Pattern Recognition 139 (2023) 109463

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Bayesian asymmetric quantized neural networks

Jen-Tzung Chien*, Su-Ting Chang

Institute of Electrical and Computer Engineering, National Yang Ming Chiao Tung University, Hsinchu, Taiwan

ARTICLE INFO

Article history:
Received 13 September 2021
Revised 26 November 2022
Accepted 20 February 2023
Available online 25 February 2023

Keywords:
Quantized neural network
Model compression
Binary neural network
Bayesian asymmetric quantization

ABSTRACT

This paper develops a robust model compression for neural networks via parameter quantization. Traditionally, quantized neural networks (QNN) were constructed by binary or ternary weights where the weights were deterministic. This paper generalizes QNN in two directions. First, *M*-ary QNN is developed to adjust the balance between memory storage and model capacity. The *representation values* and the *quantization partitions* in *M*-ary quantization are mutually estimated to enhance the resolution of gradients in neural network training. A flexible quantization with *asymmetric partitions* is formulated. Second, the variational inference is incorporated to implement the Bayesian asymmetric QNN. The *uncertainty* of weights is faithfully represented to enhance the robustness of the trained model in presence of heterogeneous data. Importantly, the *multiple spike-and-slab prior* is proposed to represent the quantization levels in Bayesian asymmetric learning. *M*-ary quantization is then optimized by maximizing the evidence lower bound of classification network. An adaptive parameter space is built to implement Bayesian quantization and neural representation. The experiments on various image recognition tasks show that *M*-ary QNN achieves similar performance as the full-precision neural network (FPNN), but the memory cost and the test time are significantly reduced relative to FPNN. The merit of Bayesian *M*-ary QNN using multiple spike-and-slab prior is investigated.

© 2023 The Author(s). Published by Elsevier Ltd.
This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/)

1. Introduction

Deep neural networks have been successfully developed for a wide range of applications in natural language processing as well as computer vision where model robustness is an essential issue in different regression and classification tasks. Such an issue happens when the variations in collected data are considerable and the mismatches between source and target domains are severe. In addition, deep neural networks suffer from over-parameterization or redundant storage [1,2] which causes extra overhead in edge computation and hardware deployment in end devices. Accordingly, there are two concerns in construction of neural network models. One is the model robustness and the other is the model compression. It is crucial to design a compressive and robust neural network to compensate the variations and mismatches, and simultaneously is adjustable between memory cost and performance loss in system implementation. Model generalization is tackled by Bayesian learning which reflects statistical and systematic uncertainties by characterizing the random parameters of neural network model. The generalization from training to test conditions is pursued. This paper presents a new compression method for neural network where the Bayesian asymmetric quantization is incorporated in training procedure.

1.1. Related work

Bayesian learning has been popularly developed in neural network training [3,4]. Variational inference [5,6] was developed to infer the distribution of latent variables or correspondingly identify the structure of disentangled features from observation data based on the stochastic backpropagation algorithm [7]. On the other hand, model compression aims to minimize the redundancy in learning representation. The compression was handled by using bitstream or bit-width representation [8,9] or using different vector quantization methods including *k*-means quantization [10], product quantization, residual quantization [11], fixed-point quantization [12], and differentiable soft quantization [13]. In [10,14], the sparsity constraint of neural weights was imposed to reduce the network complexity. In [15,16], deep neural network was compressed by means of tensor decomposition of weights. Recently, the quantized neural networks [1,17] were presented to relax the memory and computation requirements while the degradation in classification performance was limited. There were two categories of quantization approaches. One was the vector quantiza-

* Corresponding author.
E-mail address: jtchien@nycu.edu.tw (J.-T. Chien).

https://doi.org/10.1016/j.patcog.2023.109463
0031-3203/© 2023 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/)

tion [11], and the other was the scalar quantization [10]. The k-means clustering was applied by minimizing the quantization error of weights, but the classification loss due to quantization was disregarded. Vector quantization was performed over the parameters in dense connection layers for model compression [18]. Alternatively, the scalar quantization over weights was handled on the basis of the reduced model complexity where either the weight parameters were decomposed or the neurons were dropped by imposing the sparse weights [19]. In [20,21], the quantized neural networks based on binary or ternary weights were proposed. In [22], the layer-wise quantization levels were adopted. Basically, these models followed the original model architecture, but the precision of parameter values was decreased due to the quantization process. There were two factors affecting the resolution in calculation of gradients during neural network quantization and learning. One was the representation levels, and the other was the quantization partitions. Representation values were learned by initializing from full-precision parameters. The quantization loss, measured by these two factors, was minimized. In previous works, the representation values were automatically learned, but the partitions were fine-tuned with a considerable computation overhead. A strong assumption of symmetric partitions was made in [23,24]. A separate estimation of quantizer and classifier was performed [18].

1.2. Main idea of this work

This study develops the Bayesian quantized neural network which handles different constraints in previous works. First, the learning objective is jointly optimized with respect to representation levels as well as quantization partitions. Second, the symmetric partitions is relaxed to truly carry out the data-driven quantization. Third, the number of quantization levels is adaptive to meet different implementation costs in system design. Four, the model uncertainty of neural networks is characterized to assure the robustness to unknown test conditions [19]. Considering these four constraints, a new Bayesian model compression [3] is proposed as the Bayesian quantized neural network. In particular, an *end-to-end* solution is proposed to quantize neural network parameters to different number of levels M where the representation levels and quantization partitions are jointly updated. The *asymmetric partitions* in *M-ary quantization* are flexibly performed to train the quantized model. Accordingly, the memory demand and classification performance are tunable by using this M-ary asymmetric quantization. The losses in quantization and classification are controllable. Over-parameterization can be handled. Overall, there are two ideas proposed in this paper [25]. First, the flexibility in quantization scheme is enhanced. A set of M representation levels are estimated to build a delicate weight space. A general compressive model using M-ary quantized neural network (QNN) is constructed with adjustable number of representation values M. Second, the asymmetric quantization partitions are proposed to replace symmetric partitions in previous work [21]. Asymmetric partitions are learned from data. The traditional way to minimize the quantization error is here changed to maximize the evidence lower bound using a classification network where the weight uncertainties are marginalized. Bayesian asymmetric quantization is implemented to impose stochastic property in QNN. The conjugate prior using multi-spike-and-slab distribution is implemented to reflect the representation levels or the clusters of weight parameters. The resulting posterior is derived as a multi-spike-and-slab distribution to carry out a new Bayesian learning and variational inference by minimizing the Kullback-Leibler divergence between true posterior and parameterized posterior. Bayesian asymmetric learning for quantized neural network is proposed. A series of analytical experiments are evaluated to illustrate the performance in image classification under different model architectures and quantization settings.

2. Binary to M-ary quantized neural networks

It is crucial to deploy high-performance learning machine on the embedded systems where the storage and computation resources are limited in hardware implementation. Previously, the issue of model compression was resolved by using different methods including the parameter prediction using low-rank decomposition, the network pruning using sparsity regularization, and the network compression based on quantization scheme. The over-parameterization in deep learning was tackled by using a few weight values which could predict the remaining values. Low-rank decomposition was applied to reduce the number of free parameters. In [10], the parameters in neural networks were randomly pruned and then fine-tuned to fulfill network compression with sparsity constraint. The channel pruning was performed to accelerate the training of convolutional neural networks [14]. In addition, the compressed neural network was implemented via quantization of weight parameters. In [24], the Hessian-based k-means clustering was developed as a vector quantizer to minimize the quantization loss for network compression. An alternative solution to model compression was based on scalar quantization where each weight was quantized separately. To this end, the previous methods based on binary neural network (binNN) [1] and ternary neural network (ternNN) [17,23] were successfully proposed for model compression as detailed in what follows.

2.1. Binary and ternary quantized neural networks

In [1], the binary NN (binNN) quantized the weights and activations of a neural model into binary values. The binary function in binNN was defined as

$$\widehat{w} = \text{sign}(w) = \begin{cases} +1, & w \geq 0 \\ -1, & w < 0 \end{cases} \tag{1}$$

where w is the full-precision weight and $\widehat{w}$ is its binarized value. In addition to this deterministic function, the stochastic binary function was also considered [1]. During training stage of binNN, it is crucial to enhance the resolution of gradient calculation so as to approximate the layer-wise full-precision parameters via back-propagation algorithm. However, the binarized weights in a discrete function are nondifferentiable. Such an issue can be relaxed by applying the *straight-through estimator* for the gradient of objective function $\mathcal{L}$ with respect to full-precision weight in a form of [26]

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial \widehat{w}} \cdot 1_{|w| \leq 1}. \tag{2}$$

The indicator function 1 is used to preserve the training stability by imposing the gradient with large weight magnitude, here $|w| > 1$, to zero. The bit-based batch normalization [1] was proposed to accelerate the learning speed on small devices. In [27], the normalization was approximated by using the bit-shift method where the multiplications were almost ignored.

As for the ternary NN (ternNN) [17], the weights were quantized into three levels $\{\lambda_1, 0, \lambda_2\}$. The ternarized weight $\widehat{w}$ in quantization scheme was defined as

$$\widehat{w}_i = \begin{cases} \lambda_2, & w_i > \triangle \\ 0, & |w_i| \leq \triangle \\ \lambda_1, & w_i < -\triangle \end{cases} \tag{3}$$

where $\triangle$ was the *predefined* quantization interval or threshold which was symmetric at $-\triangle$ and $\triangle$. This ternNN was seen as sparse neural network since the weights in partition $|w_i| \leq \triangle$ were

forced to be zero. By pruning the sparse weights and setting $\{\lambda_1 = -1, \lambda_2 = 1\}$ [23], ternNN was reduced to binNN where the acceleration trick was applicable. There were three steps to ternarize the weights in ternNN. The first step was to divide the weights by their maximum value so that the pre-trained full-precision weights were normalized in $[-1, +1]$. The second step was to quantize the normalized weights to $\{-1, 0, +1\}$ in accordance with the partition thresholds $\pm\triangle$. The third step was to convert the quantized weights into λ_1 and λ_2. Overall, the parameters of ternNN consisted of full-precision weights $\{w_i\}$ and representation values $\{\lambda_1, \lambda_2\}$. Back-propagation algorithm was implemented to learn both of them using three gradients. The first gradient was calculated with respect to weight via

$$\frac{\partial \mathcal{L}}{\partial w_i} = \begin{cases} \lambda_2 \frac{\partial \mathcal{L}}{\partial \widehat{w}_i}, & w_i > \triangle \\ \frac{\partial \mathcal{L}}{\partial \widehat{w}_i}, & |w_i| \le \triangle \\ \lambda_1 \frac{\partial \mathcal{L}}{\partial \widehat{w}_i}, & w_i < -\triangle. \end{cases} \tag{4}$$

Then, the gradients of learning objective $\mathcal{L}$ with respect to low level λ_1 and high level λ_2 were computed by using the weights w_i located in partition intervals I_1 and I_2 for $w_i < -\triangle$ and $w_i > \triangle$, respectively, in a way of

$$\frac{\partial \mathcal{L}}{\partial \lambda_1} = \sum_{i \in I_1} \frac{\partial \mathcal{L}}{\partial \widehat{w}_i}, \quad \text{where } I_1 = \{i | w_i < -\triangle\} \tag{5}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_2} = \sum_{i \in I_2} \frac{\partial \mathcal{L}}{\partial \widehat{w}_i}, \quad \text{where } I_2 = \{i | w_i > \triangle\}. \tag{6}$$

In [17], the quantization partition was empirically selected as

$$\triangle = 0.05 \max(|w_i|). \tag{7}$$

Unfortunately, the previous quantization methods could not be generalized to handle four levels of parameters $\{\lambda_m\}_{m=1}^4$. A new optimized quantizer is needed. Accordingly, this paper proposes the M-ary quantized neural network. The number of representation levels M in parameters $\lambda = \{\lambda_m\}_{m=1}^M$ is general for model selection. In particular, the partition thresholds $\triangle = \{\triangle_m\}_{m=1}^{M-1}$ are calculated for individual tasks with various network architectures. Fine-tuning $\triangle$ is avoided. The quantization parameters λ and $\triangle$ are estimated together in QNN without empirical tuning.

2.2. M-ary asymmetric quantization

The entire parameters in M-ary QNN contain full-precision weights, representation values and partition thresholds $\{\mathbf{w} = \{w_{ij}\}, \lambda, \triangle\}$. Different layer ℓ involves different parameters via the quantizer $Q^{(\ell)}(\cdot)$ [17]. This study separately quantizes the weights at the same layer ℓ as M representation values. A general scheme based on M-ary asymmetric quantization for any full-precision parameter w_{ij} is driven by

$$\widehat{w}_{ij} = Q^{(\ell)}(w_{ij}) = \begin{cases} \lambda_M^{(\ell)}, & \triangle_{M-1}^{(\ell)} \le w_{ij} < \infty \\ \vdots & \vdots \\ \lambda_2^{(\ell)}, & \triangle_1^{(\ell)} \le w_{ij} < \triangle_2^{(\ell)} \\ \lambda_1^{(\ell)}, & -\infty \le w_{ij} < \triangle_1^{(\ell)} \end{cases} \tag{8}$$

where $\widehat{w}_{ij}$ is the quantized weight for connecting nodes i at layer ℓ with j at layer $\ell + 1$. In training procedure of M-ary QNN, the initialization scheme can be either from scratch or based on the pre-trained weights. The training algorithm is developed through the forward/backward passes.

2.2.1. Forward pass

Forward pass involves two steps before calculating the learning objective $\mathcal{L}$. One is to quantize the weights and the other is to compute the activations. In the first step, M-ary quantization is performed and accompanied with parameter sharing over a group of weights. The quantized weight $\widehat{w}_{ij}$ is obtained by mapping a full-precision weight w_{ij} into a quantized level $1 \le m \le M$ via an indication function $I(w_{ij})$, which is expressed by

$$\widehat{w}_{ij} = \lambda_{I(w_{ij})}^{(\ell)}, \quad \text{where } I(\cdot) : \mathbb{R} \to \{1, \ldots, M\}. \tag{9}$$

The index $I(w_{ij}) = m$ is assigned when w_{ij} is valued between $\triangle_{m-1}$ and $\triangle_m$. The sample weight is then substituted by the corresponding representation as $\widehat{w}_{ij} = \lambda_m$. Using index function is a trick to pass through the gradient with respect to discrete function in back-propagation procedure. M-ary QNN is driven by a new quantizer $Q^{(\ell)}(\cdot)$. In the second step, the activation $a_j^{(\ell+1)}$ at node j in layer $\ell + 1$ is calculated in layer-wise propagation by using activation function $f(\cdot)$ where the full-precision weight is replaced by a representation value as

$$a_j^{(\ell+1)} = f\left(\sum_i Q^{(\ell)}(w_{ij}) a_i^{(\ell)}\right) = f\left(\sum_i \widehat{w}_{ij} a_i^{(\ell)}\right). \tag{10}$$

This nonlinear activation is recursively calculated to find QNN outputs for regression or classification. The loss function $\mathcal{L}(\widehat{\mathbf{y}}, \mathbf{y})$ for supervised network using sum-of-squares error $\frac{1}{2}\sum_n \|\widehat{\mathbf{y}}_n - \mathbf{y}_n\|^2$ or cross-entropy error $-\sum_n \mathbf{y}_n^\top \log \widehat{\mathbf{y}}_n$ between QNN outputs $\widehat{\mathbf{y}} = \{\widehat{\mathbf{y}}_n\}$ and targets $\mathbf{y} = \{\mathbf{y}_n\}$ is measured from N training observations $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$.

2.2.2. Backward pass

In backward pass, we calculate the gradients of loss function $\mathcal{L}(\widehat{\mathbf{y}}, \mathbf{y})$ over individual parameters and use them for parameter updating in a stochastic gradient descent (SGD) algorithm. First, the local gradient of $\mathcal{L}$ over the activation of node i at layer ℓ is computed by

$$\begin{aligned} \widehat{\delta}_i^{(\ell)} &\triangleq \frac{\partial \mathcal{L}}{\partial a_i^{(\ell)}} = \sum_j \frac{\partial \mathcal{L}}{\partial a_j^{(\ell+1)}} \frac{\partial a_j^{(\ell+1)}}{\partial a_i^{(\ell)}} \\ &= \sum_j \delta_j^{(\ell+1)} \lambda_{I(w_{ij})}^{(\ell)} f'\left(\sum_k \lambda_{I(w_{kj})}^{(\ell)} a_k^{(\ell)}\right). \end{aligned} \tag{11}$$

This is an approximate gradient $\widehat{\delta}$ due to quantization. This local gradient is calculated by accumulating those local gradients of nodes j at layer $\ell + 1$. The representation value $\widehat{w}_{ij}$ or $\lambda_{I(w_{ij})}^{(\ell)}$, functioned by quantization partition or indication function $I(w_{ij})$, is introduced. Having the local gradient at each neuron j, the gradients of loss function $\mathcal{L}$ over parameters $\{w_{ij}, \lambda_m, \triangle_m\}$ are computed for parameter updating.

In the implementation. we first update the full-precision weight w_{ij} in full-precision neural network based on the full-precision activation $z_j^{(\ell+1)} = f(\sum_i w_{ij} z_i^{(\ell)})$ and local gradient $\delta_i^{(\ell)} \triangleq \partial \mathcal{L} / \partial z_i^{(\ell)}$ according to

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial z_j^{(\ell+1)}} \frac{\partial z_j^{(\ell+1)}}{\partial w_{ij}} = \delta_j^{(\ell+1)} z_i^{(\ell)} f'\left(\sum_k w_{kj} z_k^{(\ell)}\right). \tag{12}$$

Importantly, M-ary QNN carries out the estimation of both representation value λ_m and quantization partition $\triangle_m$. Notably, the relaxation using straight-through estimator [26] is employed to handle the non-differentiable quantization function. The gradient of loss function over the representation value $\lambda_m = \{\lambda_m^{(l)}\}$ is then derived by

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \lambda_m^{(\ell)}} &= \sum_i \sum_j \frac{\partial \mathcal{L}}{\partial a_j^{(\ell+1)}} \frac{\partial a_j^{(\ell+1)}}{\partial \widehat{w}_{ij}} \frac{\partial \widehat{w}_{ij}}{\partial \lambda_m^{(\ell)}} \\ &= \sum_i \sum_j \widehat{\delta}_j^{(\ell+1)} a_i^{(\ell)} f'\left(\sum_k \lambda_{I(w_{kj})}^{(\ell)} a_k^{(\ell)}\right) \delta_{I(w_{ij})=m} \end{aligned} \tag{13}$$

which is expressed by the Dirac delta function $\delta_{I(w_{ij})=m}$ with value either zero or one. Consequently, the gradient is gathered from those quantized weights $\widehat{w}_{ij}$ connecting layers ℓ and $\ell+1$ which are associated with level m. In addition, the quantization partition $\triangle_m$ is highly affected by the representation value λ_m so that the parameter $\triangle_m$ is here estimated along with the updating of λ_m. In particular, the asymmetric partition $\triangle_m$ at layer ℓ is estimated by averaging two consecutive representation values $\{\lambda_m^{(\ell)}, \lambda_{m-1}^{(\ell)}\}$ as

$$\triangle_m^{(\ell)} = \frac{\lambda_m^{(\ell)} + \lambda_{m-1}^{(\ell)}}{2}. \tag{14}$$

This estimation meets the scalar quantization where the quantization error between the quantized and full-precision weights is minimized. This study presents a general framework of M-ary quantization. The proposed QNN with M=3 differs from ternNN [23] since the quantization partitions $\triangle_m$ are continuously adjusted in learning procedure of representation values λ_m. Joint training of quantization and classification is performed to optimize the performance of downstream classification task.

2.2.3. Training algorithm

Algorithm 1 presents the training procedure for M-ary asymmetric quantized neural network. In forward pass, the input samples $\mathbf{x}$ are fed by using the quantized weights $\widehat{\mathbf{w}}^{(\ell)}$ and full-precision weights $\mathbf{w}^{(\ell)}$ in a recursive and layer-wise way until finding the M-ary QNN outputs $\widehat{\mathbf{y}}$ from the activations $\mathbf{a}^{(L)} = \{a_k^{(L)}\}$ due to quantization in last layer L. The layer-aware quantizer $Q^{(\ell)}(\cdot)$ using parameters $\{\lambda^{(\ell)}, \triangle^{(\ell)}\}$ is applied. $\odot$ denotes the element-wise product. A common error or loss function $\mathcal{L}$ between classification or regression outputs $\widehat{\mathbf{y}}$ and target values $\mathbf{y}$ is computed by using the quantized and full-precision weights $\{\widehat{\mathbf{w}}, \mathbf{w}\}$. Notably, this loss also measures the quantization error caused by the quantizer $Q^{(\ell)}(\cdot)$ in different layers ℓ. Then, the gradients for updating M-ary QNN parameters $\{\mathbf{w}^{(\ell)}, \lambda^{(\ell)}, \triangle^{(\ell)}\}$ in individual layers ℓ are calculated in backward pass by using the activations $\mathbf{a}^{(\ell)}$ and the full precision outputs $\mathbf{z}^{(\ell)}$. SGD algorithm is fulfilled for M-ary QNN. Full-precision weights $\mathbf{w}$ are quantized during neural network training. This algorithm continuously iterates until the convergence is met. In general, threefold novelties are presented in this paper. The first one is to generalize the binary and ternary neural networks [1,17,23] to M-ary quantized neural networks where the compressive model is tunable to meet different specifications in system deployment. The second one is to relax the assumption of symmetric quantization partitions in $\triangle = \{\triangle_m^{(\ell)}\}$. Asymmetric quantization is fulfilled without the need of hyperparameter selection in [23,24]. Third, the asymmetric partitions $\triangle$ and asymmetric levels $\lambda = \{\lambda_m^{(\ell)}\}$ with any M can be jointly estimated to optimize the end performance based on regression or classification loss $\mathcal{L}$ rather than simply minimize the quantization error.

Algorithm 1: Training for M-ary quantized neural network.

Input training minibatches $\mathbf{x} = \{\mathbf{x}_n\}$ and $\mathbf{y} = \{\mathbf{y}_n\}$
Initialize full-precision weights $\mathbf{w} = \{w_{ij}\}$
Initialize quantization parameters $\lambda = \{\lambda_m\}$ and $\triangle = \{\triangle_m\}$ with sorted random number
while $\mathbf{w}, \lambda, \triangle$ *not converged* **do**
  Forward pass:
  $\mathbf{a}^{(0)} = \mathbf{z}^{(0)} = \mathbf{x}$
  for $\ell = 0, \ldots, L-1$ **do**
    $\widehat{\mathbf{w}}^{(\ell)} = Q^{(\ell)}(\mathbf{w}^{(\ell)}, \lambda^{(\ell)}, \triangle^{(\ell)})$
    $\mathbf{a}^{(\ell+1)} = f(\widehat{\mathbf{w}}^{(\ell)} \odot \mathbf{a}^{(\ell)})$
    $\mathbf{z}^{(\ell+1)} = f(\mathbf{w}^{(\ell)} \odot \mathbf{z}^{(\ell)})$
  end
  $\widehat{\mathbf{y}} = \mathbf{a}^{(L)}$
  $\mathcal{L} = \mathcal{L}(\widehat{\mathbf{y}}, \mathbf{y})$
  Backward pass:
  for $\ell = L, \ldots, 1$ **do**
    $\mathbf{w}^{(\ell)} \leftarrow \mathbf{w}^{(\ell)} - \eta \nabla_{\mathbf{w}^{(\ell)}} \mathcal{L}$
    $\lambda^{(\ell)} \leftarrow \lambda^{(\ell)} - \eta \nabla_{\lambda^{(\ell)}} \mathcal{L}$
    $\triangle_m^{(\ell)} = \frac{\lambda_m^{(\ell)} + \lambda_{m-1}^{(\ell)}}{2}$ for $\forall m = 1, \ldots, M-1$
  end
end

3. Bayesian asymmetric quantization

Quantized neural networks are crucial in hardware implementation where the costs of memory and computation can be significantly reduced in test time. However, the estimated M-ary QNN parameters are *point estimates* from training data where the model uncertainty is disregarded and the generalization to test data is bounded. From Bayesian perspective, M-ary QNN can be strengthened by compensating the weight uncertainties due to randomness in heterogeneous environments which likely happen in real applications. This study presents a new Bayesian neural network [4] for robust quantization with *distribution estimation* in M-ary QNN.

3.1. Bayesian neural network

To activate Bayesian learning [28], the distribution of weights $\mathbf{w} = \{w_{ij}\}$ is integrated in estimation of M-ary Bayesian asymmetric quantization parameters $\theta = \lambda$ by using minibatches of training data $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$. For the application of image classification, $\mathbf{x}_n$ is the image sample and $\mathbf{y}_n$ is its one-hot vector for target class. Bayesian asymmetric quantized neural network is implemented in accordance with variational inference. A supervised model is trained to predict the class label $\mathbf{y}$ from input image $\mathbf{x}$ by maximizing the conditional likelihood $p(\mathbf{y}|\mathbf{x})$ or minimizing the negative log likelihood (NLL) or cross-entropy loss function

$$\mathcal{L}_{\text{NLL}}(\widehat{\mathbf{y}}, \mathbf{y}) = -\log p(\mathbf{y}|\mathbf{x}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \log \widehat{y}_{nk} \tag{15}$$

between the neural classification outputs $\widehat{\mathbf{y}} = \{\widehat{y}_{nk}\}$ by using $\mathbf{x}$ and targets $\mathbf{y} = \{y_{nk}\}$. K is the number of classes. To fulfill Bayesian neural network, the marginal likelihood over the weight uncertainty, expressed by prior $p(\mathbf{w})$, is calculated to construct the objective function

$$p(\mathcal{D}) \triangleq p(\mathbf{y}|\mathbf{x}) = \int p_\theta(\mathbf{y}|\mathbf{x}, \mathbf{w}) p(\mathbf{w}) d\mathbf{w}. \tag{16}$$

However, directly maximizing the marginal likelihood is intractable. It is necessary to estimate analytical posterior distribution $p(\mathbf{w}|\mathcal{D})$ by using neural network. An alternative solution to optimize Eq. (16) can be derived by maximizing the variational lower bound of logarithm of conditional likelihood [29], which is yielded by merging with a tractable variational distribution $q_\phi(\mathbf{w})$ with parameter ϕ as [30]

$$L(\theta, \phi) = \mathbb{E}_{q_\phi(\mathbf{w})}[\log p_\theta(\mathbf{y}|\mathbf{x}, \mathbf{w})] - \text{KL}(q_\phi(\mathbf{w}) \| p(\mathbf{w})). \tag{17}$$

It turns out that the expected log likelihood $\log p_\theta(\mathbf{y}|\mathbf{x}, \mathbf{w})$ is calculated by using variational posterior $q_\phi(\mathbf{w})$ as the surrogate of real posterior $p(\mathbf{w}|\mathcal{D})$, and is maximized by using training data $\mathcal{D} = \{\mathbf{x}, \mathbf{y}\}$. A kind of negative cross-entropy error function $-\mathcal{L}_{\text{NLL}}$ is maximized for classification. At the same time, the estimated posterior $q_\phi(\mathbf{w})$ gets close to the prior $p(\mathbf{w})$ by minimizing the Kullback-Leibler (KL) divergence. The evidence lower bound (ELBO) $L(\theta, \phi)$ is maximized to estimate model parameter θ as well as

variational parameter ϕ. In the implementation, the stochastic gradient variational Bayes (SGVB) inference [7] is performed to replace the error backpropagation by maximizing the ELBO. The first term of ELBO is calculated through an unbiased, differentiable and minibatch-based Monte Carlo estimator as

$$\frac{N}{N_b \cdot S} \sum_{n=1}^{N_b} \sum_{s=1}^{S} \log p_\theta(\mathbf{y}_n | \mathbf{x}_n, \{w_{ij}^{(s)} = g_{\phi_{ij}}(\epsilon^{(s)})\}). \tag{18}$$

$\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{N_b}$ denotes the mini-batch samples [30]. The variational distribution of each full-precision weight is assumed to be Gaussian

$$w_{ij}^{(s)} \sim q_\phi(w_{ij}) = \mathcal{N}(\mu_{ij}, \sigma_{ij}^2). \tag{19}$$

The reparameterization trick is applied to find S samples via a differentiable function

$$w_{ij}^{(s)} = g_{\phi_{ij}}(\epsilon^{(s)}), \quad \text{where} \quad \epsilon^{(s)} \sim p(\epsilon) = \mathcal{N}(0,1). \tag{20}$$

Gaussian parameters $\phi = \{\phi_{ij}\} = \{\mu_{ij}, \sigma_{ij}^2\}$ are seen as the variational parameters or distribution estimates for weights $\mathbf{w} = \{w_{ij}\}$. The mini-batch size N_b is insensitive by using Monte-Carlo estimator.

3.2. Probabilistic asymmetric quantization

Traditionally, binNN and ternNN were constrained by using two and three quantization levels, respectively. It is important to extend them to M-ary quantized neural network with arbitrary M [31]. In addition, previous quantization methods in binNN and ternNN are assumed to be deterministic without considering weight uncertainties for learning representation. To tackle these restrictions, this study implements a new Bayesian QNN where the learning objective in Eq. (17) is constructed according to the probabilistic M-ary quantization. A robust and flexible quantization is pursued. Instead of deterministic quantization in Eq. (8), Bayesian model compression adopts the probabilistic quantization. In particular, a probabilistic asymmetric quantizer is introduced to quantize the layer-dependent weights into M quantization values $\lambda^{(\ell)} = \{\lambda_m^{(\ell)}\}_{m=1}^M$. A general scheme to asymmetric M-ary quantization $Q^{(\ell)}(\cdot)$ for a random weight $w_{ij} \sim p(w_{ij}|\mu_{ij}, \sigma_{ij}^2)$ between neurons i and j at layers ℓ and $\ell+1$ is driven by

$$\begin{aligned} \widehat{w}_{ij} = Q^{(\ell)}(w_{ij}) &= \arg \max_{\lambda_m^{(\ell)} \in \lambda^{(\ell)}} p(\lambda_m^{(\ell)} | \mu_{ij}, \sigma_{ij}^2) \\ &= \arg \min_{\lambda_m^{(\ell)} \in \lambda^{(\ell)}} (\lambda_m^{(\ell)} - \mu_{ij})^2 \end{aligned} \tag{21}$$

to find the quantized weight $\widehat{w}_{ij}$. The layer-aware quantizer is considered, but the layer index ℓ is ignored in $\{w_{ij}, \mu_{ij}, \sigma_{ij}^2\}$ for simplification. Importantly, the quantization value $\lambda_m^{(\ell)}$ from $\lambda^{(\ell)} = \{\lambda_m^{(\ell)}\}_{m=1}^M$ giving the highest likelihood is selected. The mean μ_{ij} and variance σ_{ij}^2 in distribution of weight w_{ij} are used to characterize the likelihood. In particular, the approximate posterior is considered to implement the probabilistic quantization as a kind of post-trained quantization scheme. Owing to the Gaussianity in the variational posterior of weight, seeking the highest posterior is equivalent to choosing the closest representation $\lambda_m^{(\ell)}$ with the smallest squared distance to posterior mean μ_{ij}. Although variance σ_{ij}^2 does not really affect the quantization, its contribution is mainly on the Bayesian estimation of $\{\theta, \phi\}$. Bayesian parameters $\phi = \{\mu_{ij}, \sigma_{ij}^2\}$ and quantization parameter $\lambda = \{\lambda_m^{(\ell)}\}$ should be jointly trained in accordance with a unified framework.

3.3. Implementation for learning algorithm

There are a couple of issues we need to tackle to carry out the training procedure of Bayesian quantized neural network (BQNN). The first issue is the sampling of weights while the second issue is the specialized distribution for the quantized weights.

3.3.1. Local reparameterization trick

In calculation of differentiable objective function $L(\theta, \phi)$, it is unstable to directly draw the continuous samples $w_{ij}^{(s)}$ using Gaussian $\mathcal{N}(\mu_{ij}, \sigma_{ij}^2)$. A reparameterization trick was addressed to deal with this problem, i.e. finding the random weights via [32]

$$w_{ij}^{(s)} = g_{\phi_{ij}}(\epsilon^{(s)}) = \mu_{ij} + \sigma_{ij}\epsilon^{(s)} \tag{22}$$

based on S standard Gaussian samples $\{\epsilon^{(s)}\}$. But, this trick is computationally demanding because of high dimensional weights. Alternatively, local reparameterization trick can be applied indirectly over the activation of hidden neuron j at layer $\ell+1$ using

$$z_j^{(s)} = g_{\phi_{ij}}(\zeta^{(s)}) = \gamma_j + \sqrt{\delta_j}\zeta^{(s)}, \quad \text{where} \quad \zeta^{(s)} \sim \mathcal{N}(0,1). \tag{23}$$

and $\{\gamma_j, \delta_j\}$ is the transformation parameters. Considering the calculation $\mathbf{z} = f(W\mathbf{a})$ with activations $\mathbf{z} = \{z_j\}_{j=1}^J$ at layer $\ell+1$, weights $W = \{w_{ij}\}$ and activations $\mathbf{a} = \{a_i\}_{i=1}^I$ at layer ℓ, the stochastic variable $z_j^{(s)}$ is therefore sampled by the variational distribution

$$\begin{aligned} &z_j^{(s)} \sim q_\phi(z_j|\mathbf{a}) = \mathcal{N}(\gamma_j, \delta_j) \\ &\text{where} \quad \gamma_j = \sum_{i=1}^I a_i \mu_{ij} \quad \text{and} \quad \delta_j = \sum_{i=1}^I a_i^2 \sigma_{ij}^2. \end{aligned} \tag{24}$$

Sampling $w_{ij}^{(s)} \sim q_\phi(w_{ij}) = \mathcal{N}(\mu_{ij}, \sigma_{ij}^2)$ is replaced by sampling $z_j^{(s)} \sim q_\phi(z_j|\mathbf{a})$ using Gaussian parameters $\{\gamma_j, \delta_j\}$, which are calculated from the Gaussian parameters of weights $\{\mu_{ij}, \sigma_{ij}^2\}$ and the activations $\{a_i\}$ at layer ℓ. It is noted that the parameter $\phi = \{\mu_{ij}, \sigma_{ij}^2\}$ is still estimated, but the sampling of $\zeta^{(s)}$ for activations $z_j^{(s)}$ can be run rapidly since the number of samples is greatly decreased from $I \cdot J$ for $w_{ij}^{(s)}$ to J for $z_j^{(s)}$. Monte Carlo estimator in Eq. (18) is accordingly calculated by using samples $\{z_j^{(s)}\}$. Training procedure speeds up for Bayesian neural network. In what follows, the prior and posterior distributions are reshaped to reflect the physical meaning of model compression for the quantized weights.

3.3.2. Prior and posterior distributions

BQNN is constructed as the distribution estimate for representation values $\lambda = \{\lambda_m\}$ of layer-dependent weights $\mathbf{w} = \{w_{ij}\}$. The distribution parameter ϕ is inferred to implement Bayesian quantization of weights as the representation values λ. The marginal likelihood with respect to the weights, which are seen as latent variables of BQNN, is calculated to form objective function $p(\mathcal{D})$ in Eq. (16). Maximizing this objective turns out to estimating both model parameter $\theta = \{\lambda_m\}$ and variational parameter $\phi = \{\mu_{ij}, \sigma_{ij}^2\}$. The posteriors of weights $q_\phi(w_{ij})$ are inferred to reveal the *clustering* or *grouping* of weights for M-ary asymmetric quantization. It is noted that the weights have been automatically grouped into M different quantization levels due to the probabilistic asymmetric quantization in Eq. (21). It is interesting that the proposed BQNN does not require M quantization partitions $\triangle = \{\triangle_m^{(\ell)}\}$ for various representation values $\lambda = \{\lambda_m^{(\ell)}\}$.

In the implementation, the prior of weight $p(w_{ij})$ is employed to regularize the estimation of variational posterior $q_\phi(w_{ij})$ because the KL divergence in learning criterion Eq. (17) is minimized. It is crucial to choose a proper prior $p(w_{ij})$, which reflects the concentration in some representation value $\lambda_m^{(\ell)}$. The traditional Bayesian neural network assuming the standard Gaussian as the

prior $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ could not truly reflect such a physical meaning. Gaussian prior is *not* fitted to Bayesian quantized neural network. Therefore, this study adopts the prior $p(w_{ij})$ which introduces sparsity to learn the concentration effect in quantization value. Correspondingly, the learned posterior $q_\phi(w_{ij})$ is closely located at λ_m as a sparse distribution. The sparsity inducing prior is shaped through a scale invariant log-uniform distribution as

$$p(\log|w_{ij}|) \propto \text{const.} \quad \text{or} \quad p(w_{ij}) \propto \frac{1}{|w_{ij}|} \tag{25}$$

which is known as the spike-and-slab distribution [21], namely a sharp peak as the spike and the heavy tails as the slab. On the other hand, the variational posterior of weight w_{ij} is represented by a multiplicative Gaussian noise [33]

$$q_\phi(w_{ij}) = \mathcal{N}(\mu_{ij}, \sigma_{ij}^2) = \mathcal{N}(\mu_{ij}, \beta_{ij}\mu_{ij}^2). \tag{26}$$

This is a sampling distribution $w_{ij}^{(s)} \sim \mathcal{N}(\mu_{ij}, \beta_{ij}\mu_{ij}^2)$ where the variance $\beta_{ij}\mu_{ij}^2$ is formed as a kind of multiplicative noise. The reparameterization trick is then modified as

$$w_{ij}^{(s)} = g_{\phi_{ij}}(\epsilon^{(s)}) = \mu_{ij}(1 + \sqrt{\beta_{ij}}\epsilon^{(s)}) \tag{27}$$

where $\epsilon^{(s)} \sim \mathcal{N}(0, 1)$. The log variational posterior $\log q_\phi(w_{ij})$ can be divided into those terms which are functions of mean μ_{ij} and scale of variance β_{ij}. Owing to the mean-field approximation in variational distribution $q_\phi(\mathbf{w}) = \prod_i \prod_j q_\phi(w_{ij})$, KL term of all weights $\mathbf{w} = \{w_{ij}\}$ in the ELBO of Eq. (17) is approximated by accumulating those terms associated with individual weights at the same layer

$$\text{KL}(q_\phi(\mathbf{w}) \| p(\mathbf{w})) = \sum_i \sum_j \text{KL}(q_\phi(w_{ij}) \| p(w_{ij})). \tag{28}$$

Using Bayesian quantization, the negative KL term in variational inference under the assumed prior in Eq. (25) and the posterior in Eq. (26) is calculated as [34]

$$\begin{aligned} -\text{KL}(q_\phi(w_{ij}) \| p(w_{ij})) &\approx c_1\sigma(c_2 + c_3 \log \beta_{ij}) - 0.5\log(1 + \beta_{ij}^{-1}) \\ &\quad + \text{const.} \triangleq F_{\text{KL}}(\mu_{ij}, \sigma_{ij}) \end{aligned} \tag{29}$$

using $c_1 = 0.63576$, $c_2 = 1.87320$, $c_3 = 1.48695$, and sigmoid function $\sigma(\cdot)$. This KL term is denoted by $F_{\text{KL}}(\cdot)$, and depends on β_{ij} under the relation $\sigma_{ij} = \sqrt{\beta_{ij}}\mu_{ij}$.

3.3.3. Multi-spike-and-slab prior

In [32,35], the prior using Eq. (25) and the posterior using Eq. (26) were assumed to reduce the variance of stochastic gradients in SGVB inference for variational dropout in full-precision neural networks. In [21], the same assumption was made to formulate the ternary neural network where the symmetric representation codebook and symmetric partition interval in Eq. (3) were considered. This study relaxes these constraints and carries out the Bayesian neural network based on M-ary quantization scheme using asymmetric representation levels and partition intervals. However, the prior in Eq. (25) is only specified for one spike. M-ary Bayesian quantization needs a prior which characterizes M spikes at different representation values $\{\lambda_m\}_{m=1}^M$. As a result, the multi-spike-and-slab prior [21] is incorporated to implement the M-ary BQNN. An integrated and shifted spike-and-slab prior is constructed. This prior is shaped from the scaled mixture of Gaussians. The mean of weight prior is represented by summing M shifted and weighted delta functions as

$$p(m_{ij}) = \sum_{m=1}^M \omega_m \delta(m_{ij} - \lambda_m) \tag{30}$$

which are seen as M spikes centered at the locations $\{\lambda_m\}_{m=1}^M$. In this mixture model, a common weighting coefficient $\omega_m = 1/M$ can be used. The distribution of weight w_{ij} is obtained by integrating over a Gaussian $\mathcal{N}(w_{ij}|m_{ij}, \rho_{ij})$ and marginalizing with respect to mean m_{ij} with distribution in Eq. (30) and variance ρ_{ij} with log-uniform distribution

$$p(\rho_{ij}) \propto \frac{1}{|\rho_{ij}|} \tag{31}$$

to find

$$p(w_{ij}) = \int \mathcal{N}(w_{ij}|m_{ij}, \rho_{ij}) p(\rho_{ij}) p(m_{ij}) d\rho_{ij} dm_{ij}. \tag{32}$$

This prior is derived via a scaled mixture of Gaussians as

$$\begin{aligned} p_\theta(w_{ij}) &\propto \sum_{m=1}^M \omega_m \int \frac{1}{|\rho_{ij}|} \mathcal{N}(w_{ij}|\lambda_m, \rho_{ij}) d\rho_{ij} \\ &= \sum_{m=1}^M \omega_m \frac{1}{|w_{ij} - \lambda_m|} \end{aligned} \tag{33}$$

which turns out as a multi-spike-and-slab distribution or equivalently a mixture of shifted and weighted log-uniform distributions. In [21], Bayesian ternNN was setup by assuming M=3, $\lambda_2 = 0$ and $\lambda_3 = -\lambda_1 = c$. This study conducts the asymmetric quantization for location of spikes so that the assumed model can flexibly represent M quantized values. The prior of weight can be rewritten by merging the scaling factors $\{s_m\}_{m=1}^M$ for representation levels, i.e.

$$\lambda_m \leftarrow s_m \lambda_0, \quad \text{for} \quad 1 \le m \le M \tag{34}$$

with an initial value λ_0, in a form of

$$p_\theta(w_{ij}) \propto \sum_{m=1}^M \omega_m \frac{1}{|w_{ij} - s_m\lambda_0|}. \tag{35}$$

Finding the quantization values $\{\lambda_m\}$ is comparable with estimating the scalars $\{s_m\}$. The prior of weight $p_\theta(w_{ij})$ has parameters $\theta = \{\omega_m, s_m\}$. The KL loss for a weight w_{ij} or simply w in the ELBO of Eq. (17) is manipulated by considering the proposed multi-spike-and-slab prior to form

$$\begin{aligned} \mathcal{L}_{\text{KL}}(\phi, \theta) &= \text{KL}(q_\phi(w) \| p_\theta(w)) \\ &= \int q_\phi(w) \log \frac{q_\phi(w)}{\frac{1}{\lambda_0}\sum_{m=1}^M \frac{\omega_m}{\left|\frac{w}{\lambda_0} - s_m\right|}} dw + \text{const.} \\ &= \int q_\phi(\lambda_0 z) \log \frac{q_\phi(\lambda_0 z)}{\frac{1}{\lambda_0}\sum_{m=1}^M \frac{\omega_m}{|z - s_m|}} \lambda_0 \, dz + \text{const.} \\ &= \int q_{\tilde{\phi}}(z) \log \frac{q_{\tilde{\phi}}(z)}{\sum_{m=1}^M \frac{\omega_m}{|z - s_m|}} dz + \text{const.} \\ &= \text{KL}\Big(q_{\tilde{\phi}}(w) \| p(w, s) = \textstyle\sum_{m=1}^M \frac{\omega_m}{|w - s_m|}\Big) + \text{const.} \\ &\approx \tilde{F}_{\text{KL}}\left(\frac{\mu}{\lambda_0}, \frac{\beta}{\lambda_0}, \{s_m\}\right) \end{aligned} \tag{36}$$

where $\phi = \{\mu, \beta\}$, $\tilde{\phi} = \{\mu/\lambda_0, \beta/\lambda_0\}$ and [21]

$$\tilde{F}_{\text{KL}}(\mu, \beta, \{s_m\}) = \sum_{m=1}^M \mathcal{G}(\mu - s_m) F_{\text{KL}}(\mu - s_m, \beta) \tag{37}$$

with $s_m \neq 0$ and

$$\mathcal{G}(\mu) = \exp\left(-\frac{1}{2}\frac{\mu^2}{\tau^2}\right) \tag{38}$$

is a Gaussian window or a scaling function using $\tau = 0.075$. Here, the indices i and j in w_{ij}, μ_{ij}, β_{ij} and σ_{ij} are ignored for simplicity. The KL term $\mathcal{L}_{\text{KL}}$ is approximated to train the asymmetric M-ary BQNN. Meaningfully, the derived KL term is seen as a scaled mixture of M KL terms which reflect M shifted locations corresponding

to M quantization levels. The scaling function $\mathcal{G}(\cdot)$ acts to smooth for the continuity with high weights near M quantization levels $\lambda = \{s_m\lambda_0\}_{m=1}^{M}$. Obviously, the quantization values

$$\lambda_m = s_m\lambda_0 \tag{39}$$

are asymmetric. The probabilistic quantization in Eq. (21) comparably carries out a quantization with asymmetric partition intervals $\triangle_m$ which are excluded in training of Bayesian QNN.

3.3.4. Training algorithm

Training procedure for M-ary BQNN is illustrated in Algorithm 2. The KL loss $\mathcal{L}_{\mathrm{KL}}$ and the cross entropy loss $\mathcal{L}_{\mathrm{NLL}}$

Algorithm 2: Training for Bayesian quantized neural network.

Input training minibatches $\mathbf{x} = \{\mathbf{x}_n\}$ and $\mathbf{y} = \{\mathbf{y}_n\}$
Initialize weight parameters $\phi = \{\mu_{ij}, \beta_{ij}\}$
Initialize quantization parameter $\theta = \{\omega_m, s_m\}$
while ϕ, θ *not converged* **do**
  Forward pass:
  $\mathbf{a}^{(0)} = \mathbf{z}^{(0)} = \mathbf{x}$
  for $\ell = 0, \ldots, L-1$ **do**
    sample $\mathbf{w}^{(\ell)}$ using $q_\phi(\mathbf{w}^{(\ell)})$
    $\widehat{\mathbf{w}}^{(\ell)} = Q^{(\ell)}(\mathbf{w}^{(\ell)}, \mu^{(\ell)})$
    $\mathbf{a}^{(\ell+1)} = f(\widehat{\mathbf{w}}^{(\ell)} \odot \mathbf{a}^{(\ell)})$
    $\mathbf{z}^{(\ell+1)} = f(\mathbf{w}^{(\ell)} \odot \mathbf{z}^{(\ell)})$
    $\mathcal{L}_{\mathrm{KL}}^{(\ell)}(\phi, \theta) = \mathrm{KL}(q_\phi(\mathbf{w}^{(\ell)}) \| p_\theta(\mathbf{w}^{(\ell)}))$
  end
  $\widehat{\mathbf{y}} = \mathbf{a}^{(L)}$
  $\mathcal{L} = \mathcal{L}_{\mathrm{NLL}}(\widehat{\mathbf{y}}, \mathbf{y}) + \sum_\ell \mathcal{L}_{\mathrm{KL}}^{(\ell)}(\phi, \theta)$
  Backward pass:
  for $\ell = L, \ldots, 1$ **do**
    $\phi^{(\ell)} \leftarrow \phi^{(\ell)} - \eta \nabla_{\phi^{(\ell)}} \mathcal{L}$
    $\theta^{(\ell)} \leftarrow \theta^{(\ell)} - \eta \nabla_{\theta^{(\ell)}} \mathcal{L}$
  end
end

are computed along various layers ℓ in forward pass. The negative of ELBO in Eq. (17) is formed by $\mathcal{L} = \mathcal{L}_{\mathrm{NLL}} + \mathcal{L}_{\mathrm{KL}}$ which is minimized to train BQNN. In calculation of $\mathcal{L}_{\mathrm{NLL}}$ in Eq. (15), the random samples between two layers $\{w_{ij}\}$ are drawn from variational posterior $q_\phi(w_{ij})$ using the current distribution estimates of network parameters $\phi = \{\mu_{ij}, \beta_{ij}\}$. These samples $\{w_{ij}\}$ are adopted to integrate the cross-entropy errors $\mathcal{L}_{\mathrm{NLL}}(\widehat{\mathbf{y}}, \mathbf{y})$ using random outputs $\{\widehat{\mathbf{y}}_n\}$. The quantized weights $\widehat{\mathbf{w}}^{(\ell)}$ are obtained via quantizer $Q^{(\ell)}(\cdot)$ to calculate the activations $\mathbf{a}^{(\ell+1)}$ based on activation function $f(\cdot)$. In calculation of $\mathcal{L}_{\mathrm{KL}}$ in Eq. (36), the weight distribution parameters $\phi = \{\mu_{ij}, \beta_{ij}\}$ and quantization parameters $\theta = \{\omega_m, s_m\}$ are both used. SGD algorithm is then performed for updating two parameter sets ϕ and θ in a backward manner. The multi-spike-and-slab prior with spikes $\{\lambda_m\}$ or scalars $\{s_m\}$, and the variational posterior for individual weights $\{w_{ij}\}$ are estimated. Parameter s_m only depends on $\mathcal{L}_{\mathrm{KL}}$ rather than $\mathcal{L}_{\mathrm{NLL}}$ while μ_{ij} and β_{ij} are affected by both $\mathcal{L}_{\mathrm{NLL}}$ and $\mathcal{L}_{\mathrm{KL}}$. In test phase, the quantized weights $\widehat{w}_{ij}$ corresponding to original weights w_{ij} at various layers are calculated by M-ary quantization scheme $Q^{(\ell)}(\cdot)$. The output $\widehat{\mathbf{y}} = \mathbf{a}^{(L)}$ is computed to find classification result.

3.4. Comparison with the related methods

In comparison of memory consumption, each real-valued weight w_{ij} for individual connection between two layers in FPNN using the floating-point representation is stored by 32 bits. Contrarily, each weight in M-ary quantization only spends $\log_2 M$ bits to store M representation levels. The memory costs in bits by using FPNN, QNN and BQNN are compared in Table 1. The memory consumption is measured for test phase while the cost in training phase is disregarded in deployment of real system. Typically, the cost of QNN is measured from the quantized parameters for individual weights $(\log_2 M)N$ as well as the floating-point parameters for individual representation values at different layers $32ML$. In real cases, the number of weights N is significantly larger than the number of levels M times the number of layers L. Number of bits $\log_2 M$ in QNN is significantly smaller than 32 in FPNN for parameter storage. Compared with FPNN, the memory usage of QNN is greatly reduced due to M-ary quantization. The required bits are comparable for QNN and BQNN since only means of weights $\{\mu_{ij}\}$ are stored in test time. This study evaluates the reduction ratio of memory cost by using QNN and BQNN relative to FPNN in different architectures in the following experiments. In case of considering 16 quantization levels (M=16), the size of full-precision weights in network topology 784-256-128-10 is 234,752 while the number of quantized weights or parameters is only 64 (16 levels times 3 hidden layers) in QNN and BQNN. The model size of QNN and BQNN is greatly reduced relative to FPNN.

Table 1
Comparison of memory sizes using FPNN, QNN and BQNN where N is the number of weights, L is the number of layers and M is the number of quantization levels.

Model	Memory in bits
FPNN	$32N$
QNN/BQNN	$(\log_2 M)N + 32ML$

This study presents a substantial extension from the previous compressed neural networks where only binary and ternary weights were considered and the weights were assumed to be deterministic and symmetric. The proposed Bayesian M-ary QNN deals with these constraints. The novelties of this method consist of the flexible M-ary quantization as well as the Bayesian asymmetric learning where the multi-spike-and-slab prior for M-ary weights is proposed in optimization procedure. In [17], the symmetric ternary weights were learned for QNN where Bayesian asymmetric learning for M-ary quantization was neglected. In [33], the sparse Bayesian compression was developed for computational efficiency in deep learning where the single spike-and-slab prior was used without involving parameter quantization. In [36], the local reparameterization trick was developed to learn discrete weights in Bayesian binary and ternary NNs where the M-ary asymmetric quantization for adaptive model compression was disregarded. In [37], the network with learned quantization (LQ-Net) was presented by implementing a multi-bit subspace quantization where the inner products of activations and quantized weights were calculated in a bitwise manner. In [38], the multi-bit symmetric quantizer was implemented via multiple binary codes where the unified quantization and binarization (named by UniQ [38]) were learned. In [13], the differentiable soft quantization (DSQ) was proposed to balance the clipping and rounding errors in the deployment of 2 to 4-bit quantization. LQ-Net, UniQ and DSQ obtained state-of-the-art (SOTA) image recognition accuracy on different tasks and network structures. The step size $\triangle$ was learned in multi-bit quantization [39] instead of estimating the quantization level λ in M-ary quantization. But, Bayesian M-ary quantization and joint optimization for end performance were ignored in three methods. In this study, the generalization to M-ary quantization in BQNN aims to adjust model compression to meet various hardware specifications. In addition to hardware consideration, an alternative view of this work is to treat the number of quantization levels M in QNN or BQNN as a hyperparameter for model comparison. Obviously, this hyperparameter is adjustable to control model

complexity which accordingly balances the tradeoff between memory cost and system accuracy. Hyperparameter M can be selected empirically via cross validation or theoretically through Bayesian optimization. Selecting M involves black-box optimization which requires extra computation to iteratively evaluate end performance over different M. The issue of selecting M is disregarded in QNN and BQNN.

In addition, a deep compressed neural network with the quantized weights using k-means clustering was proposed to implement the other type of M-ary QNN where the quantization levels were also asymmetric or non-uniform [18]. However, the weight quantizer and the neural classifier in this compressed network were separately trained by using different objectives in separate sessions. Different from [18], the proposed QNN and BQNN estimate the quantization levels and the classification parameters *jointly* according to a unified end-performance objective via the cross-entropy loss given in Eq. (15). As shown in Algorithm 1, the quantization levels λ depend on the unquantized weights $\mathbf{w}$ so that both levels and weights are updated through SGD method during training phase. The joint estimation potentially carries out a precise and adaptive neural network for classification. Some other solutions to non-uniform quantization or representation levels were proposed and surveyed in [40] where these previous methods either individually trained the stand-alone quantizer or empirically adopted the rule-based non-uniform quantization. In [41], another type of binary neural network, denoted by Bi-RealN, was implemented by approximating the derivative of sign function and implementing the magnitude-aware gradient over weight. The real-valued activations were preserved. In [42], a variant of binary neural network, denoted by ReActNet, was developed by approximating the real-valued network where the sign and activation functions were generalized. More recently, the post-training quantization (PTQ) was proposed to implement the neural network compression without retraining where the schemes of adaptive rounding (denoted by PTQ-AR) [43] and block reconstruction (denoted by BRECQ [44]) were developed to obtain SOTA results. Different from the previous works, this study presents an asymmetric M-ary QNN through an end-to-end solution to quantization-aware training where the parameters of quantizer and classifier are jointly trained. Notably, the robustness and quantization are connected together by introducing a new Bayesian quantization framework with a specialized prior/posterior setting. Different methods are evaluated in the experiments for comparative study.

4. Experiments

4.1. Experimental setup

This paper presents general approaches to neural network compression which was evaluated by four tasks under various architectures. First, MNIST dataset, consisting of 60K training images and 10K test images from ten handwritten digits, was collected. The gray-scale images of size 28×28 were observed. The convolutional neural network (ConvNet) with LeNet structure was evaluated for the estimated weights. This structure was built by two convolutional layers with pooling cascaded by three fully connected layers (120-84-10). Kernel size 5×5 and stride 1 or 2 were adopted. Second, CIFAR-10 dataset with ten natural objects was adopted [45]. Third, CIFAR-100 dataset with 100 object classes was investigated. The training and test sets in CIFAR-10 and 100 contained 50K and 10K images, respectively. A held-out set of 5K images was used for validation. CIFAR consisted of color images of size 32×32 with three channels. The classification performance using CIFAR-10 and 100 were evaluated under the DenseNet structure with dense blocks as detailed in [46] where the network growth rate was 12 and the total number of layers was 40. There was no data augmentation applied in these three datasets. Fourth, ImageNet dataset contained about 1.28 million training images and 50K validation images from 1000 object classes. Training images were first resized to 256×256, and then randomly cropped to 224×224. At test time, the center 224×224 crop was taken. The evaluation was conducted under two architectures including ResNet-18 and the lightweight neural network called MobileNetV2 where the original settings in [47,48] were adopted. MobilNetV2 was implemented by the inverted residuals and linear bottlenecks. Top-1 and top-5 accuracies were evaluated. All images were preprocessed by normalization using channel means and variances. The proposed QNN and Bayesian QNN (BQNN) with the fixed size of quantization levels M were evaluated. The quantized neural networks using binNN [1] (M=2) and ternNN [17,23] (M=3) were included in the comparison. The proposed BQNN with M=3 was compared with the ternary Bayesian neural network (denoted by ternBNN) in [21]. Local reparameterization trick was applied in variational inference. A key novelty of the proposed QNN with M=3, M=2, and BQNN with M=3 compared with ternNN, binNN, and ternBNN, respectively, was the *flexibility* of *asymmetric* quantization. The proposed methods conducted asymmetric quantization levels as well as partition intervals. The quantization in various methods was all layer-aware or block-aware. The initialization in neural network training from scratch or using pre-trained model was examined. All parameters except λ_0 were initialized by uniform distributions. Parameter λ_0 was empirically set as 0.2. Adam optimizer was used. The learning rate in SGD stayed at 0.005 in the first 50 epochs, then linearly scaled down to 0.001 until the last epoch where a momentum parameter of 0.9 was used. A minibatch size of 256 was used. Batch normalization was applied. Weight decay parameter was set to 1e-5. Different priors $p_\theta(w_{ij})$ in BQNN were assessed. The test classification accuracy of various methods after 80 training epochs was reported. The memory ratios of the required memory using various quantized networks relative to that using FPNN were listed. The recent and competing results using LQ-Net [37], UniQ [38], DSQ [13], PTQ-AR [43] and BRECQ [44] were included for comparison. The deep compressed neural network using k-means clustering (denoted by DCNN with different k or M) was implemented by referring [18] which was seen as a solution to separate training of quantizer and classifier according to the regression error and the classifier error, respectively. The network pruning and Huffman coding in [18] were disregarded. The computation costs of training time (GPU) and test time (CPU) were measured for different methods relative to FPNN. The computation time of running Python codes in PyTorch was based on the hardware using GPU with Tesla P100 PCIe 16GB and CPU with Intel Xeon E5-2620 v4 @2.10GHz where memory of DDR4 128G RAM was used.

4.2. Evaluation for the estimated parameters

In this set of experiments, the estimated weights $\{w_{ij}\}$ and representation levels $\{\lambda_m\}$ in the proposed M-ary quantized neural network (QNN) and the estimated means for weights $\{\mu_{ij}\}$ in the proposed Bayesian QNN (BQNN) are evaluated. Figs. 1 and 2 illustrate the histograms of the estimated weights $\{w_{ij}\}$ (shown in magenta) in the first four layers where the QNNs with M=2 and M=3 are evaluated, respectively. The task of CIFAR-10 with DenseNet architecture is investigated. We find that the estimated weights are affected by the asymmetric quantization. Meaningfully, the estimated weights are closely centered at two and three representation values in QNN with M=2 and M=3, respectively. The corresponding representation levels $\{\lambda_m^{(\ell)}\}$ are displayed by black dashed lines. These weights $\mathbf{w}^{(\ell)}$ are highly related to the estimated $\lambda^{(\ell)}$. The asymmetric parameters in

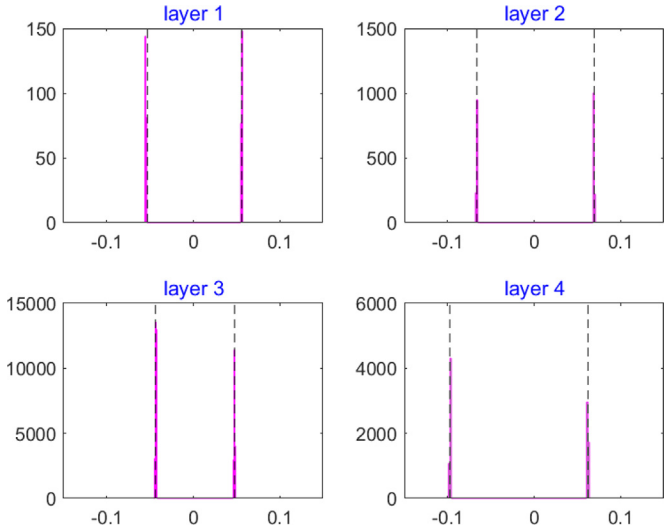

Fig. 1. Histograms of the estimated weights $\{w_{ij}\}$ in various layers ℓ by using QNN with M=2.

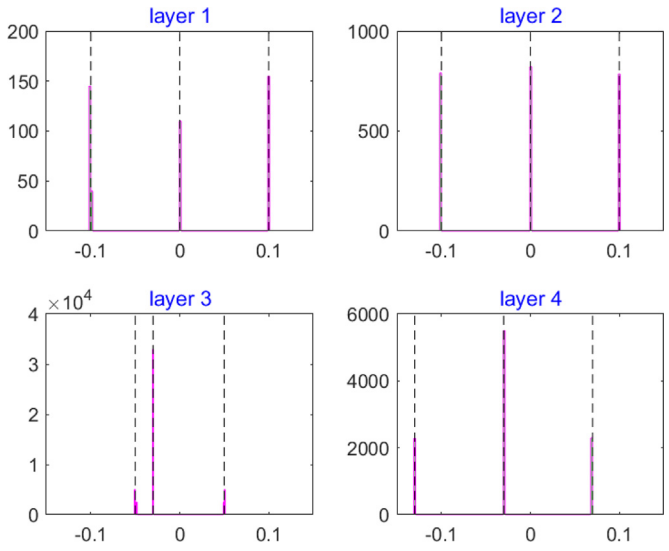

Fig. 2. Histograms of the estimated weights $\{w_{ij}\}$ in various layers ℓ by using QNN with M=3.

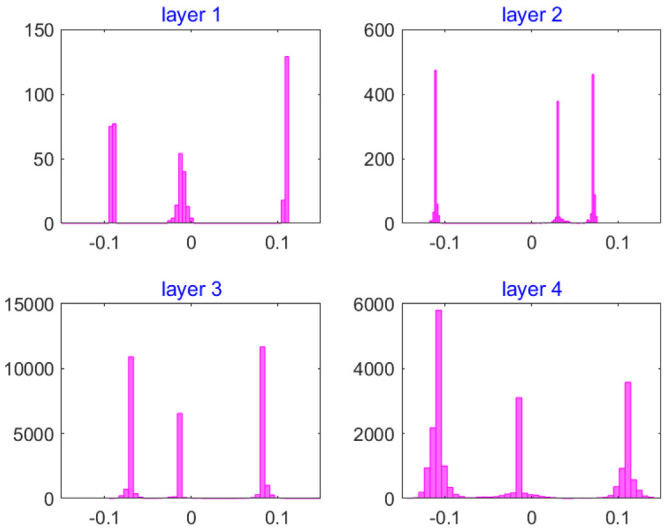

Fig. 3. Histogram of the estimated means $\{\mu_{ij}\}$ of the weights in various layers ℓ by using BQNN with M=3.

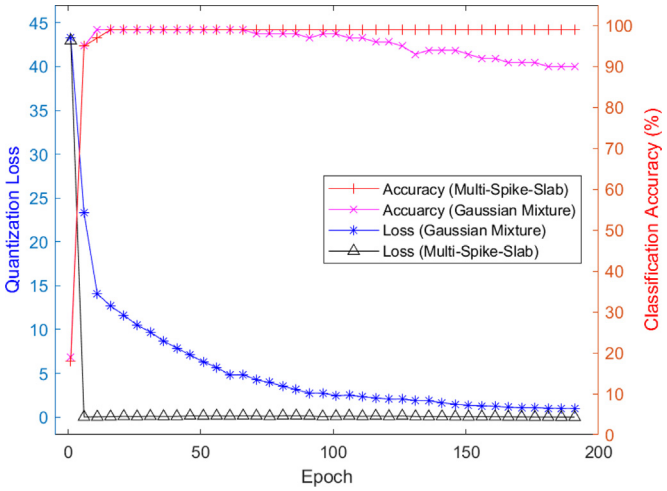

Fig. 4. Quantization loss and classification accuracy by using Gaussian and multi-spike-and-slab priors for weights versus the training epochs in Bayesian quantization.

different layers behave differently. In QNN with M=2, the representation values in layers 1, 2, 3 and 4 are $\{-0.053, 0.056\}$, $\{-0.066, 0.070\}$, $\{-0.044, 0.048\}$ and $\{-0.098, 0.062\}$, respectively. In QNN with M=3, the second bin has the value close to zero in layers 1 and 2. The representation values in layers 3 and 4 are $\{-0.05, -0.03, 0.05\}$, $\{-0.13, -0.03, 0.07\}$, respectively, which are asymmetric. The weights of the second representation level can *not* be ignored. Traditional method using ternNN [17,23] enforced zero value in the second quantization level and imposed symmetric values in the first and third levels, which restricted the flexibility in learning representation. On the other hand, Fig. 3 shows the histograms of the estimated means $\phi = \{\mu_{ij}\}$ of the variational posterior $q_\phi(\mathbf{w})$ at various layers where BQNN is applied. Obviously, these parameters are densely clustered at three positions, which are associated with three spikes $\theta = \{\lambda_m\}_{m=1}^3$ of a multi-spike-and-slab prior $p_\theta(\mathbf{w})$ in different layers. Again, the asymmetric behavior is still observed in the histograms. The distribution estimates using BQNN (Fig. 3) differ from the point estimates using QNN (Figs. 1 and 2). Uncertainty modeling is considered in BQNN.

4.3. Evaluation for prior distributions

Next, the effect of prior distribution of weight parameter $p_\theta(w_{ij})$ is evaluated for variational inference in M-ary quantized neural network. Bayesian QNN is implemented by characterizing M quantization values using the proposed multi-spike-and-slab prior. This prior acts to regularize the estimation of variational posterior $q_\phi(w_{ij})$ which is used to draw weight samples in different layers for Bayesian learning of quantized network. The priors using the multi-spike-and-slab distribution with parameters $\theta = \{\omega_m, s_m\}$ and using the Gaussian mixture distribution with parameters $\theta = \{\omega_m, \lambda_m, r_m\}$ are investigated. The Gaussian means in M mixture components are assigned by the quantization levels $\{\lambda_m\}$. Here, $\{\omega_m\}$ and $\{r_m\}$ denote the mixture weights and Gaussian variances, respectively. The effect of using different priors is evaluated by the metric of quantization loss or the square error due to quantized weights compared with original weights. The classification accuracy using different priors is also examined. Figure 4 shows the quantization loss (left vertical axis) as well as the classification accuracy (right vertical axis) versus the learning epochs (horizontal axis) where two priors in BQNN are em-

ployed in variational inference. Training data in MNIST task are evaluated with M=3. In case of using multi-spike-and-slab prior, the first epoch in BQNN attains loss value about 43 but suddenly dropped to near zero in the next few epochs. However, the decrease of quantization loss in learning procedure using Gaussian mixture prior is slower than that using multi-spike-and-slab prior. The training classification accuracy using multi-spike-and-slab prior is consistently higher than that using Gaussian mixture prior. Over-estimation happens in using Gaussian mixture prior. These results make sure the goodness of using multi-spike-and-slab prior in Bayesian asymmetric learning for M-ary quantized neural network. This prior is accordingly adopted in subsequent experiments.

4.4. Evaluation for classification accuracy

A set of experiments were conducted for comparison of FPNN and different quantization methods including the symmetric quantization in binNN [1], ternNN [17,23] and ternBNN [21], and the asymmetric quantization in the proposed M-ary QNN and BQNN. The effects of M-ary quantization, asymmetric partition and uncertainty modeling were evaluated in terms of classification accuracy. Tables 2 and 3 assess the classification performance and memory cost in presence of MNIST and CIFAR-10, respectively. The memory cost of various quantized networks in terms of reduction ratio relative to FPNN is reported. The averaged accuracy with one standard deviation was calculated across test images. In MNIST task, the effect of initialization in network quantization and training was investigated. The pre-trained model using full-precision parameters from FPNN was adopted. The classification result of different methods was obtained by adopting the same convolutional network architecture for color images by referring [49] where the gray-level images were resized to 256 × 256 and then copied to form color images in the implementation. There were 24 layers and various settings as given in [49]. In CIFAR-10 task, the computation costs in training and test phases were measured by using a quantized model relative to full-precision model. Different architectures were investigated. In this table, the original classification accuracies of FPNN in [23] with VGG-7 architecture [50], FPNN in [46] with DenseNet, binNN in [1] with ConvNet, ternNN in [23] with VGG-7, and ternBNN in [21] with DenseNet in prior literature are shown and cited. Memory cost and computation time are measured by ourselves. The proposed QNN and BQNN are implemented under DenseNet structure from our FPNN which is also included in the comparison. The highest accuracy, the lowest memory consumption and the lowest computation time among different compressed models are shown by bold.

A number of observations are summarized from ablation study over different methods. First, initialization from scratch performs consistently better than that using pre-trained model. The pre-trained parameters do *not* work better than the random weights as the initialization for training QNN as well as binNN or ternNN. Second, a significant reduction in memory cost is found by using various quantized networks. The additional memory storage of QNNs with M=2 and M=3, and BQNN with M=3 compared with binNN, ternNN and ternBNN are bounded, respectively. Third, the classification accuracy is increased or the quantization loss is decreased when the number of quantization clusters k or levels M is increased in DCNN, QNN and BQNN. Four, QNNs with M=2 and M=3 and BQNN with M=3 obtain comparable or even a bit higher accuracy than the competing results of previous binNN in [1], ternNN in [23] and ternBNN in [21], respectively, where different architectures are considered. This implies that asymmetric quantization provides better performance than symmetric quantization. It is attractive to find that QNN with M=4, 8 and BQNN with M=4, 8, 16 even perform better than FPNN. This is possible due to the

Table 2
Comparison of classification accuracy (%) and memory reduction ratio by using various methods where MNIST is used. Initialization conditions (pre-trained & scratch) are examined.

Model	Accuracy (pre-trained)	Accuracy (scratch)	Mem. reduc. ratio
FPNN	–	99.1±0.25	1×
binNN [1]	87.7±0.24	88.8±0.25	**0.0204**×
ternNN [23]	92.1±0.23	92.3±0.24	0.0349×
QNN (M=2)	88.2±0.22	89.6±0.22	0.0209×
QNN (M=3)	94.0±0.22	95.0±0.18	0.0371×
QNN (M=4)	**96.0**±0.20	**97.2**±0.20	0.0502×

Table 3
Comparison of classification accuracy (%), memory reduction ratio, training and test times by using different methods relative to FPNN where CIFAR-10 is used. DCNNs, QNNs, BQNNs under different k or M are investigated.

Model	Classification accuracy	Mem. reduc. ratio	Training time	Test time
FPNN [23]	92.9	–	–	–
FPNN [46]	93.0	–	–	–
FPNN	93.1±0.33 (ours)	1×	1×	1×
binNN [1]	88.6	**0.0295**×	**1.25**×	**0.033**×
ternNN [23]	92.6	0.0396×	1.49×	0.039×
ternBNN [21]	91.7	0.0401×	1.62×	0.048×
DCNN (k=2) [18]	88.5±0.39	0.0305×	1.40×	0.037×
DCNN (k=4) [18]	91.9±0.31	0.0529×	1.61×	0.049×
DCNN (k=16) [18]	92.2±0.35	0.0992×	1.82×	0.094×
QNN (M=2)	89.7±0.34	0.0305×	2.39×	0.037×
QNN (M=3)	92.8±0.32	0.0411×	2.43×	0.048×
QNN (M=4)	93.2±0.34	0.0529×	2.49×	0.049×
QNN (M=8)	93.9±0.31	0.0732×	2.61×	0.070×
QNN (M=16)	93.0±0.32	0.0992×	2.89×	0.095×
BQNN (M=2)	90.8±0.31	0.0310×	2.43×	0.041×
BQNN (M=3)	92.9±0.29	0.0419×	2.53×	0.049×
BQNN (M=4)	93.7±0.29	0.0554×	2.64×	0.053×
BQNN (M=8)	94.0±0.28	0.0797×	2.88×	0.071×
BQNN (M=16)	**94.1**±0.29	0.1099×	3.02×	0.097×

Table 4
Comparison of classification accuracy (%) by using full-precision and compressed models where CIFAR-100 and ImageNet are used. The training and test times of using different methods relative to FPNN are evaluated.

Model	CIFAR-100	ImageNet (Top 1)	ImageNet (Top 5)	Train/Test Times
FPNN [46]	72.5	–	–	–
FPNN [23]	–	65.4	86.8	–
FPNN [17]	–	69.6	89.2	–
FPNN [41]	–	68.0	88.1	–
FPNN [42]	–	72.4	–	–
FPNN	72.3 (ours)	69.9 (ours)	89.2 (ours)	1×/1×
binNN [17]	67.0 (ours)	60.8	83.0	**1.31**×/**0.038**×
Bi-RealN [41]	–	56.4	88.1	–
ReActNet [42]	–	69.4	–	–
ternNN [23]	69.2 (ours)	61.8	84.2	1.60×/0.050×
ternNN [17]	–	66.6	87.2	–
ternBNN [21]	70.0 (ours)	68.0	87.5	1.71×/0.053×
LQ-Net-2 [37]	–	68.0	88.0	–
LQ-Net-3 [37]	–	69.3	88.8	–
UniQ-2 [38]	–	67.8	–	–
UniQ-3 [38]	–	70.5	–	–
DSQ [13]	–	68.7	–	–
PTQ-AR [43]	–	69.6	–	–
BRECQ [44]	–	70.7	–	–
DCNN (k=2) [18]	68.0	63.1	83.7	1.49×/0.042×
DCNN (k=4) [18]	70.2	68.0	88.1	1.73×/0.056×
DCNN (k=16) [18]	72.1	70.5	89.2	1.97×/0.099×
QNN (M=2)	69.2	65.2	84.9	2.48×/0.039×
QNN (M=3)	71.3	67.1	87.1	2.59×/0.054×
QNN (M=4)	72.2	68.9	89.0	2.63×/0.056×
QNN (M=8)	72.9	69.8	90.1	2.85×/0.079×
QNN (M=16)	71.9	70.9	89.3	3.01×/0.102×
BQNN (M=2)	70.1	66.0	86.2	2.48×/0.043×
BQNN (M=3)	71.9	68.6	88.9	2.59×/0.057×
BQNN (M=4)	73.1	70.2	90.2	2.71×/0.059×
BQNN (M=8)	74.0	71.0	90.8	2.97×/0.083×
BQNN (M=16)	73.2	71.6	89.9	3.10×/0.106×
c-QNN (M=8)	74.2	71.5	90.0	2.94×/0.080×
c-BQNN (M=8)	**74.5**	**72.0**	**91.0**	3.01×/0.083×

reason that M-ary quantization in QNN and BQNN *regularizes* the model training while FPNN may be over-trained. In case of M=16, QNN and BQNN only occupy around ten percent of memory consumption relative to FPNN. But, QNN with M=16 have been overestimated. QNN and BQNN with higher M are significantly better than previous methods with SOTA results in [1,21,23] although the storage and computation demands are increased as well. Five, BQNNs under different M consistently obtains higher accuracy than the corresponding QNNs. Bayesian learning does help. The overhead in memory requirement is very limited. Six, the standard deviations of classification accuracy using BQNNs are slightly smaller than those using QNNs. BQNN has higher confidence of the results than QNN. Seven, training time is raised by adopting the asymmetric partition, Bayesian learning and precise quantization. However, the critical concern in deployment of end devices is the test time rather than training time. Obviously, the test time using different quantized models is much smaller than that using full-precision model. Eight, the classification accuracies of QNN are higher than those of DCNN under M or k being 2, 4 and 16. But, the training time of QNN and BQNN in a way of joint training is considerably increased relative to that of DCNN where the separate estimation is performed. The metrics, memory reduction ratio and test time, are basically unchanged between QNN and DCNN under a fixed M. In general, the proposed QNN and BQNN are developed for implementation of end devices where the memory reduction and test time are more important relative to the training time.

In addition, this study examines the proposed methods by scaling up the number of classes and training samples in classification tasks. Table 4 compares the classification accuracy by using different methods where CIFAR-100 and ImageNet based on DenseNet and ResNet-18 are evaluated, respectively. The results of previous FPNNs in [17,23,41,42,46] and our FPNN are included for comparison. The competing results on binNN in [17], Bi-RealN [41], ReActNet [42], ternNN [17,23], ternBNN and the other related methods including LQ-Net [37], UniQ [38], DSQ [13], PTQ-AR [43], BRECQ [44] and DCNN [18] with k being 2, 4 and 16 are listed for comparison. LQ-Net-2 and LQ-Net-3 indicate the variants of LQ-Net with 2 and 3-bit subspace quantization, respectively. QNN and BQNN under various number of representation levels M are investigated. The results on binNN and ternNN in MNIST and CIFAR-100, and ternBNN in CIFAR-100 and ImageNet were implemented by ourselves. The results of Bi-RealN [41], ReActNet [42], ternNN [17], LQ-Net [37], UniQ [38], DSQ [13], PTQ-AR [43] and BRECQ [44] were retrieved from original papers. UniQ-2 and UniQ-3 imply the realizations of UniQ with 2 and 3 bits, respectively. Since the training data in ImageNet were scaled up, the computation using QNN and BQNN was simplified by using the pre-trained model with only a couple of updatings on classifier parameters. For the ablation study on non-uniform or asymmetric quantization, this is shown by the results via comparing ternNN and QNN with M=3 or comparing ternBNN and BQNN with M=3 in different tasks. The ablation study on new prior using multi-spike-and-slab distribution for non-uniform quantization is revealed by comparing the results of ternBNN and BQNN with M=3. From the results, the improvement of classification performance illustrates the merit of the proposed asymmetric quantization with the new prior/posterior setting. On the other hand, it is meaningful that the classification accuracy is increased by enhancing the model resolution using large M. In general, the findings of using different methods in CIFAR-100 and ImageNet are comparable with those in CIFAR-10. The proposed BQNN with higher M again achieves higher accuracy than FPNN. BQNN consistently works better than DCNN and QNN under various k or

Table 5
Comparison of top 1 accuracy (%), memory reduction ratio, training and test times of using different methods relative to FPNN where ImageNet is used and the architectures ResNet-18 and MobileNetV2 are evaluated.

Model	Architecture	Top 1 accuracy	Mem. reduc. ratio	Train/Test Times
FPNN		69.9	1× (11.4M params)	1×/1×
binNN [17]		60.8	**0.0259**×	**1.31**×/**0.038**×
ternBNN [21]	ResNet-18	68.0	0.0389×	1.71×/0.053×
c-QNN (M=8)		71.5	0.0689×	2.94×/0.080×
c-BQNN (M=8)		**72.0**	0.0758×	3.01×/0.083×
FPNN [38]		71.5	–	–
FPNN		70.9 (ours)	1× (3.4M params)	1×/1×
binNN		58.3	**0.0268**×	**1.39**×/**0.041**×
ternBNN		63.5	0.0397×	1.82×/0.068×
c-QNN (M=4)	MobileNetV2	69.4	0.0499×	2.59×/0.069×
c-QNN (M=8)		68.7	0.0701×	2.93×/0.082×
c-BQNN (M=4)		**71.0**	0.0512×	2.87×/0.089×
c-BQNN (M=8)		69.9	0.0793×	3.04×/0.098×

M. The accuracies of QNN are still higher than those of DCNN in these two tasks. In many cases, QNN and BQNN with M=8 performs better than those with the other M. The accuracies of QNN and BQNN with M=8 are higher than the competing ones of the previous methods. Finally, we fix this setting and further investigate the variants c-QNN and c-BQNN. These two variants are implemented by referring [37] which implements the channel-wise quantizers for weights where the convolution filters are individually quantized. As seen in the table, the recognition accuracies using these two variants are raised due to this sophisticated specification. Nevertheless, the margin of improvement of c-BQNN relative to c-QNN is reduced. In addition, the training and test times of different methods relative to FPNN on CIFAR-100 are evaluated under the same architecture using DenseNet. The increase of computation time from CIFAR-10 to CIFAR-100 is limited since the amount of training data is comparable in these two tasks.

Table 5 further reports the top 1 accuracy as well as the memory cost and computation time relative to FPNN where ImageNet dataset is used. In this set of experiments, the ablation study on architectures using ResNet-18 and a lightweight neural network called MobileNetV2 is investigated. The new variants of compressed models using c-QNN and c-BQNN with M=4 and M=8 are included for comparison. The results corresponding to ResNet-18 were retrieved from Table 4. The number of parameters of FPNN is 11.4M using ResNet-18 and 3.4M using MobileNetV2. Basically, top 1 accuracies using MobileNetV2 are lower than those of using ResNet-18 when the same compressed method is evaluated. The memory cost and computation time of different methods relative to FPNN in training and testing are comparable under both architectures. Considering the light architecture, c-BQNN still works better than c-QNN with the same M. The highest accuracy 71.0% is achieved by using c-BQNN with M=4. Increasing M to 8 does not improve the accuracy. A lighter model prefers to adopt a smaller quantization level M for an improved performance. Overall, the merits of Bayesian asymmetric quantization compared with full-precision model and symmetric quantization are illustrated in four tasks under different model architectures and specifications. The source code of implementing the Bayesian asymmetric quantized neural networks is accessible at https://github.com/NYCU-MLLab.

5. Conclusions

This paper has presented the Bayesian asymmetric learning for compressive neural representation. The representation values were generalized to implement the M-ary quantized neural networks with different number of representation values. A general solution to tradoff between model precision and memory requirement was developed. The overall parameters of weights, representation values and quantization partitions were estimated in a unified end-to-end manner to fulfill the proposed asymmetric M-ary quantization. Furthermore, Bayesian approach to M-ary quantization was extended so that the trained model was robust to heterogeneous image data. The procedure of sampling and quantizing weight parameters was implemented in convolutional neural networks. The discrete levels were sampled from differentiable function to calculate the variational lower bound of log likelihood for optimizing classification network. The multi-spike-and-slab prior was presented to truly reflect those representation values. The experiments on image classification under various architectures and tasks showed that the proposed Bayesian asymmetric quantization performed better than the deterministic, symmetric and restricted quantization in conventional methods. Classification performance was enhanced by elevating the quantization resolution via increasing the number of representation values. Bayesian compression performed better than deterministic compression. The multi-spike-and-slab prior was more fitted to Bayesian quantization than Gaussian prior. The estimated hyperparameters were densely clustered at the associated representation values. The memory cost was significant reduced by using the quantized models. The quantized network with an increased number of quantization values worked even better than full-precision network. Such an adjustable compression was potentially seen as a regularization method or alternatively viewed as an issue of model selection. Future work will be extended by automatically determining the number of quantization levels for Bayesian learning of compressive neural network. The learning tasks in natural language processing will be investigated.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

This work is partially supported by the National Science and Technology Council, Taiwan, under NSTC 111-2634-F-A49-012.

References

[1] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks, in: Advances in Neural Information Processing Systems, 2016, pp. 4107–4115.

[2] S.-K. Yeom, P. Seegerer, S. Lapuschkin, A. Binder, S. Wiedemann, K.-R. Müller, W. Samek, Pruning by explaining: a novel criterion for deep neural network pruning, Pattern Recognit. 115 (2021) 1–14.
[3] C. Blundell, J. Cornebise, K. Kavukcuoglu, D. Wierstra, Weight uncertainty in neural network, in: Proc. of International Conference on Machine Learning, 2015, pp. 1613–1622.
[4] J.-T. Chien, Y.-C. Ku, Bayesian recurrent neural network for language modeling, IEEE Trans. Neural Netw. Learn. Syst. 27 (2) (2016) 361–374.
[5] B. Dai, C. Zhu, B. Guo, D. Wipf, Compressing neural networks using the variational information bottleneck, in: Proc. of International Conference on Machine Learning, 2018, pp. 1135–1144.
[6] J.-T. Chien, Deep Bayesian multimedia learning, in: Proc. of ACM Multimedia Conference, 2020, pp. 4791–4793.
[7] D.J. Rezende, S. Mohamed, D. Wierstra, Stochastic backpropagation and approximate inference in deep generative models, in: Proc. of International Conference on Machine Learning, 2014, pp. 1278–1286.
[8] Q. Jin, L. Yang, Z. Liao, AdaBits: neural network quantization with adaptive bit-widths, in: Proc. of IEEE Conference on Computer Vision and Pattern Recognition, 2020, pp. 2146–2156.
[9] Y. Yang, L. Deng, S. Wu, T. Yan, Y. Xie, G. Li, Training high-performance and large-scale deep neural networks with full 8-bit integers, Neural Netw. 125 (2020) 70–82.
[10] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Advances in Neural Information Processing Systems, 2015, pp. 1135–1143.
[11] Y. Gong, L. Liu, M. Yang, L. Bourdev, Compressing deep convolutional networks using vector quantization, arXiv preprint arXiv:1412.6115(2014).
[12] D. Lin, S. Talathi, S. Annapureddy, Fixed point quantization of deep convolutional networks, in: Proc. of International Conference on Machine Learning, 2016, pp. 2849–2858.
[13] R. Gong, X. Liu, S. Jiang, T. Li, P. Hu, J. Lin, F. Yu, J. Yan, Differentiable soft quantization: bridging full-precision and low-bit neural networks, in: Proc. of International Conference on Computer Vision, 2019, pp. 4852–4861.
[14] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: Proc. of IEEE International Conference on Computer Vision, 2017, pp. 1389–1397.
[15] S. Swaminathan, D. Garg, R. Kannan, F. Andres, Sparse low rank factorization for deep neural network compression, Neurocomputing 398 (2020) 185–196.
[16] A.-H. Phan, K. Sobolev, K. Sozykin, D. Ermilov, J. Gusak, P. Tichavskỳ, V. Glukhov, I. Oseledets, A. Cichocki, Stable low-rank tensor decomposition for compression of convolutional neural network, in: Proc. of European Conference on Computer Vision, 2020, pp. 522–539.
[17] C. Zhu, S. Han, H. Mao, W.J. Dally, Trained ternary quantization, in: Proc. of International Conference on Learning Representations, 2017, pp. 1–10.
[18] S. Han, H. Mao, W.J. Dally, Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding, in: Proc. of International Conference on Learning Representations, 2016, pp. 1–14.
[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.
[20] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Quantized neural networks: training neural networks with low precision weights and activations, J. Mach. Learn. Res. 18 (1) (2017) 6869–6898.
[21] J. Achterhold, J.M. Koehler, A. Schmeink, T. Genewein, Variational network quantization, in: Proc. of International Conference on Learning Representations, 2018, pp. 1–18.
[22] Y. Xu, Y. Wang, A. Zhou, W. Lin, H. Xiong, Deep neural network compression with single and multiple level quantization, in: Proc. of AAAI Conference on Artificial Intelligence, Vol. 32, 2018, pp. 4335–4342.
[23] F. Li, B. Zhang, B. Liu, Ternary weight networks, arXiv preprint arXiv:1605.04711(2016).
[24] Y. Choi, M. El-Khamy, J. Lee, Towards the limit of network quantization, in: Proc. of International Conference on Learning Representations, 2017, pp. 1–12.
[25] J.-T. Chien, S.-T. Chang, M-ary quantized neural networks, in: Proc. of IEEE International Conference on Multimedia and Expo, 2020, pp. 1–6.
[26] Y. Bengio, N. Léonard, A. Courville, Estimating or propagating gradients through stochastic neurons for conditional computation, arXiv preprint arXiv:1308.3432(2013).
[27] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, in: Proc. of International Conference on Machine Learning, 2015, pp. 448–456.
[28] S. Watanabe, J.-T. Chien, Bayesian Speech and Language Processing, Cambridge University Press, 2015.
[29] J.-T. Chien, C.-W. Wang, Hierarchical and self-attended sequence autoencoder, IEEE Trans. Pattern Anal. Mach. Intell. 44 (9) (2022) 4975–4986.
[30] D.P. Kingma, M. Welling, Auto-encoding variational Bayes, in: Proc. of International Conference on Learning Representations, 2014, pp. 1–14.
[31] J.-T. Chien, S.-T. Chang, Bayesian learning for neural network compression, in: Proc. of IEEE International Conference on Multimedia & Expo Workshops, 2020, pp. 1–6.
[32] D.P. Kingma, T. Salimans, M. Welling, Variational dropout and the local reparameterization trick, in: Advances in Neural Information Processing Systems, 2015, pp. 2575–2583.
[33] C. Louizos, K. Ullrich, M. Welling, Bayesian compression for deep learning, in: Advances in Neural Information Processing Systems, 2017, pp. 3288–3298.
[34] D. Molchanov, A. Ashukha, D. Vetrov, Variational dropout sparsifies deep neural networks, in: Proc. of International Conference on Machine Learning, 2017, pp. 2498–2507.
[35] J.-T. Chien, Y.-L. Chang, Bayesian sparse topic model, J Signal Process Syst 74 (3) (2014) 375–389.
[36] O. Shayer, D. Levi, E. Fetaya, Learning discrete weights using the local reparameterization trick, in: Proc. of International Conference on Learning Representations, 2018, pp. 1–12.
[37] D. Zhang, J. Yang, D. Ye, G. Hua, LQ-Nets: learned quantization for highly accurate and compact deep neural networks, in: Proc. of European Conference on Computer Vision, 2018, pp. 365–382.
[38] P. Pham, J.A. Abraham, J. Chung, Training multi-Bit quantized and binarized networks with a learnable symmetric quantizer, IEEE Access 9 (2021) 47194–47203.
[39] S.K. Esser, J.L. McKinstry, D. Bablani, R. Appuswamy, D.S. Modha, Learned step size quantization, in: Proc. of International Conference on Learning Representations, 2020, pp. 1–12.
[40] A. Gholami, S. Kim, Z. Dong, Z. Yao, M.W. Mahoney, K. Keutzer, A survey of quantization methods for efficient neural network inference, arXiv preprint arXiv:2103.13630(2021).
[41] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, K.-T. Cheng, Bi-Real Net: enhancing the performance of 1-bit CNNs with improved representational capability and advanced training algorithm, in: Proc. of European Conference on Computer Vision, 2018, pp. 722–737.
[42] Z. Liu, Z. Shen, M. Savvides, K.-T. Cheng, ReActNet: towards precise binary neural network with generalized activation functions, in: Proc. of European Conference on Computer Vision, 2020, pp. 143–159.
[43] M. Nagel, R.A. Amjad, M. Van Baalen, C. Louizos, T. Blankevoort, Up or down? Adaptive rounding for post-training quantization, in: Proc. of International Conference on Machine Learning, 2020, pp. 7197–7206.
[44] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, S. Gu, BRECQ: pushing the limit of post-training quantization by block reconstruction, in: Proc. of International Conference on Learning Representations, 2021, pp. 1–16.
[45] J.-T. Chien, Y.-T. Bao, Tensor-factorized neural networks, IEEE Trans. Neural Netw. Learn. Syst. 29 (5) (2018) 1998–2011.
[46] G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: Proc. of IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4700–4708.
[47] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proc. of IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[48] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, MobileNetV2: inverted residuals and linear bottlenecks, in: Proc. of IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.
[49] L. Hertel, E. Barth, T. Käster, T. Martinetz, Deep convolutional neural networks as generic feature extractors, in: Proc. of International Joint Conference on Neural Networks, 2015, pp. 1–4.
[50] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: Proc. of International Conference on Learning Representations, 2015, pp. 1–10.

Jen-Tzung Chien is the Professor in the Institute of Electrical and Computer Engineering, National Yang Ming Chiao Tung University (NYCU), Taiwan. He served as the tutorial speaker of AAAI, ACL, KDD, MM, CIKM and ICASSP. His research interests include machine learning, computer vision and natural language processing.

Su-Ting Chang received her B.S. degree from Interdisciplinary Program of Engineering, National Tsing Hua University, Taiwan, in 2017, and the M.S. degree from the Institute of Communications Engineering, NYCU, in 2019. Her research interests include deep learning and model compression.