In this paper, we propose the reparameterized ternary network with ternary weights and activation. The learnable reparameterizers are demonstrated to considerably increase the expressiveness of fixed ternary values. According to our analysis and empirical results, scale and offset are able to adjust the range of quantized value, inflect sparsity of activation and accelerate training. To support efficient computing in RTN, a novel computation pattern is proposed.

Acknowledgements This work is supported by the National Research Foundation, Prime Ministers Office, Singapore under its National Cybersecurity RD Programme (No. NRF2016NCR-NCR002-020), and FY2017 SUG Grant.

References

Baskin, C.; Liss, N.; Chai, Y.; Zheltonozhskii, E.; Schwartz, E.; Girayes, R.; Mendelson, A.; and Bronstein, A. M. 2018. Nice: Noise injection and clamping estimation for neural network quantization. *arXiv preprint arXiv:1810.00162*.

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

Cai, Z.; He, X.; Sun, J.; and Vasconcelos, N. 2017. Deep learning with low precision by half-wave gaussian quantization. In *CVPR*.

Choi, J.; Wang, Z.; Venkataramani, S.; Chuang, P. I.-J.; Srinivasan, V.; and Gopalakrishnan, K. 2018. Pact: Parameterized clipping activation for quantized neural networks. In *arXiv:1805.06085*.

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

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

Deng, L.; Jiao, P.; Pei, J.; Wu, Z.; and Li, G. 2018. Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. In *Neural Networks*.

Gong, R.; Liu, X.; Jiang, S.; Li, T.; Hu, P.; Lin, J.; Yu, F.; and Yan, J. 2019. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 4852–4861.

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

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

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*.

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*.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NeurIPS*.

Kung, H. T. 1982. Why systolic architectures? In *IEEE Computer*.

Li, Z.; Ni, B.; Zhang, W.; Yang, X.; and Gao, W. 2017a. Performance guaranteed network acceleration via high-order residual quantization. In *CVPR*.

Li, Z.; Ni, B.; Zhang, W.; Yang, X.; and Gao, W. 2017b. Performance guaranteed network acceleration via high-order residual quantization. In *Proceedings of the IEEE International Conference on Computer Vision*, 2584–2592.

Li, F.; Zhang, B.; and Liu, B. 2016. Ternary weight networks. In *arXiv:1605.04711*.

Lin, M.; Chen, Q.; and Yan, S. 2013. Network in network. In *arXiv:1312.4400*.

Liu, Z.; Wu, B.; Luo, W.; Yang, X.; Liu, W.; and Cheng, K.-T. 2018. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *ECCV*.

Nair, V., and Hinton, G. E. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML*.

Nangate freepdk45 open cell library. http://www.nangate.com/?page_id=2325.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*.

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. In *IJCV*.

Salimans, T., and Kingma, D. P. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NeurIPS*.

Design compiler: Rtl synthesis. https://www.synopsys.com/support/training/rtl-synthesis/design-compiler-rtl-synthesis.html.

Vivado design suite - hlx editions productivity. multiplied. https://www.xilinx.com/products/design-tools/vivado.html.

Wan, D.; Shen, F.; Liu, L.; Zhu, F.; Qin, J.; Shao, L.; and Tao Shen, H. 2018. Tbn: Convolutional neural network with ternary inputs and binary weights. In *ECCV*.

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

Zhang, X.; Zou, J.; He, K.; and Sun, J. 2015. Accelerating very deep convolutional networks for classification and detection. In *PAMI*.

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. In *arXiv:1606.06160*.

Zhu, C.; Han, S.; Mao, H.; and Dally, W. J. 2016. Trained ternary quantization. In *arXiv:1612.01064*.