Adaptive Loss-aware Quantization for Multi-bit Networks

Zhongnan Qu[1], Zimu Zhou[2], Yun Cheng[1], and Lothar Thiele[1]

[1]Computer Engineering Group, ETH Zurich, Switzerland

`{quz, chengyu, thiele}@ethz.ch`

[2]School of Information Systems, Singapore Management University, Singapore

`zimuzhou@smu.edu.sg`

Abstract

We investigate the compression of deep neural networks by quantizing their weights and activations into multiple binary bases, known as multi-bit networks (MBNs), which accelerate the inference and reduce the storage for the deployment on low-resource mobile and embedded platforms. We propose Adaptive Loss-aware Quantization (ALQ), a new MBN quantization pipeline that is able to achieve an average bitwidth below one-bit without notable loss in inference accuracy. Unlike previous MBN quantization solutions that train a quantizer by minimizing the error to reconstruct full precision weights, ALQ directly minimizes the quantization-induced error on the loss function involving neither gradient approximation nor full precision maintenance. ALQ also exploits strategies including adaptive bitwidth, smooth bitwidth reduction, and iterative trained quantization to allow a smaller network size without loss in accuracy. Experiment results on popular image datasets show that ALQ outperforms state-of-the-art compressed networks in terms of both storage and accuracy.

1. Introduction

There is a growing interest to deploy deep neural networks on resource-constrained devices to enable new intelligent services such as mobile assistants, augmented reality, and autonomous cars. However, deep neural networks are notoriously resource-intensive. Their complexity needs to be trimmed down to fit in mobile and embedded devices.

To take advantage of the various pretrained models for efficient inference on resource-constrained devices, it is common to compress the pretrained models via pruning [10], quantization [8, 9, 26, 42, 43], distillation [12], among others. We focus on quantization, especially quantizing both the full precision weights and activations of a deep neural

network into binary encodes and the corresponding scaling factors [4, 36], which are also interpreted as binary basis vectors and floating-point coordinates in a geometry viewpoint [9]. Neural networks quantized with binary encodes replace expensive floating-point operations by bitwise operations, which are supported even by microprocessors and often result in small memory footprints [29]. Since the space spanned by only one-bit binary basis and one coordinate is too sparse to optimize, many researchers suggest a multi-bit network (MBN) [8, 9, 15, 26, 42, 43], which allows to obtain a small size without notable accuracy loss and still leverages bitwise operations. An MBN is usually obtained via trained quantization. Recent studies [31] leverage bit-packing and bitwise computations for efficient deploying binary networks on a wide range of general devices, which also provides more flexibility to design multi-bit/binary networks.

Most MBN quantization schemes [8, 9, 15, 26, 42, 43] predetermine a global bitwidth, and learn a quantizer to transform the full precision parameters into binary bases and coordinates such that the quantized models do not incur a significant accuracy loss. However, these approaches have the following drawbacks:

- A global bitwidth may be sub-optimal. Recent studies on fixed-point quantization [18, 25] show that the optimal bitwidth varies across layers.

- Previous efforts [26, 42, 43] retain inference accuracy by minimizing the weight reconstruction error rather than the loss function. Such an indirect optimization objective may lead to a notable loss in accuracy. Furthermore, they rely on approximated gradients, *e.g.* straight-through estimators (STE) to propagate gradients through quantization functions during training.

- Many quantization schemes [36, 43] keep the first and last layer in full precision, because quantizing these layers to low bitwidth tends to dramatically decrease

the inference accuracy [41, 28]. However, these two full precision layers can be a significant storage overhead compared to other low-bit layers (see Sec. 5.4.3). Also, floating-point operations in both layers can take up the majority of computation in quantized networks [27].

We overcome the above drawbacks via a novel *A*daptive *L*oss-aware *Q*uantization scheme (ALQ). Instead of using a uniform bitwidth, ALQ assigns a different bitwidth to each group of weights. More importantly, ALQ directly minimizes the loss function w.r.t. the quantized weights, by iteratively learning a quantizer that *(i)* smoothly reduces the number of binary bases and *(ii)* alternatively optimizes the remaining binary bases and the corresponding coordinates. Although loss-aware quantization has been proposed for binary and ternary networks [14, 13, 46], they are inapplicable to MBNs due to the extended optimization space. They also need approximated gradients during training. ALQ is the first loss-aware quantization scheme for MBNs and eliminates the need for approximating gradients and retaining full precision weights. ALQ is also able to quantize the first and last layers without incurring a notable accuracy loss. The main contributions of this work are as follows.

- We design ALQ, the first loss-aware quantization scheme for multi-bit networks. It is also the first trained quantizer without gradient approximation, and realizes an adaptive bitwidth w.r.t the loss for MBNs (including the first and last layers).

- ALQ enables extremely low-bit (yet dense tensor form) binary networks with an average bitwidth below 1-bit. Experiments on CIFAR10 show that ALQ can compress VGG to an average bitwidth of 0.4-bit, while yielding a higher accuracy than other binary networks [36, 4].

2. Related Work

ALQ follows the trend to quantize deep neural networks using discrete bases to reduce expensive floating-point operations. Commonly used bases include fixed-point [47], power of two [16, 45], and $\{-1, 0, +1\}$ [4, 36]. We focus on quantization with binary bases *i.e.* $\{-1, +1\}$ among others for the following considerations. *(i)* If both weights and activations are quantized with the same binary basis, it is possible to evaluate 32 multiply-accumulate operations (MACs) with only 3 instructions on a 32-bit microprocessor, *i.e.* bitwise `xnor`, `popcount`, and accumulation. This will significantly speed up the convolution operations [16]. *(ii)* A network quantized to fixed-point requires specialized integer arithmetic units (with various bitwidth) for efficient computing [1, 18], whereas a network quantized with multiple binary bases adopts the same operations mentioned before as binary networks. Popular networks quantized with binary bases include *Binary Networks* and *Multi-bit Networks*.

2.1. Quantization for Binary Networks

BNN [4] is the first network with both binarized weights and activations. It dramatically reduces the memory and computation but often with notable accuracy loss. To resume the accuracy degradation from binarization, XNOR-Net [36] introduces a layer-wise full precision scaling factor into BNN. However, XNOR-Net leaves the first and last layers unquantized, which consumes more memory. SYQ [6] studies the efficiency of different structures during binarization/ternarization. LAB [14] is the first loss-aware quantization scheme which optimizes the weights by directly minimizing the loss function.

ALQ is inspired by recent loss-aware binary networks such as LAB [14]. Loss-aware quantization has also been extended to fixed-point networks in [13]. However, existing loss-aware quantization schemes [14, 13] are inapplicable for MBNs. This is because multiple binary bases dramatically extend the optimization space with the same bitwidth (*i.e.* an optimal set of binary bases rather than a single basis), which may be intractable. Some proposals [14, 13, 46] still require full-precision weights and gradient approximation (backward STE and forward loss-aware projection), introducing undesirable errors when minimizing the loss. In contrast, ALQ is free from gradient approximation.

2.2. Quantization for Multi-bit Networks

MBNs denote networks that use multiple binary bases to trade off storage and accuracy. Gong *et al*. propose a residual quantization process, which greedily searches the next binary basis by minimizing the residual reconstruction error [8]. Guo *et al*. improve the greedy search with a least square refinement [9]. Xu *et al*. [42] separate this search into two alternating steps, fixing coordinates then exhausted searching for optimal bases, and fixing the bases then refining the coordinates using the method in [9]. LQ-Net [43] extends the scheme of [42] with a moving average updating, which jointly quantizes weights and activations. However, similar to XNOR-Net [36], LQ-Net [43] does not quantize the first and last layers. ABC-Net [26] leverages the statistical information of all weights to construct the binary bases as a whole for all layers.

All the state-of-the-art MBN quantization schemes minimize the weight reconstruction error rather than the loss function of the network. They also rely on the gradient approximation such as STE when back propagating the quantization function. In addition, they all predetermine a uniform bitwidth for all parameters. The indirect objective, the approximated gradient, and the global bitwidth lead to a sub-optimal quantization. ALQ is the first scheme to explicitly optimize the loss function and incrementally train an adaptive bitwidth while without gradient approximation.

3. Adaptive Loss-Aware Quantization

3.1. Weight Quantization Overview

Notations. We aim at MBN quantization with an adaptive bitwidth. To allow adaptive bitwidth, we structure the weights in *disjoint groups*. Specifically, for the vectorized weights $\boldsymbol{w}$ of a given layer l, where $\boldsymbol{w} \in \mathbb{R}^{N \times 1}$, we divide $\boldsymbol{w}$ into G disjoint groups. For simplicity, we omit the subscript l. Each group of weights is denoted by $\boldsymbol{w}_g$, where $\boldsymbol{w}_g \in \mathbb{R}^{n \times 1}$ and $N = n \times G$. Then the quantized weights of each group, $\hat{\boldsymbol{w}}_g = \sum_{i=1}^{I_g} \alpha_i \boldsymbol{\beta}_i = \boldsymbol{B}_g \boldsymbol{\alpha}_g$. $\boldsymbol{\beta}_i \in \{-1, +1\}^{n \times 1}$ and $\alpha_i \in \mathbb{R}_+$ are the ith binary basis and the corresponding coordinate; I_g represents the bitwidth, *i.e.* the number of binary bases, of group g. $\boldsymbol{B}_g \in \{-1, +1\}^{n \times I_g}$ and $\boldsymbol{\alpha}_g \in \mathbb{R}_+^{I_g \times 1}$ are the matrix forms of the binary bases and the coordinates. We further denote $\boldsymbol{\alpha}$ as vectorized coordinates $\{\boldsymbol{\alpha}_g\}_{g=1}^G$, and $\boldsymbol{B}$ as concatenated binary bases $\{\boldsymbol{B}_g\}_{g=1}^G$ of all weight groups in layer l. A layer l quantized as above yields an average bitwidth $I = \frac{1}{G} \sum_{g=1}^G I_g$. We discuss the choice of group size n, and the initial $\boldsymbol{B}_g, \boldsymbol{\alpha}_g, I_g$ in Sec. 5.1.

Problem Formulation. ALQ quantizes weights by directly minimizing the loss function rather than the reconstruction error. For layer l, the process can be formulated as the following optimization problem.

$$\min_{\hat{\boldsymbol{w}}_g} \quad \ell\left(\hat{\boldsymbol{w}}_g\right) \tag{1}$$

$$\text{s.t.} \quad \hat{\boldsymbol{w}}_g = \sum_{i=1}^{I_g} \alpha_i \boldsymbol{\beta}_i = \boldsymbol{B}_g \boldsymbol{\alpha}_g \tag{2}$$

$$\text{card}(\boldsymbol{\alpha}) = I \times G \leq \mathrm{I}_{\min} \times G \tag{3}$$

where ℓ is the loss; $\text{card}(.)$ denotes the cardinality of the set, *i.e.* the total number of elements in $\boldsymbol{\alpha}$; $\mathrm{I}_{\min}$ is the desirable average bitwidth. Since the group size n is the same in one layer, $\text{card}(\boldsymbol{\alpha})$ is proportional to the storage consumption.

ALQ tries to solve the optimization problem in Eq.(1)-Eq.(3) by *iteratively* solving two sub-problems as below. The overall pseudocode is illustrated in Alg. 5 in Appendix B.3.

- **Step 1: Pruning in $\boldsymbol{\alpha}$ Domain** (Sec. 3.2). In this step, we progressively reduce the average bitwidth I for a layer l by pruning the least important (w.r.t. the loss) coordinates in $\boldsymbol{\alpha}$ domain. Note that removing an element α_i will also lead to the removal of the binary basis $\boldsymbol{\beta}_i$, which in effect results in a smaller bitwidth I_g for group g. This way, no sparse tensor is introduced. Sparse tensors could lead to a detrimental irregular computation. Since the importance of each weight group differs, the resulting I_g varies across groups, and thus contributes to an adaptive bitwidth I_g for each group. In this step, we only set some elements of $\boldsymbol{\alpha}$ to zero (also remove them from $\boldsymbol{\alpha}$ leading to a reduced I_g) without changing

the others. The optimization problem for Step 1 is:

$$\min_{\boldsymbol{\alpha}} \quad \ell\left(\boldsymbol{\alpha}\right) \tag{4}$$

$$\text{s.t.} \quad \text{card}(\boldsymbol{\alpha}) \leq \mathrm{I}_{\min} \times G \tag{5}$$

- **Step 2: Optimizing Binary Bases $\boldsymbol{B}_g$ and Coordinates $\boldsymbol{\alpha}_g$** (Sec. 3.3). In this step, we retrain the remaining binary bases and coordinates to recover the accuracy degradation induced by the bitwidth reduction. Similar to [42], we take an alternative approach for better accuracy recovery. Specifically, we first search for a new set of binary bases w.r.t. the loss given fixed coordinates. Then we optimize the coordinates by fixing the binary bases. The optimization problem for Step 2 is:

$$\min_{\hat{\boldsymbol{w}}_g} \quad \ell\left(\hat{\boldsymbol{w}}_g\right) \tag{6}$$

$$\text{s.t.} \quad \hat{\boldsymbol{w}}_g = \sum_{i=1}^{I_g} \alpha_i \boldsymbol{\beta}_i = \boldsymbol{B}_g \boldsymbol{\alpha}_g \tag{7}$$

Optimizer Framework. We consider both sub-problems above as an optimization problem with *domain constraints*, and solve them using the same optimization framework: subgradient methods with projection update [5].

The optimization problem in Eq.(6)-Eq.(7) imposes domain constraints on $\boldsymbol{B}_g$ because they can only be discrete binary bases. The optimization problem in Eq.(4)-Eq.(5) can be considered as with a trivial domain constraint: the output $\boldsymbol{\alpha}$ should be a subset (subvector) of the input $\boldsymbol{\alpha}$. Furthermore, the feasible sets for both $\boldsymbol{B}_g$ and $\boldsymbol{\alpha}$ are bounded.

Subgradient methods with projection update are effective to solve problems in the form of $\min_{\boldsymbol{x}}(\ell(\boldsymbol{x}))$ s.t. $\boldsymbol{x} \in \mathbb{X}$ [5]. We apply AMSGrad [37], an adaptive stochastic subgradient method with projection update, as the common optimizer framework in the two steps. At iteration s, AMSGrad generates the next update as,

$$
\begin{aligned}
\boldsymbol{x}^{s+1} &= \Pi_{\mathbb{X}, \sqrt{\hat{\boldsymbol{V}}^s}}(\boldsymbol{x}^s - a^s \boldsymbol{m}^s / \sqrt{\hat{\boldsymbol{v}}^s}) \\
&= \underset{\boldsymbol{x} \in \mathbb{X}}{\operatorname{argmin}} \|(\sqrt{\hat{\boldsymbol{V}}^s})^{1/2}(\boldsymbol{x} - (\boldsymbol{x}^s - \frac{a^s \boldsymbol{m}^s}{\sqrt{\hat{\boldsymbol{v}}^s}}))\|
\end{aligned} \tag{8}
$$

where Π is a projection operator; $\mathbb{X}$ is the feasible domain of $\boldsymbol{x}$; a^s is the learning rate; $\boldsymbol{m}^s$ is the (unbiased) first momentum; $\hat{\boldsymbol{v}}^s$ is the (unbiased) maximum second momentum; and $\hat{\boldsymbol{V}}^s$ is the diagonal matrix of $\hat{\boldsymbol{v}}^s$.

In our context, Eq.(8) can be written as,

$$\hat{\boldsymbol{w}}_g^{s+1} = \underset{\hat{\boldsymbol{w}}_g \in \mathbb{F}}{\operatorname{argmin}} f^s(\hat{\boldsymbol{w}}_g) \tag{9}$$

$$f^s = (a^s \boldsymbol{m}^s)^{\mathrm{T}}(\hat{\boldsymbol{w}}_g - \hat{\boldsymbol{w}}_g^s) + \frac{1}{2}(\hat{\boldsymbol{w}}_g - \hat{\boldsymbol{w}}_g^s)^{\mathrm{T}} \sqrt{\hat{\boldsymbol{V}}^s}(\hat{\boldsymbol{w}}_g - \hat{\boldsymbol{w}}_g^s) \tag{10}$$

where $\mathbb{F}$ is the feasible domain of $\hat{\boldsymbol{w}}_g$.

Step 1 and Step 2 have different feasible domains of $\mathbb{F}$ according to their objective (details in Sec. 3.2 and Sec. 3.3). Eq.(10) approximates the loss increment incurred by $\hat{\boldsymbol{w}}_g$ around the current point $\hat{\boldsymbol{w}}_g^s$ as a quadratic model function under domain constraints [5, 37]. For simplicity, we replace $a^s m^s$ with $\boldsymbol{g}^s$ and replace $\sqrt{\hat{V}^s}$ with $\boldsymbol{H}^s$. $\boldsymbol{g}^s$ and $\boldsymbol{H}^s$ are iteratively updated by the loss gradient of $\hat{\boldsymbol{w}}_g^s$. Thus, the required input of each AMSGrad step is $\frac{\partial \ell^s}{\partial \hat{\boldsymbol{w}}_g^s}$. Since $\hat{\boldsymbol{w}}_g^s$ is used as an intermediate value during the forward, it can be directly obtained during the backward.

3.2. Pruning in α Domain

As introduced in Sec. 3.1, we reduce the bitwidth I by pruning the elements in $\boldsymbol{\alpha}$ w.r.t. the resulting loss. If one element α_i in $\boldsymbol{\alpha}$ is pruned, the corresponding dimension $\boldsymbol{\beta}_i$ is also removed from $\boldsymbol{B}$. Now we explain how to instantiate the optimizer in Eq.(9) to solve Eq.(4)-Eq.(5) of Step 1.

The cardinality of the chosen subset (*i.e.* the average bitwidth) is uniformly reduced over iterations. For example, assume there are T iterations in total, the initial average bitwidth is I^0 and the desired average bitwidth after T iterations I^T is $I_{\min}$. Then at each iteration t, $(M_p = \mathrm{round}((I^0 - I_{\min}) \times G/T))$ of α_i^t's are pruned in this layer. This way, the cardinality after T iterations will be smaller than $I_{\min} \times G$. See Alg. 2 in Appendix B.1 for the pseudocode.

When pruning in the $\boldsymbol{\alpha}$ domain, $\boldsymbol{B}$ is considered as invariant. Hence Eq.(9) and Eq.(10) become,

$$\boldsymbol{\alpha}^{t+1} = \underset{\boldsymbol{\alpha} \in \mathbb{P}}{\mathrm{argmin}} \; f_{\boldsymbol{\alpha}}^t(\boldsymbol{\alpha}) \qquad (11)$$

$$f_{\boldsymbol{\alpha}}^t = (\boldsymbol{g}_{\boldsymbol{\alpha}}^t)^{\mathrm{T}}(\boldsymbol{\alpha} - \boldsymbol{\alpha}^t) + \frac{1}{2}(\boldsymbol{\alpha} - \boldsymbol{\alpha}^t)^{\mathrm{T}} \boldsymbol{H}_{\boldsymbol{\alpha}}^t(\boldsymbol{\alpha} - \boldsymbol{\alpha}^t) \quad (12)$$

where $\boldsymbol{g}_{\boldsymbol{\alpha}}^t$ and $\boldsymbol{H}_{\boldsymbol{\alpha}}^t$ are similar as in Eq.(10) but are in the $\boldsymbol{\alpha}$ domain. If α_i^t is pruned, the ith element in $\boldsymbol{\alpha}$ is set to 0 in the above Eq.(11) and Eq.(12). Thus, the constrained domain $\mathbb{P}$ is taken as all possible vectors with M_p zero elements in $\boldsymbol{\alpha}^t$.

AMSGrad uses a diagonal matrix of $\boldsymbol{H}_{\boldsymbol{\alpha}}^t$ in the quadratic model function, which decouples each element in $\boldsymbol{\alpha}^t$. This means the loss increment caused by several α_i^t equals the sum of the increments caused by them individually, which are calculated as,

$$f_{\boldsymbol{\alpha},i}^t = -g_{\boldsymbol{\alpha},i}^t \, \alpha_i^t + \frac{1}{2} \, H_{\boldsymbol{\alpha},ii}^t \, (\alpha_i^t)^2 \qquad (13)$$

All items of $f_{\boldsymbol{\alpha},i}^t$ are sorted in ascending. Then the first M_p items (α_i^t) in the sorted list are removed from $\boldsymbol{\alpha}^t$, and results in a smaller cardinality $I^t \times G$. The input of the AMSGrad step in $\boldsymbol{\alpha}$ domain is the loss gradient of $\boldsymbol{\alpha}_g^t$, which can be computed with the chain rule,

$$\frac{\partial \ell^t}{\partial \boldsymbol{\alpha}_g^t} = \boldsymbol{B}_g^{t\,\mathrm{T}} \frac{\partial \ell^t}{\partial \hat{\boldsymbol{w}}_g^t} \qquad (14)$$

$$\hat{\boldsymbol{w}}_g^t = \boldsymbol{B}_g^t \boldsymbol{\alpha}_g^t \qquad (15)$$

Our pipeline allows to reduce the bitwidth smoothly, since the average bitwidth can be floating-point. In ALQ, since different layers have a similar group size (see Sec. 5.1), the loss increment caused by pruning is sorted among all layers, such that only a global pruning number needs to be determined. The global pruning number is defined by the total number of pruned α_i's, *i.e.* the difference of $\sum_l \mathrm{card}(\boldsymbol{\alpha}_l)$ before and after pruning. More details are explained in Appendix B.1 and B.3. This pruning step not only provides a loss-aware adaptive bitwidth, but also seeks a better initialization for training the following lower bitwidth quantization, since quantized weights may be relatively far from their original full precision values.

3.3. Optimizing Binary Bases and Coordinates

After pruning, the loss degradation needs to be recovered. Following Eq.(9), the objective in Step 2 is

$$\hat{\boldsymbol{w}}_g^{s+1} = \underset{\hat{\boldsymbol{w}}_g \in \mathbb{F}}{\mathrm{argmin}} \; f^s(\hat{\boldsymbol{w}}_g) \qquad (16)$$

The constrained domain $\mathbb{F}$ is decided by both binary bases and full precision coordinates. Hence directly searching optimal $\hat{\boldsymbol{w}}_g$ is NP-hard. Instead, we optimize $\boldsymbol{B}_g$ and $\boldsymbol{\alpha}_g$ in an alternative manner, as with prior MBN quantization w.r.t. the reconstruction error [42, 43].

Optimizing $\boldsymbol{B}_g$. We directly search the optimal bases with AMSGrad. In each optimizing iteration q, we fix $\boldsymbol{\alpha}_g^q$, and update $\boldsymbol{B}_g^q$. We find the optimal increment for each group of weights, such that it converts to a new set of binary bases, $\boldsymbol{B}_g^{q+1}$. This optimization step searches a new space spanned by $\boldsymbol{B}_g^{q+1}$ based on the loss reduction, which prevents the pruned space to be always a subspace of the previous one. See Alg. 3 in Appendix B.2.1 for the detailed pseudocode.

According to Eq.(9) and Eq.(10), the optimal $\boldsymbol{B}_g$ w.r.t. the loss is updated by,

$$\boldsymbol{B}_g^{q+1} = \underset{\boldsymbol{B}_g \in \{-1,+1\}^{n \times I_g}}{\mathrm{argmin}} \; f^q(\boldsymbol{B}_g) \qquad (17)$$

$$\begin{aligned} f^q = &\, (\boldsymbol{g}^q)^{\mathrm{T}}(\boldsymbol{B}_g \boldsymbol{\alpha}_g^q - \hat{\boldsymbol{w}}_g^q) + \\ &\, \frac{1}{2}(\boldsymbol{B}_g \boldsymbol{\alpha}_g^q - \hat{\boldsymbol{w}}_g^q)^{\mathrm{T}} \boldsymbol{H}^q(\boldsymbol{B}_g \boldsymbol{\alpha}_g^q - \hat{\boldsymbol{w}}_g^q) \end{aligned} \qquad (18)$$

where $\hat{\boldsymbol{w}}_g^q = \boldsymbol{B}_g^q \boldsymbol{\alpha}_g^q$.

Since $\boldsymbol{H}^q$ is diagonal in AMSGrad, each row vector in $\boldsymbol{B}_g^{q+1}$ can be independently determined. For example, the jth row is computed as,

$$\boldsymbol{B}_{g,j}^{q+1} = \underset{\boldsymbol{B}_{g,j}}{\mathrm{argmin}} \; \|\boldsymbol{B}_{g,j} \boldsymbol{\alpha}_g^q - (\hat{w}_{g,j}^q - g_j^q / H_{jj}^q)\| \quad (19)$$

In general, $n >> I_g$. For each group, we firstly compute all 2^{I_g} possible values of

$$\boldsymbol{b}^{\mathrm{T}} \boldsymbol{\alpha}_g^q , \quad \boldsymbol{b}^{\mathrm{T}} \in \{-1,+1\}^{1 \times I_g} \qquad (20)$$

Then each row vector $B_{g,j}^{q+1}$ can be directly assigned by the optimal b^{T} through exhaustive search.

Optimizing α_g. The above obtained set of binary bases B_g spans a new linear space. The current α_g is unlikely to be a (local) optimal w.r.t. the loss in this space, so now we optimize α_g. Since α_g is full precision, *i.e.* $\alpha_g \in \mathbb{R}^{I_g \times 1}$, there is no domain constraint and thus no need for projection updating. Optimizing full precision w_g takes incremental steps in original n-dim full space (spanned by orthonormal bases). Similarly, optimizing α_g searches steps in a I_g-dim subspace (spanned by B_g). Hence conventional training strategies can be directly used to optimize α_g. See Alg. 4 in Appendix B.2.2 for the detailed pseudocode.

Similar as Eq.(11) and Eq.(12), we construct an AMS-Grad optimizer in α domain but without projection updating, for each group in the pth iteration as,

$$\alpha_g^{p+1} = \alpha_g^p - a_\alpha^p m_\alpha^p / \sqrt{\hat{v}_\alpha^p} \qquad (21)$$

We also add an L2-norm regularization on α_g to enforce unimportant coordinates to zero. If there is a negative value in α_g, the corresponding basis is set to its negative complement, to keep α_g semi-positive definite. Optimizing B_g and α_g does not influence the number of binary bases I_g.

Optimization Speedup. Since α_g is full precision, updating α_g^q is much cheaper than exhaustively search B_g^{q+1}. Even if the main purpose of the first step in Sec. 3.3 is optimizing bases, we also add an updating process for α_g^q in each optimizing iteration q.

We fix B_g^{q+1}, and update α_g^q. The overall increment of quantized weights from both updating processes is,

$$\hat{w}_g^{q+1} - \hat{w}_g^q = B_g^{q+1}\alpha_g^{q+1} - B_g^q\alpha_g^q \qquad (22)$$

Substituting Eq.(22) into Eq.(9) and Eq.(10), we have,

$$\begin{aligned} \alpha_g^{q+1} = &-((B_g^{q+1})^{\mathrm{T}}H^qB_g^{q+1})^{-1}\times \\ &((B_g^{q+1})^{\mathrm{T}}(g^q - H^qB_g^q\alpha_g^q)) \end{aligned} \qquad (23)$$

To ensure the inverse in Eq.(23) exists, we add $\lambda\mathbf{I}$ onto $(B_g^{q+1})^{\mathrm{T}}H^qB_g^{q+1}$, where $\lambda = 10^{-6}$.

4. Activation Quantization

To leverage bitwise operations for speedup, the inputs of each layer (*i.e.* the activation output of the last layer) also need to be quantized into the multi-bit form. Unlike previous works [43] that quantize activations with a different binary basis ($\{0, +1\}$) as weights, we also quantize activations with $\{-1, +1\}$. This way, we only need 3 instructions rather than 5 instructions to replace the original 32 MACs (see Sec. 2).

Our activation quantization follows the idea proposed in [2], *i.e.* a parameterized clipping for fixed-point activation

quantization, but it is adapted to the multi-bit form. Specially, we replace ReLu with a step activation function. The vectorized activation x of the lth layer is quantized as,

$$x \doteq \hat{x} = x_{ref} + D\gamma = D'\gamma' \qquad (24)$$

where $D \in \{-1, +1\}^{N_x \times I_x}$, and $\gamma \in \mathbb{R}_+^{I_x \times 1}$. γ' is a column vector formed by $[x_{ref}, \gamma^{\mathrm{T}}]^{\mathrm{T}}$; D' is a matrix formed by $[\mathbf{1}^{N_x \times 1}, D]$. N_x is the dimension of x, and I_x is the quantization bitwidth for activations. x_{ref} is the introduced layerwise (positive floating-point) reference to fit in the output range of ReLu. During inference, x_{ref} is convoluted with the weights of the next layer and added to the bias. Hence the introduction of x_{ref} does not lead to extra computations. The output of the last layer is not quantized, as it does not involve computations anymore. For other settings, we directly adopt them used in [43]. γ and x_{ref} are updated during the forward propagation with a running average to minimize the squared reconstruction error as,

$$\gamma'_{new} = (D'^{\mathrm{T}}D')^{-1}D'^{\mathrm{T}}x \qquad (25)$$

$$\gamma' = 0.9\gamma' + (1 - 0.9)\gamma'_{new} \qquad (26)$$

The (quantized) weights are also further fine-tuned with our optimizer to resume the accuracy drop. Here, we only set a global bitwidth for all layers in activation quantization.

5. Experiments

We implement ALQ with Pytorch [30], and evaluate its performance on MNIST [22], CIFAR10 [19], and ILSVRC12 (ImageNet) [38] using LeNet5 [21], VGG [14, 36], and ResNet18/34 [11], respectively. More implementation details are provided in Appendix C.

5.1. ALQ Initialization

We adapt the network sketching proposed in [9] for $\hat{w}_g$ initialization, and realize a structured sketching (see Alg. 1 in Appendix A.1). Some important parameters in Alg. 1 are chosen as below.

Group Size n. We empirically decide a range for the group size n by trading off between the weight reconstruction error and the storage compression rate. A group size from 32 to 512 achieves a good balance. Accordingly, for a convolution layer, grouping in channel-wise ($w_{c,:,:,:}$), kernel-wise ($w_{c,d,:,:}$), and pixel-wise ($w_{c,:,h,w}$) appears to be appropriate. Channel-wise $w_{c,:}$ and subchannel-wise $w_{c,d:d+n}$ grouping are suited for a fully connected layer. In addition, the most frequently used structures for current popular networks are pixel-wise (convolution layers) and (sub)channel-wise (fully connected layers), which align with the bit-packing approach in [31]. See Appendix A.2 for more details on grouping.

Maximum Bitwidth $I_{\max}$ for Group g. The initial I_g is set by a predefined initial reconstruction precision or a maximum bitwidth. We notice that the accuracy degradation

caused by the initialization can be fully recovered after several optimization epochs proposed in Sec. 3.3, if the maximum bitwidth is 8. For example, ResNet18 on ILSVRC12 after such an initialization can be retrained to a Top-1/5 accuracy of 70.3%/89.4%, even higher than its full precision counterpart (69.8%/89.1%). For smaller networks, *e.g.* VGG on CIFAR10, a maximum bitwidth of 6 is sufficient.

5.2. Convergence Analysis

Settings. This experiment conducts the ablation study of our optimization step in Sec. 3.3. We show the advantages of our optimizer in terms of convergence, on networks quantized with a uniform bitwidth. Optimizing $\boldsymbol{B}_g$ with speedup (also Alg. 3) is compared, since it takes a similar alternating step as previous works [42, 43]. Recall that our optimizer *(i)* has no gradient approximation and *(ii)* directly minimizes the loss. We use AMSGrad[1] with a learning rate of 0.001, and compare with following baselines.

- *STE with rec. error:* This baseline quantizes the maintained full precision weights by minimizing the reconstruction error (rather than the loss) during forward and approximates gradients via STE during backward. This approach is adopted in some of the best-performing quantization schemes such as LQ-Net [43].

- *STE with loss-aware:* This baseline approximates gradients via STE but performs a loss-aware projection updating (adapted from our ALQ). It can be considered as a multi-bit extension of prior loss-aware quantizers for binary and ternary networks [14, 13]. See Alg. 6 in Appendix B.4 for the detailed pseudocode.
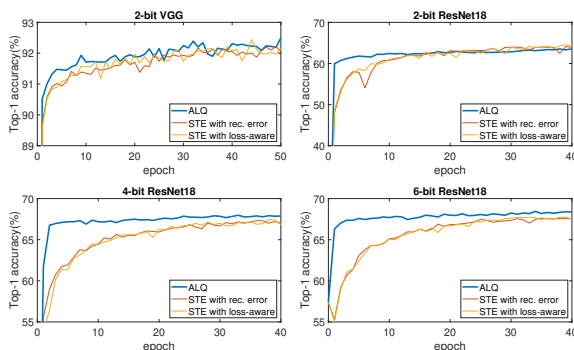

Figure 1. Validation accuracy trained with ALQ/baselines.

Results. Fig. 1 shows the Top-1 validation accuracy of different optimizers, with increasing epochs on uniform bitwidth MBNs. ALQ exhibits not only a more stable and faster convergence, but also a higher accuracy. The exception is 2-bit

---

[1]AMSGrad can also optimize full precision parameters.

ResNet18. ALQ converges faster, but the validation accuracy trained with STE gradually exceeds ALQ after about 20 epochs. For training a large network with ≤ 2 bitwidth, the positive effect brought from the high precision trace may compensate certain negative effects caused by gradient approximation. In this case, keeping full precision parameters will help calibrate some aggressive steps of quantization, resulting in a slow oscillating convergence to a better local optimum. This also encourages us to add several epochs of STE based optimization (*e.g. STE with loss-aware*) after low bitwidth quantization to further regain the accuracy.

5.3. Effectiveness of Adaptive Bitwidth

Settings. This experiment demonstrates the performance of incrementally trained adaptive bitwidth in ALQ, *i.e.* our pruning step in Sec. 3.2. Uniform bitwidth quantization (an equal bitwidth allocation across all groups in all layers) is taken as the baseline. The baseline is trained with the same number of epochs as the sum of all epochs during the bitwidth reduction. Both ALQ and the baseline are trained with the same learning rate decay schedule.

Results. Table 1 shows that there is a large Top-1 accuracy gap between an adaptive bitwidth trained with ALQ and a uniform bitwidth (baseline). In addition to the overall average bitwidth (I_W), we also plot the distribution of the average bitwidth and the number of weights across layers (both models in Table 1) in Fig. 2. Generally, the first several layers and the last layer are more sensitive to the loss, thus require a higher bitwidth. The shortcut layers in ResNet architecture (*e.g.* the 8th, 13rd, 18th layers in ResNet18) also need a higher bitwidth. We think this is due to the fact that the shortcut pass helps the information forward/backward propagate through the blocks. Since the average of adaptive bitwidth can have a decimal part, ALQ can achieve a compression rate with a much higher resolution than a uniform bitwidth, which not only controls a more precise trade-off between storage and accuracy, but also benefits our incremental bitwidth reduction (pruning) scheme.

Table 1. Comparison between Baseline (Uniform Bitwidth) and ALQ (Adaptive Bitwidth)

Method	I_W	Top-1
Baseline VGG (uniform)	1	91.8%
ALQ VGG	**0.66**	**92.0%**
Baseline ResNet18 (uniform)	2	66.2%
ALQ ResNet18	**2.00**	**68.9%**

It is worth noting that both the optimization step and the pruning step in ALQ follow the same metric, *i.e.* the loss increment modeled by a quadratic function, allowing them to work in synergy. We replace the step of optimizing $\boldsymbol{B}_g$ in ALQ with an STE step (with the reconstruction forward, see in Sec. 5.2), and keep other steps unchanged in the pipeline.
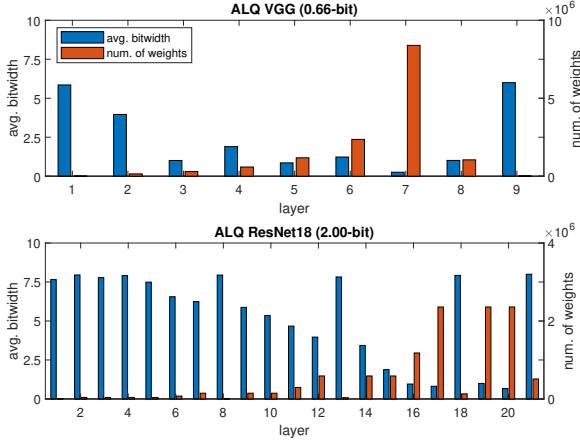
Figure 2. Distribution of the average bitwidth and the number of weights across layers.

When the VGG model is reduced to an average bitwidth of 0.66-bit, the simple combination of an STE step with our pruning step can only reach 90.7% Top-1 accuracy, which is significantly worse than ALQ's 92.0%.

5.4. Comparison with States-of-the-Arts

5.4.1 Non-structured Pruning on MNIST

Settings. Since ALQ can be considered as a (structured) pruning scheme in α domain, we first compare ALQ with two widely used non-structured pruning schemes: Deep Compression (DC) [10] and ADMM-Pruning (ADMM) [44], *i.e.* pruning in the original w domain. For a fair comparison, we implement a modified LeNet5 model as in [10, 44] on MNIST dataset [22] and compare the Top-1 prediction accuracy and the compression rate. Note that the storage consumption only counts the weights, since the weights take the most majority of the storage (even after quantization) in comparison to others, *e.g.* bias, activation quantizer, batch normalization, *etc*. The storage consumption of weights in ALQ includes the look-up-table for the resulting I_g in each group (the same goes for the following experiments).

Table 2. Comparison with State-of-the-Art Non-structured Pruning Methods (LeNet5 on MNIST).

Method	Weights (CR)	Top-1
FP	1720KB (1×)	99.19%
DC [10]	44.0KB (39×)	**99.26%**
ADMM [44]	24.2KB (71×)	99.20%
ALQ	**22.7KB (76×)**	**99.12%**

Results. ALQ shows the highest compression rate (**76×**) while keeping acceptable Top-1 accuracy compared to the two other pruning methods (see Table 2). FP stands for full precision, and the weights in the original full precision LeNet5 consume 1720KB [10]. CR denotes the compression

rate of storing the weights.

It is worth mentioning that both DC [10] and ADMM [44] rely on sparse tensors, which need special libraries or hardwares for execution [24]. Their operands (the shared quantized values) are still floating-point. Hence they hardly utilize bitwise operations for speedup. In contrast, ALQ achieves a higher compression rate without sparse tensors, which is more suited for general off-the-shelf platforms.

The average bitwidth of ALQ is below 1.0-bit (1.0-bit corresponds to a compression rate slightly below 32), indicating some groups are fully removed. In fact, this process leads to a new network architecture containing less output channels of each layer, and thus the corresponding input channels of the next layers can be safely removed. The original configuration $20 - 50 - 500 - 10$ is now $18 - 45 - 231 - 10$.

5.4.2 Binary Networks on CIFAR10

Settings. In this experiment, we compare the performance of ALQ with state-of-the-art binary networks [3, 36, 14]. A binary network is an MBN with the lowest bitwidth, *i.e.* single-bit. Thus, the storage consumption of a binary network can be regarded as the lower bound of a (multi-bit) binary network. For a fair comparison, we implement a small version of VGG from [40] on CIFAR10 dataset [19], as in many state-of-the-art binary networks [3, 14, 36].

Table 3. Comparison with State-of-the-Art Binary Networks (VGG on CIFAR10).

Method	I_W	Weights (CR)	Top-1
FP	32	56.09MB (1×)	92.8%
BC [3]	1	1.75MB (32×)	90.1%
BWN [36]*	1	1.82MB (31×)	90.1%
LAB [14]	1	1.77MB (32×)	89.5%
AQ [18]	0.27	1.60MB (35×)	90.9%
ALQ	**0.66**	**1.29MB (43×)**	**92.0%**
ALQ	**0.40**	**0.82MB (68×)**	**90.9%**

*: both first and last layers are unquantized.

Results. Table 3 shows the performance comparison to popular binary networks. I_W stands for the quantization bitwidth for weights. Since ALQ has an adaptive quantization bitwidth, the reported bitwidth of ALQ is an average bitwidth of all weights. For the statistic information, we plot multiple training loss curves in Appendix C.2.

ALQ allows to compress the network to under 1-bit, which remarkably reduces the storage and computation. ALQ achieves the smallest weight storage and the highest accuracy compared to all weights binarization methods BC [3], BWN [36], LAB [14]. Similar to results on LeNet5, ALQ generates a new network architecture with fewer output channels per layer, which further reduces our models in Table 3 to 1.01MB (0.66-bit) or even 0.62MB (0.40-bit). The computation and the run-time memory can also decrease.

Furthermore, we also compare with AQ [18], the state-of-the-art adaptive fixed-point quantizer. It assigns a different bitwidth for each parameter based on its sensitivity, and also realizes a pruning for 0-bit parameters. Our ALQ not only consumes less storage, but also acquires a higher accuracy than AQ [18]. Besides, the non-standard quantization bitwidth in AQ cannot efficiently run on general hardware due to the irregularity [18], which is not the case for ALQ.

5.4.3 MBNs on ILSVRC12

Settings. We quantize both the weights and the activations of ResNet18/34 [11] with a low bitwidth ($\leq$ 2-bit) on ILSVRC12 dataset [38], and compare our results with state-of-the-art multi-bit networks. The results for the full precision version are provided by Pytorch [30]. We choose ResNet18, as it is a popular model on ILSVRC12 used in the previous quantization schemes. ResNet34 is a deeper network used more in recent quantization papers.

Results. Table 4 shows that ALQ obtains the highest accuracy with the smallest network size on ResNet18/34, in comparison with other weight and weight+activation quantization approaches. I_W and I_A are the quantization bitwidth for weights and activations respectively.

Several schemes (marked with *) are not able to quantize the first and last layers, since quantizing both layers as other layers will cause a huge accuracy degradation [41, 28]. It is worth noting that the first and last layers with floating-point values occupy 2.09MB storage in ResNet18/34, which is still a significant storage consumption on such a low-bit network. We can simply observe this enormous difference between TWN [23] and LQ-Net [43] in Table 4 for example. The evolved floating-point computations in both layers can hardly be accelerated with bitwise operations either.

For reported ALQ models in Table 4, as several layers have already been pruned to an average bitwidth below 1.0-bit (*e.g.* in Fig. 2), we add extra 50 epochs of our *STE with loss-aware* in the end (see in Sec. 5.2). The final accuracy is further boosted (see the results marked with [e]). ALQ can quantize ResNet18/34 with 2.00-bit (across all layers) *without any accuracy loss*. To the best of our knowledge, this is the first time that the 2-bit weight-quantized ResNet18/34 can achieve the accuracy level of its full precision version, even if some prior schemes keep the first and last layers unquantized. These results further demonstrate the high-performance of the pipeline in ALQ.

6. Conclusion

In this paper, we propose a novel loss-aware trained quantizer for multi-bit networks, which realizes an adaptive bitwidth for all layers (w.r.t. the loss). The experiments on current open datasets reveal that ALQ outperforms state-of-the-art multi-bit/binary networks in both accuracy and

Table 4. Comparison with State-of-the-Art Multi-bit Networks (ResNet18/34 on ILSVRC12).

Method	I_W/I_A	Weights	Top-1
ResNet18			
FP [30]	32/32	46.72MB	69.8%
TWN [23]	2/32	2.97MB	61.8%
LR [39]	2/32	4.84MB	63.5%
LQ [43]*	2/32	4.91MB	68.0%
QIL [17]*	2/32	4.88MB	68.1%
INQ [45]	3/32	4.38MB	68.1%
ABC [26]	5/32	7.41MB	68.3%
ALQ	**2.00/32**	**3.44MB**	**68.9%**
ALQ[e]	**2.00/32**	**3.44MB**	**70.0%**
BWN [36]*	1/32	3.50MB	60.8%
LR [39]*	1/32	3.48MB	59.9%
DSQ [7]*	1/32	3.48MB	63.7%
ALQ	**1.01/32**	**1.77MB**	**65.6%**
ALQ[e]	**1.01/32**	**1.77MB**	**67.7%**
LQ [43]*	2/2	4.91MB	64.9%
QIL [17]*	2/2	4.88MB	65.7%
DSQ [7]*	2/2	4.88MB	65.2%
GroupNet [48]*	4/1	7.67MB	66.3%
RQ [27]	4/4	5.93MB	62.5%
ABC [26]	5/5	7.41MB	65.0%
ALQ	**2.00/2**	**3.44MB**	**66.4%**
SYQ [6]*	1/8	3.48MB	62.9%
LQ [43]*	1/2	3.50MB	62.6%
PACT [2]*	1/2	3.48MB	62.9%
ALQ	**1.01/2**	**1.77MB**	**63.2%**
ResNet34			
FP [30]	32/32	87.12MB	73.3%
ALQ[e]	**2.00/32**	**6.37MB**	**73.6%**
ALQ[e]	**1.00/32**	**3.29MB**	**72.5%**
LQ [43]*	2/2	7.47MB	69.8%
QIL [17]*	2/2	7.40MB	70.6%
DSQ [7]*	2/2	7.40MB	70.0%
GroupNet [48]*	5/1	12.71MB	70.5%
ABC [26]	5/5	13.80MB	68.4%
ALQ	**2.00/2**	**6.37MB**	**71.0%**
TBN [41]*	1/2	4.78MB	58.2%
LQ [43]*	1/2	4.78MB	66.6%
ALQ	**1.00/2**	**3.29MB**	**67.4%**

*: both first and last layers are unquantized.
[e]: adding extra epochs of *STE with loss-aware* in the end.

storage. Currently, we are deploying ALQ on a mobile platform to measure the inference efficiency.

Acknowledgement

References

[1] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O'Leary, Roman Genov, and Andreas Moshovos. Bit-pragmatic deep neural network computing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 382–394, 2017.

[2] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. PACT: parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, abs/1805.06085, 2018.

[3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proceedings of Advances in Neural Information Processing Systems*, pages 3123–3131, 2015.

[4] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

[5] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[6] Julian Faraone, Nicholas J. Fraser, Michaela Blott, and Philip Heng Wai Leong. SYQ: learning symmetric quantization for efficient deep neural networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 4300–4309, 2018.

[7] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of International Conference in Computer Vision*, 2019.

[8] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[9] Yiwen Guo, Anbang Yao, Hao Zhao, and Yurong Chen. Network sketching: exploiting binary structure in deep cnns. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 5955–5963, 2017.

[10] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proceedings of International Conference on Learning Representations*, 2016.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[12] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *Proceedings of NIPS Deep Learning Workshop*, 2014.

[13] Lu Hou and James T Kwok. Loss-aware weight quantization of deep networks. In *Proceedings of International Conference on Learning Representations*, 2018.

[14] Lu Hou, Quanming Yao, and James T Kwok. Loss-aware binarization of deep networks. In *Proceedings of International Conference on Learning Representations*, 2017.

[15] Qinghao Hu, Peisong Wang, and Jian Cheng. From hashing to cnns: Training binary weight networks via hashing. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2018.

[16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18(187):1–30, 2017.

[17] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 4350–4359, 2019.

[18] Soroosh Khoram and Jing Li. Adaptive quantization of neural networks. In *Proceedings of International Conference on Learning Representations*, 2018.

[19] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). http://www.cs.toronto.edu/~kriz/cifar.html.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010.

[23] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. In *Proceedings of Advances in Neural Information Processing Systems*, 2016.

[24] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *Proceedings of International Conference on Learning Representations*, 2017.

[25] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *Proceedings of International Conference on Machine Learning*, pages 2849–2858, 2016.

[26] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Proceedings of Advances in Neural Information Processing Systems*, pages 345–353, 2017.

[27] Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. Relaxed quantization for discretized neural networks. In *Proceedings of International Conference on Learning Representations*, 2019.

[28] Asit Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. In *Proceedings of International Conference on Learning Representations*, 2018.

[29] Asit Mishra, Eriko Nurvitadhi, Jeffrey J Cook, and Debbie Marr. WRPN: Wide reduced-precision networks. In *Proceedings of International Conference on Learning Representations*, 2018.

[30] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *Proceedings of NIPS Autodiff Workshop: The Future of Gradient-based Machine Learning Software and Techniques*, 2017.

[31] Fabrizio Pedersoli, George Tzanetakis, and Andrea Tagliasacchi. Espresso: Efficient forward propagation for binary deep neural networks. In *Proceedings of International Conference on Learning Representations*, 2018.

[32] Pytorch. Pytorch example of lenet-5 on mnist. `https://github.com/pytorch/examples/blob/master/mnist/main.py`. Accessed: 2019-09-28.

[33] Pytorch. Pytorch example on cifar10. `https://github.com/kuangliu/pytorch-cifar/blob/master/main.py`. Accessed: 2019-10-08.

[34] Pytorch. Pytorch example on imagenet. `https://github.com/pytorch/examples/blob/master/imagenet/main.py`. Accessed: 2019-09-24.

[35] Pytorch. Pytorch example on resnet. `https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py`. Accessed: 2019-10-15.

[36] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proceedings of European Conference on Computer Vision*, pages 525–542, 2016.

[37] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *Proceedings of International Conference on Learning Representations*, 2018.

[38] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[39] Oran Shayer, Dan Levi, and Ethan Fetaya. Learning discrete weights using the local reparameterization trick. In *Proceedings of International Conference on Learning Representations*, 2018.

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of International Conference on Learning Representations*, 2015.

[41] Diwen Wan, Fumin Shen, Li Liu, Fan Zhu, Jie Qin, Ling Shao, and Heng Tao Shen. Tbn: Convolutional neural network with ternary inputs and binary weights. In *Proceedings of European Conference on Computer Vision*, 2018.

[42] Chen Xu, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha. Alternating multi-bit quantization for recurrent neural networks. In *Proceedings of International Conference on Learning Representations*, 2018.

[43] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of European Conference on Computer Vision*, pages 365–382, 2018.

[44] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of European Conference on Computer Vision*, pages 184–199, 2018.

[45] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. In *Proceedings of International Conference on Learning Representations*, 2017.

[46] Aojun Zhou, Anbang Yao, Kuan Wang, and Yurong Chen. Explicit loss-error-aware quantization for low-bit deep neural networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 9426–9435, 2018.

[47] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[48] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Structured binary neural networks for accurate image classification and semantic segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2019.