PAPER *Special Section on Multiple-Valued Logic and VLSI Computing*

Extending Binary Neural Networks to Bayesian Neural Networks with Probabilistic Interpretation of Binary Weights

Taisei SAITO[†], ***Student Member***, **Kota ANDO**[††], ***and*** **Tetsuya ASAI**[††a)], ***Members***

SUMMARY Neural networks (NNs) fail to perform well or make excessive predictions when predicting out-of-distribution or unseen datasets. In contrast, Bayesian neural networks (BNNs) can quantify the uncertainty of their inference to solve this problem. Nevertheless, BNNs have not been widely adopted owing to their increased memory and computational cost. In this study, we propose a novel approach to extend binary neural networks by introducing a probabilistic interpretation of binary weights, effectively converting them into BNNs. The proposed approach can reduce the number of weights by half compared to the conventional method. A comprehensive comparative analysis with established methods like Monte Carlo dropout and Bayes by backprop was performed to assess the performance and capabilities of our proposed technique in terms of accuracy and capturing uncertainty. Through this analysis, we aim to provide insights into the advantages of this Bayesian extension.

key words: *Bayseian neural network, Binary connect, quantization, memory reduction, uncertainty estimation*

1. Introduction

In recent years, neural networks (NNs) have been widely adopted in a various applications, such as image classification [1], natural language processing [2], and object detection [3]. However, NN models are prone to overfitting to their training data and exhibit low robustness to outliers [4]. For example, NNs can make highly confident erroneous predictions when presented with out-of-distribution samples (unknown classes). Bayesian neural networks (BNNs) [5], [6], which apply Bayesian inference to deep learning architecture, can estimate prediction uncertainty and ensure prediction confidence. Moreover, BNNs solve problems such as out-of-distribution detection and model interpretability, which are critical for deploying of AI systems in real-world applications.

BNNs assume a probability distribution for a set of weights and can estimate the uncertainty inherent in data by executing multiple feedforward propagations by sampling the scalar values from the weight distributions [5]. Therefore, they require more computational cost and memory footprint, posing substantial challenges when deployed in resource-constrained environments. The burgeoning demand for implementing BNNs on edge devices has necessitated a paradigm shift towards a more efficient Bayesian deep learning method [7], [8]. This research domain encompasses model compression and efficient approximation strategies to reduce the computational and hardware resource requirements of BNNs without compromising uncertainty quantification, model calibration, and robustness. The implications of efficient Bayesian deep learning are far-reaching, spanning domains like autonomous robotics, edge AI, and real-time decision-making systems. Such models can provide actionable uncertainty estimates even in resource-constrained environments, improving safety, reliability, and interpretability.

This study proposes a new method to compress Bayesian deep learning models by introducing an existing hardware-oriented weight binarization algorithm into variational inference with Bernoulli distribution. The contributions of this paper are as follows:

1. We proposed a learning method for BNNs with binary weights, more suitable for hardware implementation, by combining the learning approach of binary NNs with variational inference methods of BNNs.
2. We compared the performance of our proposed method against existing BNN methods in terms of accuracy and uncertainty capture.

2. Preliminaries

2.1 Bayesian Neural Networks

Compared to the vanilla NN models designed to learn the values of weights, BNNs learn the probability distribution of weights by using Bayes' theorem,

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)} \tag{1}$$

and find the maximum a posteriori (MAP) weights, where D is the training dataset, w is the weight of the model, $p(w)$ is the prior distribution of the weight, $p(D|w)$ is the likelihood, and $p(w|D)$ is the posterior distribution. Owing to the high dimensionality of $p(D) = \int p(D|w)p(w)dw$, which is the integral of all possible latent variables, computing the posterior distribution directly using Bayes' theorem is intractable. Instead, in variational inference [9], the posterior distribution is approximated by a restricted distribution $q(w|\theta)$ with the learnable parameter θ. θ is optimized by minimizing the Kullback-Leibker (KL)-divergence between the approximated distribution $q(w|\theta)$ and the true distribution $p(w|D)$.

Manuscript received October 26, 2023.
Manuscript revised March 3, 2024.
Manuscript publicized April 17, 2024.

[†]Graduate School of Information Science and Technology, Hokkaido University, Sapporo-shi, 060–0814 Japan.
[††]Faculty of Information Science and Technology, Hokkaido, Hokkaido University, Sapporo-shi, 060–0814 Japan.
a) E-mail: asai@ist.hokudai.ac.jp

DOI: 10.1587/transinf.2023LOP0009

To introduce the objective function in variational inference, we transform the log marginal likelihood $\ln p(D)$ as follows:

$$\begin{aligned} \ln p(D) &= \ln \int p(D,w)dw \\ &= \ln \int q(w|\theta)\frac{p(D,w)}{q(w|\theta)}dw \\ &\geq \int q(w|\theta) \ln \frac{p(D,w)}{q(w|\theta)}dw \\ &= \mathcal{L}[q(w|\theta)], \end{aligned} \tag{2}$$

where, $\mathcal{L}[q(w|\theta)]$ represents the lower bound of the log marginal likelihood, known as the Evidence Lower Bound (ELBO), and Jensen's inequality is used in the third line of equation. Furthermore, the ELBO is transformed as follows:

$$\begin{aligned} \mathcal{L}[q(w|\theta)] &= \int q(w|\theta) \ln \frac{p(w|D)p(D)}{q(w|\theta)}dw \\ &= \int q(w|\theta) \ln \frac{p(w|D)}{q(w|\theta)}dw + \int q(w|\theta) \ln p(D)dw \\ &= -D_{KL}(q(w|\theta)||p(w|D)) + \ln p(D). \end{aligned} \tag{3}$$

Here, the second term in (3) does not depend on θ, minimizing the first term in (3) is equivalent to maximizing the ELBO. The first term of (3) is transformed as follows:

$$\begin{aligned} D_{KL}(q(w|\theta)||p(w|D)) &= -\int q(w|\theta) \ln \frac{p(w|D)}{q(w|\theta)}dw \\ &= -\int q(w|\theta) \ln p(D|w)dw + D_{KL}(q(w|\theta)||p(w)) + \mathrm{C} \\ &= -E_{q(w|\theta)}[\ln p(D|w)] + D_{KL}(q(w|\theta)||p(w)), \end{aligned} \tag{4}$$

where, $\mathrm{C} = \int q(w|\theta) \ln p(D)dw$ and since this term does not depend on θ, it was removed in the third line of Eq. (4). Thus, the objective function is calculated as (4), and minimized using the gradient descent method. Here, the first term in (4) is the expected negative log-liklihood computed by sampling weights from $q(w|\theta)$ multiple times, and multiple feedforward operations. The second term in (4) is the regularization term. In the testing time, predictions for unseen data x, y are calculated using the variational posterior distribution $q(w|\theta)$ as follows:

$$p(y|x,D) = \int p(y|w,x)q(w|\theta)dw. \tag{5}$$

However, because the integral of all possible latent variables is also intractable, the prediction, Eq. (5) is approximated by Monte Carlo (MC) sampling as follows:

$$p(y|x,D) = \frac{1}{T}\sum_{t=1}^{T} p(y|w_t,x). \tag{6}$$

Here, T is the number of weights sampled from the variational posterior distribution $q(w|\theta)$.

2.2 Bayes By Backprop

Blundell et al. proposed the Bayes by backprop (BBB) [5] algorithm, which uses backpropagation to optimize a BNN model with Gaussian approximation. In BBB, the posterior distribution of the weights $q(w|\theta)$ is approximated with Gaussian distribution parametrized by mean μ and variance σ^2. The weight value is sampled from Gaussian distribution as $q(w|\theta) \sim \mathcal{N}(\mu, \sigma^2)$, where $\mathcal{N}$ represents Gaussian distribution, and θ represents μ and σ. However, the sampling obtained from Gaussian distributions is not differentiable as it lacks an explicit formula, making it impossible to propagate gradients and optimize the BNN objective (4). Therefore, reparameterization trick [10] is employed to generate weights as follows:

$$\begin{aligned} w &= \mu + \epsilon \odot \ln(1 + \exp(\rho)), \\ \epsilon &\sim \mathcal{N}(0,1), \end{aligned} \tag{7}$$

where $\odot$ is the point-wise multiplication operator, and ρ is a learnable parameter to represent standard deviation σ as positive value, that is $\sigma = \ln(1 + \exp(\rho))$. Then, the posterior distribution of the weight is optimized by applying the gradient descent method with respect to the mean and standard deviation of the Gaussian distribution.

2.3 Monte-Carlo Dropout

Gal and Ghahramani proposed the Monte Carlo dropout (MCD) [6] algorithm, which extends dropout, primarily used for model regularization It is the same as variational inference and leverages the Bernoulli distribution of dropout layers within deep neural networks. The Bernoulli approximating variational distribution of the weights $q(w_i)$ for each layer i of $K \times K$ dimension is defined as follows:

$$\begin{aligned} q(w_i) &= M_i \odot \mathrm{diag}([z_i^j]_{j=1}^{K_i}) \\ z_i^j &\sim \mathrm{Bernoulli}(p_i) \quad \text{for } i = 1,\dots,L,\ j = 1,\dots,K_{i-1}, \end{aligned} \tag{8}$$

where z_i^j is the Bernoulli distributed variable with probability p_i, and M_i are the variational parameters to be trained. The training is almost the same as in vanilla NN training, but involves minimizing KL-divergence (4) in variational inference. This innovative approach involves performing multiple forward passes through a NN with dropout enabled at test time, resulting in an ensemble of stochastic predictions. By averaging these predictions, MCD can compute the uncertainty estimates of model outputs such as predictive variances, quantifing of prediction confidence and reliability.

2.4 BinaryConnect

Courbariax et al (2015). proposed BinaryConnect [11] as a training method for neural networks with binary weights and full-precision activations. In standard deep learning models, the weight parameters are continuous values, demanding substantial memory and computational resources during both training and inference. In contrast, BinaryConnect maintains the weights in floating-point precision during

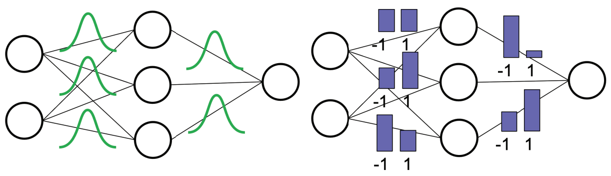


Fig. 1 Graphical overview of the difference in weights of BBB (left) and proposed method (right). The BBB's weight is a Gaussian distribution, whereas that of the proposed method is a Bernoulli distribution. Binary weight values are represented as a Bernoulli distribution.

training to approximate gradient for binary weight representation using a straight-through estimator(STE) [12] and quantizes them into binary values, typically -1 and 1, with a sign function during the test time. This binarization process significantly reduces memory consumption and computational overhead. BinaryConnect has demonstrated its efficacy in various machine learning tasks, making it a compelling technique for optimizing neural network deployment in real-world scenarios. Its ability to balance model efficiency and performance has positioned it as a noteworthy advancement in deep learning, particularly when computational resources are limited.

3. Probabilistic Interpretation of Binary Weights

In the conventional BBB method, the weight parameters are represented as a Gaussian distribution. Such models require twice the memory of vanilla and MCD models because they utilize mean and standard deviation of parameters.

To solve this problem, we replace the weight parameter from the Gaussian to Bernoulli distribution with binary weight. Unlike the Gaussian distribution, the Bernoulli distribution has only a single parameter θ, which can express mean and variance as θ and $\theta(1-\theta)$, respectively. Because the weight is represented as a binary value (-1, or 1) following Bernoulli distribution, we partially adopt a BinaryConnect scheme [11] to train the BNN model, approximated as a Bernoulli distribution.

During inference, the binary weight w_b is sampled from the Bernoulli distribution with parameter θ as follows:

$$w_b = 2 \times \text{Bernoulli}(\theta) - 1. \tag{9}$$

Here, Bernoulli(θ) means sampling 1 and 0 from the Bernoulli distribution with probability θ and $1-\theta$, respectively. Multiple feedforward operations were executed to compute the expected value of the negative log-likelihood. A backpropagation algorithm must be used to train the model. However, because the Bernoulli distribution is not consecutive, it is not differentiable, and its gradient cannot be calculated. In the BinaryConnect training scheme, STE estimates discrete and non-differentiable binarized weight gradients by passing the gradient backward without modification. However, in our proposed method, we define the gradient of the Bernoulli distribution as follows:

$$\frac{\partial w_b}{\partial \theta} = 2 \times \frac{\partial}{\partial \theta}\text{Bernoulli}(\theta)$$

Algorithm 1 Pseudo code for training of Bernoulli weight BNN

```
Require: a minibatch of inputs and targets $(x_0, t)$, previous parameters $\theta$,
  learning rate $\eta$, regularization parameter $\beta$
Ensure: updated parameters $\theta^t$
  1. Multiple Forward Propagation
  for $t = 1$ to $T$ do
    for $l = 1$ to $L$ do
      $\theta_l^b \leftarrow 2 \cdot \text{Bernoulli}(\theta_l) - 1$
      $u_l \leftarrow \text{BatchNorm}(x_{l-1} \cdot \theta_l^b)$
      if $l < L$ then
        $x_l \leftarrow \text{ReLU}(u_l)$
      end if
    end for
    $accum_t \leftarrow \text{Softmax}(u_L)$
  end for
  $out \leftarrow \frac{1}{T}\sum_{t=1}^{T}(accum_t)$
  2. Backward Propagation
  Compute $\frac{\partial \mathcal{F}}{\partial u_L}$ knowing $out$ and $t$
  for $l = L$ to 1 do
    $\frac{\partial \theta_l^b}{\partial \theta_l} \leftarrow 2 \cdot \theta_l \cdot (1-\theta_l)$
    $\frac{\partial \mathcal{F}}{\partial u_{l-1}} \leftarrow \frac{\partial \mathcal{F}}{\partial u_l} \cdot \theta_l^b$
    $\frac{\partial \mathcal{F}}{\partial \theta_l} \leftarrow \frac{\partial \mathcal{F}}{\partial u_l} \cdot x_{l-1} \cdot \frac{\partial \theta_l^b}{\partial \theta_l}$
  end for
  3. Parameter update
  for l=1 to L do
    Compute $\frac{\partial}{\partial \theta_l} D_{KL}(Q||P)$ knowing $\theta_l$
    $\theta_l^t \leftarrow \text{Clamp}(\theta_l - \eta \cdot \frac{\partial \mathcal{F}}{\partial \theta_l} - \frac{1}{\beta} \cdot \frac{\partial}{\partial \theta_l} D_{KL}(Q||P), 0, 1)$
  end for
```

$$\triangleq 2 \times (1-\theta) \times \theta. \tag{10}$$

We created a gradient with the local maximum value at 0.5, preventing the weights from concentrating on 0.5. This is because the probability of 0.5 is a completely random output, and we consider this as the source of the noise-causing error.

We calculated the KL-divergence between Q and P, two Bernoulli distributions with parameters μ_1, μ_2 as follows.

$$D_{KL}(Q||P) = \mu_1 \ln \frac{\mu_1}{\mu_2} + (1-\mu_1)\ln \frac{1-\mu_1}{1-\mu_2}. \tag{11}$$

In our method, parameter μ_2, prior distribution for Bernoulli distribution is constant ($\mu_2 = 0.5$), whereas μ_1 is a learnable parameter ($\mu_1 = \theta$). Furthermore, for numerical stability, we add 1 inside the ln() function when performing calculations on a computer. The derivative of the KL-divergence is calculated as follows.

$$\begin{aligned} D_{KL}(Q||P) &= \theta \ln \frac{\theta}{0.5} + (1-\theta)\ln \frac{1-\theta}{1-0.5}, \\ \frac{\partial}{\partial \theta} D_{KL}(Q||P) &= \ln \frac{\theta}{1-\theta}. \end{aligned} \tag{12}$$

Parameter θ gradually approaches 0.5 when optimized and serves as a regularization term. This regularization term strongly penalizes the Bernoulli distribution with deterministic probabilities, preventing overfitting and keeping the stochastic behavior of our BNN method. This regularization term is controlled by parameter β, determined as follows [5].

$$\beta = \frac{2^{M-i}}{2^{M}-1}, \tag{13}$$

where M denotes the current number of the subsets split from the training dataset, and i denotes the number of iterations. The first few iterations of the subsets are heavily influenced by the KL-divergence, whereas the later subsets are gradually weakened. Thus, the objective function $\mathcal{F}(\theta)$ is computed as follows.

$$\mathcal{F}(\theta) = -E_{q(w|\theta)}[\ln p(D|w)] + \frac{1}{\beta} D_{KL}(Q||P). \tag{14}$$

The weight value is clamped from 0 to 1 when updated because we aim to calculate the mean of the Bernoulli distribution with a probability of sampling 0 or 1. The pseudocode of our algorithm is shown in Algorithm 1.

4. Experiment

4.1 Multiclass Classification Task

We evaluated the proposed method in terms of accuracy and uncertainty in the multiclass classification task compared to other BNN methods, BBB and MCD. The PyTorch framework was used to implement the models. We used the Adam optimizer with default coefficients from PyTorch. The mini-batch size was set to 64. The number of MC samples during training was 5, and during evaluation, it was 10. The loss function was the cross-entropy loss because of using the softmax function in the last activation function. We used the MNIST dataset [13] to evaluate the performance of our method with a multi-layer perceptron (MLP) model. MNIST is a handwritten digit dataset of 70,000 28*28 grayscale images representing Arabic numerals from 0 to 9, with 7,000 images per digit. The dataset contains 60,000 training and 10,000 testing images. We did not use data augmentation. The MLP model was implemented with 32-bit floating point for all operations and parameters as follows.

$$\begin{aligned}&input(784) - fc(200) - fc(200)\\&\qquad -fc(10) - softmax(10),\end{aligned}$$

where fc is the fully connected layer followed by batch normalization and the ReLU function, excluding the last fully connected layer. As for the training of MLP, the learning rate was set to 0.01 and the number of epochs is 300. The hyperparameters used in MNIST classification are listed in Table 1.

Additionally, to confirm the performance of the larger model, we used the CIFAR-10 [14] dataset and evaluated the performance of our approach in comparison with that of a ResNet-18 model, which is a basic convolutional neural network (CNN) architecture. CIFAR-10 is a dataset of 60,000 32*32 color images in 10 classes, with 6,000 images per class, divided into 50,000 training images and 10000 testing images. To compensate for the low accuracy of the ResNet-18 model, we used data augmentation by applying vertical flips with a probability of 0.5. The following architecture was used for the ResNet-18 model with 32-bit floating point for all operations and parameters.

$$\begin{aligned}&input(3*32*32) - 64C7K1S - 64C3K2S \times 2\\&- 128C3K2S \times 2 - 256C3K2S \times 2 - 512C3K2S \times 2\\&- GAP - fc(10) - softmax(10)\end{aligned}$$

where $nCxKzS$ is the convolution layer with kernel size x and stride z for n-channels followed by batch normalization and the ReLU function, and GAP is global average pooling layer. As for the training of ResNet-18, the initial learning rate was set to 0.005, and the validation results were monitored for 5 epochs out of 50 epochs. If no improvement was observed during this period, the learning rate was reduced by a factor of 0.5. The hyperparameters used in CIFAR-10 classification are listed in Table 1.

The results are provided in terms of accuracy in Table 2. The proposed method can classify the samples in the MNIST dataset with accuracy similar to that of the conventional

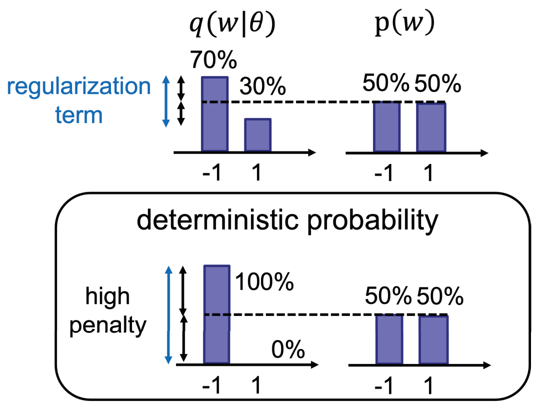


Fig. 2 Graphical overview of the regularization term which is a measure of the distance between the parameterized Bernoulli distribution $q(w|\theta)$ and the Bernoulli distribution with 50%, p(w). This regularization term strongly penalizes the Bernoulli distribution with deterministic probabilities.

Table 1 Hyperparameters of each method used in the multiclass classification of CIFAR-10 and MNIST datasets.

Mini-batch size	64	
Optimizer	Adam	
Weight decay	0.0	
Beta1,Beta2	0.9, 0.999	
# of MC samples	train:5, evaluation:10	
Loss Function	Cross entropy loss	
Dataset	MNIST	CIFAR-10
Number of epoch	300	50
Learning rate	0.01	0.005
Scheduler	None	ReduceLROnPlateau

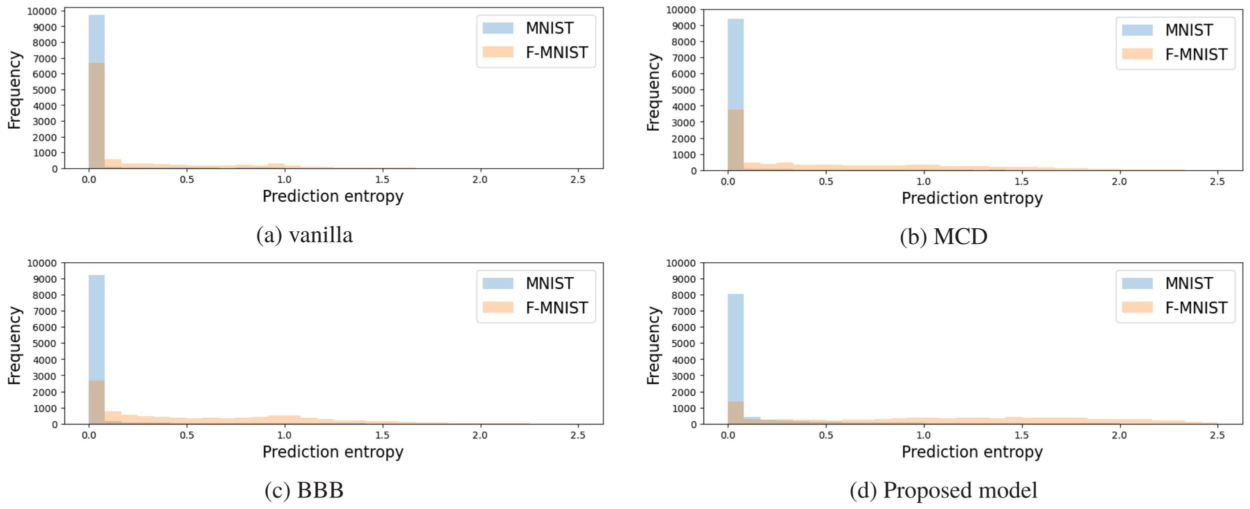


Fig. 3 Histgrams of prediction entropy for each BNN method with MLP. The blue and orange bins correspond to the MNIST and Fashion-MNIST datasets, respectively; each dataset comprise 10,000 images. Each method concentrates on the low and high entropy regions for the MNIST and Fashion-MNIST datasets, respectively.

Table 2 Result of prediction accuracy of each method. We measured the performance using MLP on the MNIST dataset and ResNet-18 on the CIFAR-10 dataset; each dataset comprises 10,000 test images. Each result was computed by repeating the average prediction of 10 MC samples 10 times.

Method	MNIST (MLP)	CIFAR-10 (ResNet-18)
vanilla	98.46 ± 0.00%	91.39 ± 0.00%
BBB	98.46 ± 0.02%	89.69 ± 0.08%
MCD	98.45 ± 0.03%	92.04 ± 0.06%
Our method	98.43 ± 0.03%	88.72 ± 0.21%

methods. However, for the CIFAR-10 classification, the proposed method exhibited an accuracy 3.32% lower than that of MCD because the error in the binarization of the weights accumulates as the model becomes larger. However, because the accuracy of our proposed method is relatively high, it is applicable to CNN models.

We calculated the average prediction entropy (aPE) for in-distribution and out-of-distribution datasets to measure the uncertainty of the model's predictions. The aPE is obtained by averaging the entropy of likelihoods for each class under the softmax function across all data points, calculated using Eq. (15) over a dataset of size D and class K.

$$aPE = \frac{1}{D}\sum_{i=1}^{D}\left[-\sum_{k=1}^{K} p(y_i^k|x_i)\log p(y_i^k|x_i)\right]. \tag{15}$$

We evaluated the uncertainty for out-of-distribution data sampled from the Fashion-MNIST [15] and SVHN [16] dataset by calculating aPE of MLP and CNN with each method trained on MNIST and CIFAR-10 dataset, respectively, which is in-distribution data. Table 3 and Table 4 show aPE of Bayesian and vanilla NN methods trained on MNIST and CIFAR-10 datasets, respectively. Each BNN method can describe the uncertainty by predicting with high entropy for out-of-distribution dataset compared to the vanilla method. Figure 3 shows a histogram of the prediction entropy with each method using MLP. A histogram of the out-of-distribution dataset is spread widely by binarizing the weights, whereas the in-distribution dataset is concentrated on 0. Therefore, our method achieved higher entropy for outliers and described uncertainty more accurately. We attribute this result to weight binarization, which avoids overfitting the training dataset. Figure 4 shows a a histogram of the prediction entropy with each method using ResNet-18. Compared to previous methods, our approach exhibits high entropy in out-of-distribution data predictions but struggles with low-entropy predictions for in-distribution data. This is owing to the reduction in model expressiveness caused by binarization, which results in ResNet-18 not fitting the dataset adequately.

Additionally, we measure expected calibration error (ECE) [17], a metric indicating how confidence accuracy in the model's predictions. Thus, it measures whether a

Table 3 Result of average prediction entropy (aPE). It is measured using MLP with each three BNN methods and a vanilla method against the learned MNIST dataset and Fashion-MNIST dataset, which is not learned; each dataset comprises 10,000 test images.

Method	MNIST	FMNIST (outliers)
vanilla	0.0156 nats	0.200 nats
BBB	0.0461 nats	0.569 nats
MCD	0.0375 nats	0.540 nats
Our method	0.1137 nats	1.093 nats

Table 4 Result of average prediction entropy (aPE). It is measured using ResNet-18 with each three BNN methods and a vanilla method against the learned CIFAR-10 dataset and SVHN dataset, which is not learned; each dataset comprises 10,000 test images.

Method	CIFAR-10	SVHN (outliers)
vanilla	0.151 nats	0.558 nats
BBB	0.419 nats	1.495 nats
MCD	0.248 nats	1.406 nats
Our method	0.881 nats	1.997 nats

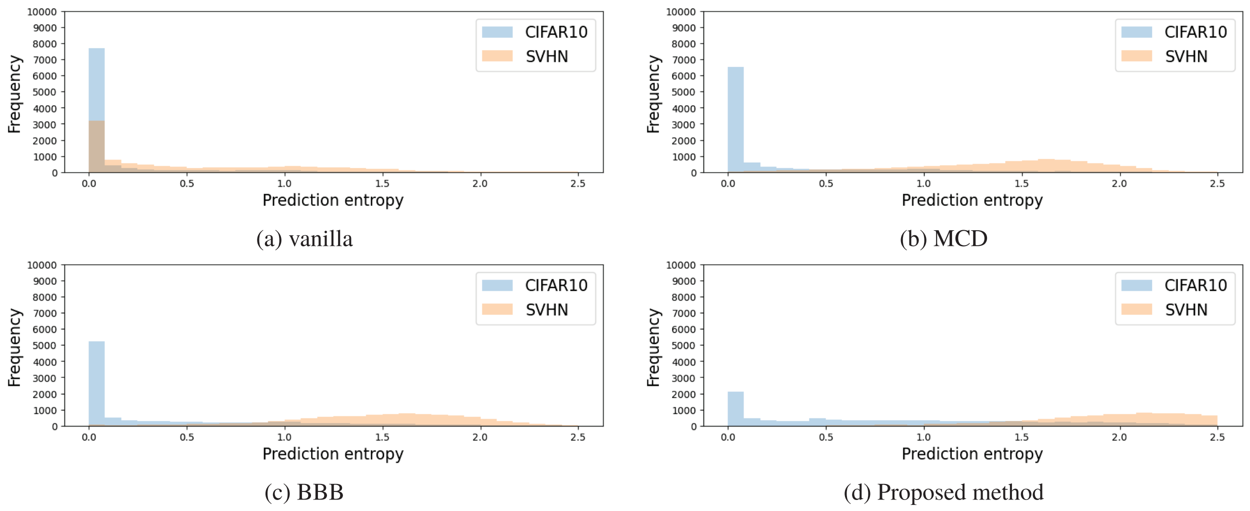


Fig. 4 Histgrams of prediction entropy for each BNN method with ResNet-18. The blue and orange bins correspond to the CIFAR-10 and SVHN datasets, respectively; each dataset comprises 10,000 images. Our proposed method exhibits high prediction entropy on the out-of-distribution SVHN dataset compared to the other methods. Simultaneously, it demonstrates relatively high prediction entropy on the in-distribution CIFAR-10 dataset.

Table 5 Expected calibration error of each method. We measured the performance using MLP on the MNIST dataset and ResNet-18 on the CIFAR-10 dataset, each comprising of 10,000 test images.

Method	MNIST (MLP)	CIFAR-10 (ResNet-18)
vanilla	1.105 %	5.460 %
BBB	0.590 %	0.977 %
MCD	0.717 %	1.507 %
Our method	0.902 %	8.877 %

Table 6 Hyperparameters used in regression of sine curve dataset and binary classification of 2-Moon dataset.

Optimizer	Adam	
Loss Function	Mean squared error loss	
Beta1,Beta2	0.9, 0.999	
Scheduler	None	
# of MC samples	train:5, evaluation:10	
Dataset	2-Moon	Sine curve
Mini-batch size	32	200
Number of epoch	2000	1000
Learning rate	0.01	0.001
weight decay	0.0	0.001

network is over-confident or under-confident in its predictions. To compute ECE, we discretize the model's predicted probability values into several bins and assign the model's predictions to a specific bin. Then, a weighted average of the difference in the accuracy and confidence across bins is computed as follows:

$$\mathrm{ECE} = \sum_{b=1}^{B} \frac{n_B}{N} |\mathrm{accuracy}(b) - \mathrm{confidence}(b)|, \tag{16}$$

where B is number of discretized bins and n_b is the number of predictions in bin b; accuracy(b) and confidence(b) are the averages of accuracy and confidence of bin b, respectively. Here, $B = 10$.

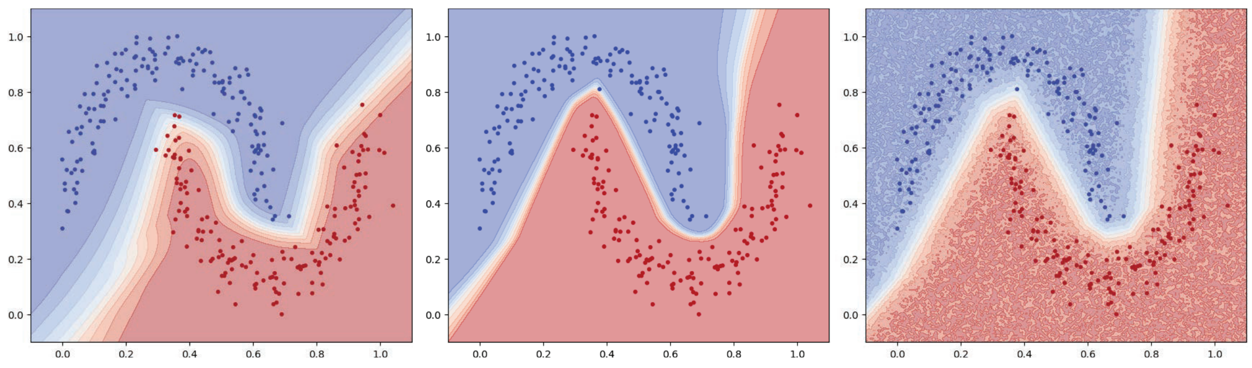
Table 5 shows the result of ECE for MLP and ResNet-18 models. In classifying of MNIST using an MLP, our method exhibits lower ECE than the vanilla method, indicating good calibration. However, in classifying CIFAR-10 using ResNet-18, ECE is significantly higher than other methods due to the proposed model's lack of expressiveness caused by weight binarization.

4.2 Binary Classification Task

To visualize the uncertainty of the BNN models in a classification task, we conduct an experiment using the 2-Moon dataset, a well-known synthetic dataset often used to illustrate binary classification concepts. The 2-Moon dataset comprises of two crescent-shaped clusters of data points, each representing a different class. We generated 100 data points and added Gaussian noise with a standard deviation of 0.1 for each class. The MLP model used in this experiment was implemented with 32-bit floating point for all operations and parameters as follows.

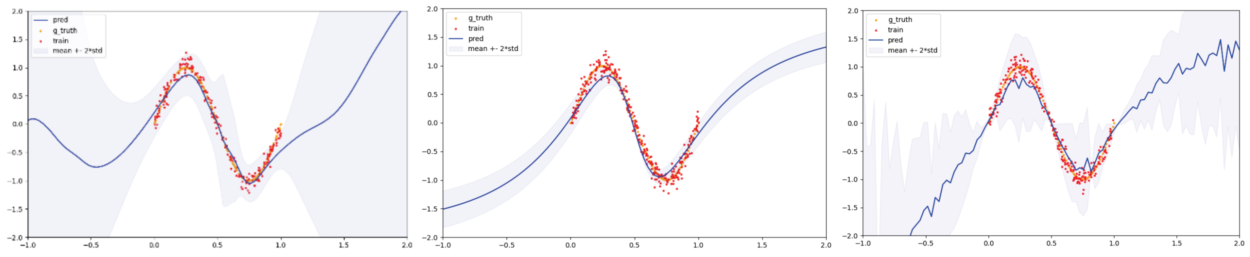
$$input(2) - fc(64) - fc(64) - Sigmoid(1),$$

where fc is the fully connected layer followed by batch normalization and the ReLU function, excluding the last fully connected layer, and Id is the identity function. The hyperparameters used in this experiment are listed in Table 6. Figure 5 shows the classification result with three BNN methods. Each BNN model exhibits a low likelihood near class decision boundaries, allowing for effective uncertainty estimation. However, our model has wider regions of lower likelihood near decision boundaries, while MCD shows irregularly scattered regions of low likelihood even for data points far from the decision boundaries. Additionally, BBB has the narrowest low likelihood decision boundaries and the highest overconfidence.



(a) Proposed method (b) BBB (c) MCD

Fig. 5 Classification results on the two moons dataset with the three BNN methods. Left: proposed method; middle: BBB using the Gaussian distribution; right: MCD. Each result is the average of 10 prediction samples. The red and blue points represent class 0 and 1 data points, respectively. The model becomes more confident in classifying data as class 0 and 1 as the region becomes red and blue, respectively.



(a) Proposed method (b) BBB (c) MCD

Fig. 6 Regression results with sine function for the three different BNN methods. Left: proposed algorithm; middle: BBB; right: MCD. The red data points represent the training data for the sine function, yellow line represents the true sine function, and blue line represents model predictions, which is the average of 10 sampled predictions. The light blue envelope represents the 95% confidence interval.

4.3 Regression Task

To visualize the uncertainty of BNN models in a regression problem, we conduct an experiment using a simple sine function regression. We generated training data from the sine function curve with 200 data points as follows:

$$y = \sin(2\pi(x)) + \epsilon \qquad (0 \leq x \leq 1), \tag{17}$$

where ϵ is the Gaussian noise with standard deviation 0.1. The MLP model used in this experiment was implemented with 32-bit floating point for all operations and parameters as follows.

$$input(1) - fc(32) - Id(1),$$

where fc is the fully connected layer followed by batch normalization and the tanh function, excluding the last fully connected layer, and Id is the identity function. The Hyperparameters used in this experiment are listed in Table 6. Figure 6 shows the regression result of the generated data points using three BNN methods. All three models display large confidence intervals outside the training range $(x < 0, x > 1)$. However, within the 95% confidence intervals in regions where training data exists $(0 \leq x \leq 1)$, BBB fails to cover data with narrow confidence intervals, whereas MCD exhibits wide and narrow confidence intervals. In contrast, our proposed method can cover most data points within the 95% confidence intervals.

4.4 Activation Quantization

We experimentally explore the number of bits for activation fixed-point quantization in our method with MNIST classification using MLP through quantization aware training [18]. In this experiment, we reduce the number of bits of the activations before and after the batch normalization layer, originally represented as 32-bit floating-point numbers, to a fixed-point representation with a specific number of bits for the model used in MNIST classification. The hyperparameters used in this experiment are the same as those listed in Table 1. Figure 7 shows the relationship between accuracy and activation quantization. The number of activation quantization bits has a minimal impact on inference accuracy for 8 bits and above, with a significant drop in accuracy observed from the 7 bits or fewer. Therefore, 8-bit fixed-point quan-

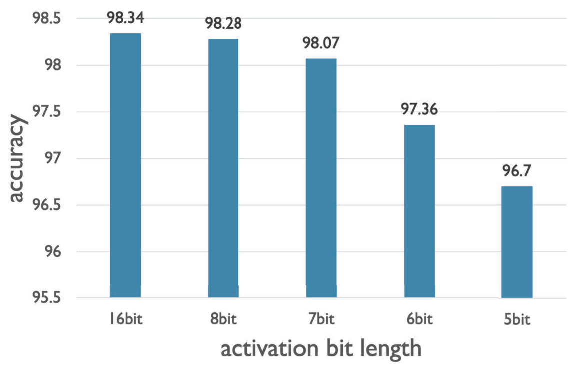


Fig. 7 Relationship between activation bit length and accuracy with quantization aware training.

tization is beneficial for implementing the proposed method on resource-constrained devices. The proposed method has half the parameter size compared to BBB. However, when compared to MCD and vanilla, the sizes are equivalent, and no superiority can be demonstrated. Nevertheless, since the weights are sampled to binary values (-1 or 1), making it possible to replace multiply-accumulate operations with fixed-point 8-bit additions when implemented in hardware.

5. Conclusion

In this study, we propose a method to extend binary NNs to BNNs by interpreting binary weights as probabilities, and investigate their performance across tasks, such as multiclass classification, binary classification, and regression. The proposed method reduces memory consumption by half compared to the BBB method by reducing the weight parameters. In addition, compared to the MCD method, our method offers the advantage of reducing hardware resources by replacing multipliers with adders and enabling faster computations. Additionally, our proposed method could capture uncertainty and achieve similar accuracy as other methods when applied to smaller network sizes. Weight binarization suited for hardware implementation is controllable at the bit-level signal and does not yield full benefits when implemented on GPUs or CPUs. Therefore, as a future work, we plan to implement the proposed algorithm in a digital circuit such as FPGA and confirm the acceleration of operations and reduction of hardware resources. Furthermore, since our approach currently demonstrates lower accuracy and fails to capture uncertainty effectively compared to other methods when applied to larger network sizes, we plan to explore improvement techniques from an algorithmic perspective.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp.770–778, 2016.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[3] C.-Y. Wang, A. Bochkovskiy, and H.-Y.M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 2023 IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR), 2023.

[4] M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. Fieguth, X. Cao, A. Khosravi, U.R. Acharya, V. Makarenkov, and S. Nahavandi, "A review of uncertainty quantification in deep learning: Techniques, applications and challenges," Information Fusion, vol.76, pp.243–297, Dec. 2021.

[5] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," 2015.

[6] Y. Gal and Z. Ghahramani, "Bayesian convolutional neural networks with bernoulli approximate variational inference," 2015.

[7] Q. Wan and X. Fu, "Fast-bcnn: Massive neuron skipping in bayesian convolutional neural networks," 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp.229–240, 2020.

[8] Y. Hirayama, T. Asai, M. Motomura, and S. Takamaeda, "A hardware-efficient weight sampling circuit for bayesian neural networks," International Journal of Networking and Computing, vol.10, no.2, pp.84–93, 2020.

[9] Z. Ghahramani and M. Beal, "Propagation algorithms for variational bayesian learning," Advances in Neural Information Processing Systems, 2000.

[10] D.P. Kingma and M. Welling, "Auto-encoding variational bayes," 2022.

[11] M. Courbariaux, Y. Bengio, and J.P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," 2016.

[12] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013.

[13] L. Deng, "The mnist database of handwritten digit images for machine learning research," IEEE Signal Processing Magazine, vol.29, no.6, pp.141–142, 2012.

[14] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research),"

[15] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.

[16] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A.Y. Ng, "Reading digits in natural images with unsupervised feature learning," NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011, 2011.

[17] C. Guo, G. Pleiss, Y. Sun, and K.Q. Weinberger, "On calibration of modern neural networks," 2017.

[18] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," 2018 IEEE/CVF Conf. Computer Vision and Pattern Recognition, 2018.

Taisei Saito received B.E. in electronic engineering from Hokkaido University, Sapporo, Japan, in 2022. He is currently pursuing his master degree at Graduate School of Information Science and Technology, Hokkaido University. His current research interests include Bayesian Neural Networks and its hardware accelerator for an edge device.

Kota Ando received his B.E. degree in electronics and M.S. degree in information technology from Hokkaido University, Sapporo, Japan, in 2016 and 2018, respectively. He received his Ph.D. degree in engineering from Tokyo Institute of Technology, Yokohama, Japan, in 2021. He is currently an assistant professor at Hokkaido University, Sapporo, Japan. He worked as an assistant professor at Tokyo Institute of Technology, Yokohama, Japan, from 2021 to 2022.His research interests include reconfigurable architectures, memory-centric processing, and hardwareaware algorithms for efficient deep learning processing.

Tetsuya Asai received B.S. and M.S. degrees in electronic engineering from Tokai University, Hiratsuka, Japan, in 1993 and 1996, respectively, and a Ph.D. degree from the Toyohashi University of Technology, Toyohashi, Japan, in 1999. He is currently a Professor with the Graduate School/Faculty of Information Science and Technology, Hokkaido University, Sapporo, Japan. His current research interests include developing intelligent integrated circuits and their computational applications, emerging research architectures, deep learning accelerators, and device-aware neuromorphic very large-scale integrations.