BYNQNet: Bayesian Neural Network with Quadratic Activations for Sampling-Free Uncertainty Estimation on FPGA

Hiromitsu Awano*† and Masanori Hashimoto*
*Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka, 565–0871, Japan
†JST, PRESTO
4-1-8 Honcho, Kawaguchi, Saitama, 332-0012, Japan
Email: awano@ist.osaka-u.ac.jp, hasimoto@ist.osaka-u.ac.jp

***Abstract*—An efficient inference algorithm for Bayesian neural network (BNN) named BYNQNet, Bayesian neural network with quadratic activations, and its FPGA implementation are proposed. As neural networks find applications in mission critical systems, uncertainty estimations in network inference become increasingly important. BNN is a theoretically grounded solution to deal with uncertainty in neural network by treating network parameters as random variables. However, an inference in BNN involves Monte Carlo (MC) sampling, i.e., a stochastic forwarding is repeated N times with randomly sampled network parameters, which results in N times slower inference compared to non-Bayesian approach. Although recent papers proposed sampling-free algorithms for BNN inference, they still require evaluation of complex functions such as a cumulative distribution function (CDF) of Gaussian distribution for propagating uncertainties through nonlinear activation functions such as ReLU and Heaviside, which requires considerable amount of resources for hardware implementation. Contrary to conventional BNN, BYNQNet employs quadratic nonlinear activation functions and hence the uncertainty propagation can be achieved using only polynomial operations. Our numerical experiment reveals that BYNQNet has comparative accuracy with MC-based BNN which requires N=10 forwardings. We also demonstrate that BYNQNet implemented on Xilinx PYNQ-Z1 FPGA board achieves the throughput of 131$\times 10^3$ images per second and the energy efficiency of 44.7$\times 10^3$ images per joule, which corresponds to 4.07$\times$ and 8.99$\times$ improvements from the state-of-the-art MC-based BNN accelerator.**

I. Introduction

Neural networks have recently achieved tremendous success in various fields such as the image recognition with suprehuman precision [1]–[3] and playing Go game [4]. Stimulated by these success, neural networks are now finding applications in critical and safety-sensitive domains such as self-driving cars [5], [6] and flight control [7]. For example, in [8], a neural network is used to directly map raw images captured by front-facing cameras into steering commands.

It is obvious that for such safety critical applications, making model predictions is not enough: we need *uncertainty* of the neural network on generated predictions. For example, for self-driving car to avoid accidents, the model should know what situation has not seen before so that it can return the control to human. Unfortunately, however, conventional neural networks only generate prediction without uncertainty, which is known to be a shortcoming of non-Bayesian neural networks.

To solve this issue, Bayesian neural networks (BNNs) have been developed [9], [10]. Contrary to traditional neural networks having fixed weights, all weights in BNNs have their probability distributions. The network predictions are also represented by a probability distribution with which we can estimate the uncertainty of the outputs, i.e., the wide probability distribution of the output indicates the large uncertainty on it. However, the inference of BNN is very slow since it requires Monte Carlo (MC) sampling, i.e., N forward passes with randomly sampled weights are performed to approximate the statistical distributions of activations [11]–[13]. Therefore, the conventional hardware accelerator for BNN [14] focused on the acceleration of Gaussian random number generator (GRNG) to accelerate the sampling process.

To avoid time-consuming MC sampling, recent researches developed moment propagation algorithms [15], [16]. Here, the input to the activation function (called "pre-activation" hereafter) is a linear combination of the output from neurons (called "post-activation" hereafter) in the preceding layer and the corresponding synaptic weights. Therefore, the pre-activation follows a Gaussian distribution thanks to central limit theorem (CLT) even though the post-activation distributions are complicated due to nonlinear activation functions. Knowing that the network activation approximately follows a Gaussian distribution, we only need to propagate first and second moments of activations through the network since a Gaussian distribution is completely determined by a mean and a variance.

Although the sampling-free techniques in [15] and [16] drastically improved BNN inference time by eliminating the MC sampling, their methods require computations of complex functions such as a cumulative distribution function (CDF) of a Gaussian distribution for propagating moments through nonlinear activation functions. Hence, their methods cannot be directly deployed on resource constrained devices such as field programmable gate arrays (FPGAs).

In this paper, we propose BYNQNet, Bayesian neural network with quadratic activations, which replaces ReLU with a simple quadratic function to enable exact and efficient moment computation and propagation on FPGAs whereas moments are approximately computed for ReLU in [15], [16]. The use of quadratic activations is studied in [17], but the objective was to make operations in neural networks compatible with a fully homomorphic encryption schemes. On the other hand, we exploit the simplicity of the quadratic activations to derive a lightweight algorithm for the moment propagation. As we will show in Sec. III, substituting ReLU with quadratic activations allows us to compute the moment of post-activations only by

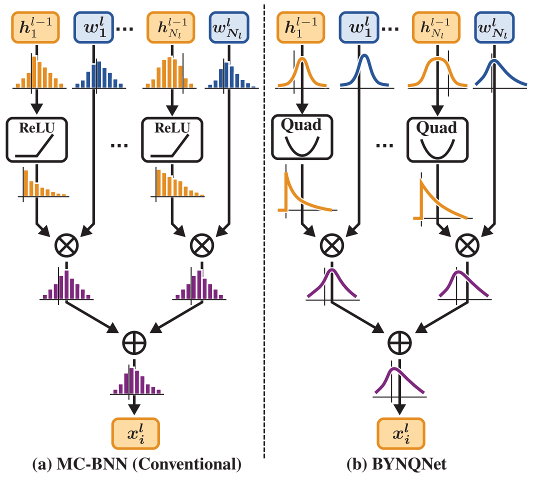

Fig. 1. (a) MC-based BNN and (b) proposed BYNQNet.

polynomial operations, which can be efficiently implemented on FPGAs. The difference between the conventional BNN and BYNQNet is illustrated in Fig. 1.

Our preliminary experiment using software implemented BYNQNet reveals that BYNQNet has comparable performance with an MC-based BNN (MC-BNN) when N=10. Although single forwarding of BYNQNet costs approximately 3× more computations compared with that of MC-BNN, BYNQNet has the advantage that it requires no MC sampling for uncertainty estimation. Hence, in this particular situation, BYNQNet achieved 10/3≈3.3× reduction in computational costs. Furthermore, BYNQNet implemented on PYNQ-Z1 FPGA board demonstrates the throughput of 131×10^3 images per second and the energy efficiency of 44.7×10^3 images per joule, which corresponds 4.07× and 8.47× improvement from the state-of-the-art MC-BNN accelerator in [14].

Followings summarize the contributions of this paper.

- To the best of our knowledge, this is the first FPGA implementation of sampling-free BNN.
- We conduct an exhaustive experiment with software implementation of BYNQNet and demonstrate that the CLT assumption holds for the practical neural network structure and that BYNQNet has comparable performance with MC-BNN when N=10.
- BYNQNet is implemented on PYNQ-Z1 board having 220 DSPs and it achieves 4.07× and 8.47× higher throughput and energy efficiency compared to the state-of-the-art MC-BNN accelerator proposed in [14].

The rest of this paper is organized as follows. In Sec.II, we provide the preliminaries required to introduce BYNQNet, followed by the detail of BYNQNet in Sec.III. The software implementation of BYNQNet is compared with MC-BNN in Sec. IV. Then, in Sec. V, BYNQNet on PYNQ-Z1 board is compared with the state-of-the-art MC-BNN accelerator proposed in [14]. Finally, concluding remarks are provided in Sec.VI.

II. Preliminaries

A. Bayesian Neural Network

Bayesian neural network (BNN) is an extension of neural network that can model uncertainties of predictions. For a general Bayesian model, we are interested in finding the posterior distribution over the network weights, $\mathbf{W} = (\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L)$, given the training data, $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_M)$, and the target label, $\boldsymbol{Y} = (y_1, y_2, \cdots, y_M)$. Here, $\boldsymbol{w}^l$ is the $N_l \times N_{l-1}$ weight matrix of l-th layer having N_{l-1} input and N_l output nodes, $\boldsymbol{x}_l$ is a D-dimensional vector, and M is the number of training samples. By using the Bayes' theorem, the posterior distribution can be represented by the combination of the likelihood and the prior distribution:

$$P(\boldsymbol{W}|\boldsymbol{X}) = \frac{P(\boldsymbol{X}|\boldsymbol{W})P(\boldsymbol{W})}{P(\boldsymbol{X})}. \tag{1}$$

In practical applications, this posterior distribution is not tractable and hence the *variational inference* technique has been developed to approximate the posterior distribution, i.e., $P(\boldsymbol{W}|\boldsymbol{X}) \approx q(\boldsymbol{W}|\boldsymbol{\theta})$, where $q(\cdot|\cdot)$ is a variational posterior distribution and $\boldsymbol{\theta}$ is variational parameters.

For simplicity, a Gaussian distribution is usually employed for the variational posterior distribution. Hence, an (i,j)-element of $\boldsymbol{w}^l$ can be obtained by

$$w_{ij}^l = \mu_{ij}^l + \epsilon \cdot \log\left(1 + \exp\left(\rho_{ij}^l\right)\right), \tag{2}$$

where ϵ is a random variable sampled from the unit Gaussian distribution, and μ_{ij}^l and ρ_{ij}^l are the variational parameters. During test time, we perform the forward propagation repeatedly with randomly sampled network weights to approximate the statistical distribution of network outputs:

$$P(y|\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y}) \approx \frac{1}{N}\sum_{i=1}^{N} g(y, \boldsymbol{x}, \boldsymbol{W}^{(i)}), \tag{3}$$

where $\boldsymbol{x}$ and y are network input and output, respectively, and $\boldsymbol{W}^{(i)}$ is i-th sample drawn from $q(\boldsymbol{W}|\boldsymbol{\theta})$. Using generated samples, we can evaluate the uncertainty of the classification by several metrics such as the entropy defined by

$$H[y|\boldsymbol{x},\boldsymbol{X},\boldsymbol{Y}] = -\sum_c P(y=c|\boldsymbol{x},\boldsymbol{X},\boldsymbol{Y}) \log P(y=c|\boldsymbol{x},\boldsymbol{X},\boldsymbol{Y}). \tag{4}$$

BNN is capable of reporting the uncertainty in the network output, which is the distinct advantage of BNN, but it involves a large amount of computation since the forward propagation is repeated N times to make each prediction. This means that BNN is N times slower compared to the conventional neural network having the same network structure, which is unacceptable for real-time applications such as self-driving cars.

B. Moment Propagation

To eliminate the time consuming MC sampling, several recent works have developed moment propagation algorithms [15], [16]. We briefly review the algorithm using a single-hidden layer neural network. At the l-th layer, the network computes

$$x_i^l = b_i^l + \sum_{j=1}^{N_{l-1}} w_{ij}^l h_j^{l-1}, \tag{5}$$

where $\boldsymbol{b}^l = (b_1^l, b_2^l, \cdots, b_{N_l}^l)$ is the bias of the neuron and $\boldsymbol{x}^l = (x_1^l, x_2^l, \cdots, x_{N_l}^l)$ and $\boldsymbol{h}^{l-1} = (h_1^{l-1}, h_2^{l-1}, \cdots, h_{N_{l-1}}^{l-1})$ are the l-th layer pre-activations and $l-1$-th layer post-activations, respectively. The post-activations are computed by using a nonlinear function $\phi(\cdot)$, i.e., $h_j^{l-1} = \phi(x_j^{l-1})$. In Eq. (5), we notice that x_i^l is a sum of independent and identically distributed (i.i.d) terms since weights are taken to be i.i.d and the post-activations, h_j^{l-1} and $h_{j'}^{l-1}$, are independent for $j \neq j'$. Hence, according to the central limit theorem (CLT), x_i^l follows a Gaussian distribution at the limit of infinite neurons, i.e., $N_{l-1} \to \infty$.

Knowing that x_i^l approximately follows a Gaussian distribution, all we need to compute is the first and second moments (mean and variance) of activations since a Gaussian distribution is fully described by the mean and variance. By modeling w_{ij}^l and h_j^{l-1} as independent random variables, means and variances of post-activations can be represented as:

$$\mathbb{E}[x_i^l] = b_i^l + \sum_{j=1}^{N_{l-1}} \mathbb{E}[w_{ij}^l]\mathbb{E}[h_j^{l-1}], \tag{6}$$

$$\mathbb{V}[x_i^l] = \sum_{j=1}^{N_{l-1}} \left\{ \left(\mathbb{V}[w_{ij}^l] + \mathbb{E}[w_{ij}^l]^2\right)\mathbb{V}[h_j^{l-1}] + \mathbb{V}[w_{ij}^l]\mathbb{E}[h_j^{l-1}]^2 \right\}. \tag{7}$$

Here, $\mathbb{E}[x]$ and $\mathbb{V}[x]$ are the mean and variance of the random variable x. Note here that the distributions over network parameters are fixed during test time and that $\mathbb{E}[w_{ij}^l]$ and $\mathbb{V}[w_{ij}^l]$ are precomputable constants. Hence, by recursively applying Eqs. (6) and (7), we can compute the first and second moment of the final layer activations as a deterministic function of the moments of network inputs.

The remaining difficult part is the moment propagation through the nonlinear activation function, $\phi(\cdot)$. A closed-form approximation of $\mathbb{E}[\phi(x_i^l)]$ and $\mathbb{V}[\phi(x_i^l)]$ are proposed for ReLU activations in [15]. However, the approximation still contains complex functions such as a CDF of a Gaussian distribution, which requires huge circuit area when implemented on hardware.

III. Proposed Method

A. Quadratic Nonlinearity

In this section, we propose new quadratic activation functions to enable simple yet exact moment propagation through activation functions. Firstly, we employ the following quadratic polynomial as the nonlinearity:

$$\phi(x_i^l) = (x_i^l)^2. \tag{8}$$

Then, the first moment of the post-activation is represented as a function of the first and second moments of the pre-activations as follows:

$$\mathbb{E}\left[h_i^l\right] = \mathbb{E}[(x_i^l)^2] = \mathbb{V}[x_i^l] + \mathbb{E}[x_i^l]^2. \tag{9}$$

For the second moment of the post-activation, we have

$$\mathbb{V}[h_i^l] = \mathbb{V}[(x_i^l)^2] = \mathbb{E}\left[\left((x_i^l)^2 - \mathbb{E}[(x_i^l)^2]\right)^2\right]$$
$$= \mathbb{E}\left[(x_i^l)^4\right] - \mathbb{E}\left[(x_i^l)^2\right]^2. \tag{10}$$

Then, using the fact that the fourth moment of Gaussian distributed random variable X, $\mathbb{E}[X^4]$, is given by $\mathbb{E}[X]^4 + 6\mathbb{E}[X]^2\mathbb{V}[X] + 3\mathbb{V}[X]^2$, we have

$$\mathbb{V}\left[h_i^l\right] = 2\mathbb{V}[x_i^l]\left(\mathbb{V}[x_i^l] + 2\mathbb{E}[x_i^l]^2\right). \tag{11}$$

Eqs. (9) and (11) tell us that the first and second moments of post-activations can be represented as the polynomial function of $\mathbb{E}[x_i^l]$ and $\mathbb{V}[x_i^l]$. Hence, owing to the quadratic nonlinearity, the moment propagation through nonlinear activation functions can be computed using only simple polynomial operations.

Algorithm 1 Moment propagation in BYNQNet.

```
Input:
  $\mathbb{E}[\boldsymbol{h}^{l-1}]$ and $\mathbb{V}[\boldsymbol{h}^{l-1}]$   ▷ $N_{l-1}$-dimensional vectors of means and variances of $l$–1-th layer post-activations
  $\widehat{\boldsymbol{W}}_\mu^l$, $\widehat{\boldsymbol{W}}_v^l$, and $\widehat{\boldsymbol{W}}_{\mu^2}^l$   ▷ $N_l \times N_{l-1}$ matrices whose $(i,j)$-elements are given by Eqs. (15), (16), and (17), respectively
  $\widehat{\boldsymbol{b}}^l$   ▷ $N_l$-dimensional vector whose elements are given by Eq. (18)
Output:
  $\mathbb{E}[\boldsymbol{h}^l]$ and $\mathbb{V}[\boldsymbol{h}^l]$   ▷ $N_l$-dimensional vectors of means and variances of $l$-th layer post-activations
$\boldsymbol{m} \leftarrow \widehat{\boldsymbol{b}}^l + \widehat{\boldsymbol{W}}_\mu^l \mathbb{E}[\boldsymbol{h}^{l-1}]$
$\boldsymbol{v} \leftarrow \widehat{\boldsymbol{W}}_v^l \mathbb{V}[\boldsymbol{h}^{l-1}] + \widehat{\boldsymbol{W}}_{\mu^2}^l \left(\mathbb{E}[\boldsymbol{h}^{l-1}] \circ \mathbb{E}[\boldsymbol{h}^{l-1}]\right)$
$\mathbb{E}[\boldsymbol{h}^l] \leftarrow \boldsymbol{v} + \boldsymbol{m} \circ \boldsymbol{m}$
$\mathbb{V}[\boldsymbol{h}^l] \leftarrow 2\boldsymbol{v} \circ (\boldsymbol{v} + 2\boldsymbol{m} \circ \boldsymbol{m})$
```

We specifically call the BNN having the proposed quadratic activation function as Bayesian neural network with quadratic activations (BYNQNet).

B. Fused Batch Normalization for BYNQNet

Batch normalization (BN) is an indispensable technique for modern neural networks to stabilize network training as well as to improve the generalization capability. BN transforms the layer output, x_i^l, as follows:

$$\widehat{x}_i^l = \gamma_i^l \frac{x_i^l - \mu_i^l}{\sqrt{(\sigma_i^l)^2 + \epsilon}} + \beta_i^l, \tag{12}$$

where $\widehat{x}_i^l$ is the activations after BN, ϵ is a small constant for numerical stability, and γ_i^l and β_i^l are trainable parameters. μ_i^l and σ_i^l are the mean and variance of a batch, which will be recomputed for each epoch during training.

Since γ_i^l, β_i^l, μ_i^l, and σ_i^l are fixed during test time, BN is essentially identical to a linear transformation and hence BN can be fused with the weight matrix multiplication:

$$\mathbb{E}[\widehat{x}_i^l] = \widehat{b}_i^l + \sum_{j=1}^{N_l} \widehat{w}_{\mu,ij}^l \mathbb{E}[h_j^{l-1}], \tag{13}$$

$$\mathbb{V}[\widehat{x}_i^l] = \sum_{j=1}^{N_l} \left\{ \widehat{w}_{v,ij}^l \mathbb{V}[h_j^{l-1}] + \widehat{w}_{\mu^2,ij}^l \mathbb{E}[h_j^{l-1}]^2 \right\}, \tag{14}$$

where $\widehat{w}_{\mu,ij}^l$, $\widehat{w}_{v,ij}^l$, $\widehat{w}_{\mu^2,ij}^l$, and $\widehat{b}_i^l$ are the precomputable parameters defined as:

$$\widehat{w}_{\mu,ij}^l = \frac{\gamma_i^l}{\sqrt{(\sigma_i^l)^2 + \epsilon}} \mathbb{E}[w_{ij}^l], \tag{15}$$

$$\widehat{w}_{v,ij}^l = \frac{(\gamma_i^l)^2}{(\sigma_i^l)^2 + \epsilon} \left(\mathbb{V}[w_{ij}^l] + \mathbb{E}[w_{ij}^l]^2\right), \tag{16}$$

$$\widehat{w}_{\mu^2,ij}^l = \frac{(\gamma_i^l)^2}{(\sigma_i^l)^2 + \epsilon} \mathbb{V}[w_{ij}^l], \tag{17}$$

$$\widehat{b}_i^l = \beta_i^l + \gamma_i^l \frac{b_i^l - \mu_i^l}{\sqrt{(\sigma_i^l)^2 + \epsilon}}. \tag{18}$$

Algorithm 1 summarizes the final algorithm for propagating moments through single hidden layer of BYNQNet, where $\circ$ represents the element-wise product.

C. Hardware Implementation

1) Design Flow: The overall design flow of implementing BYNQNet on PYNQ-Z1 board is shown in Fig. 2. First, BYNQNet is implemented and trained using TensorFlow framework. To fully exploit the automated gradient computation

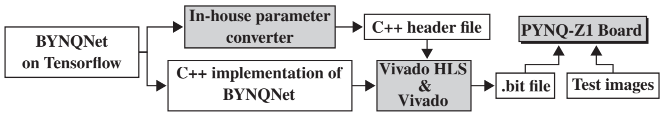

Fig. 2. Design flow of BYNQNet accelerator.

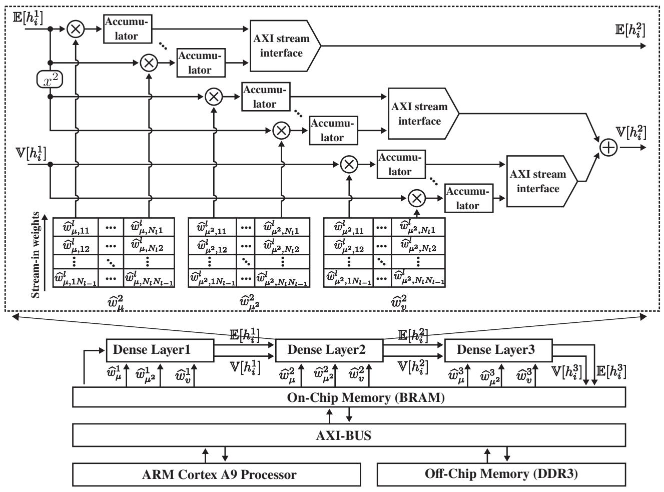

Fig. 3. Proposed BYNQNet accelerator.

provided by TensorFlow, we used reparameterization trick for model training, which suggests to sample unit Gaussian random variable, followed by scaling and shifting so that gradients can propagate through scaling and shifting variable [18]. After the training, the means and variances of weights are extracted as a NumPy array format. Then, our in-house parameter converter computes Eqs. (15), (16), (17), and (18) and formats the results as a C++ header file to be stored in on-chip BRAMs.

The hardware architecture of BYNQNet is designed using C++ language. The native C++ code uses floating-point format to represent weight and activations, which comes at a great cost when implemented on hardware [19]. Hence, we replace floating-point values with low-bit fixed-point values. To simulate the behaviour of fixed-point arithmetics in software, we adopt "ap_fixed" template class provided by Xilinx. Then, Xilinx Vivado toolchain is used to generate a ".bit" file which is used to configure an FPGA. To control the BYNQNet on FPGA, we use PYNQ framework, with which the FPGA accelerator can be controlled via Python scripting language.

2) Overall Architecture: The overall architecture of BYNQNet accelerator is shown in Fig. 3, which consists of on-chip BRAMs for storing network parameters and three processing elements (PEs) each of which corresponds to a layer of BYNQNet. We employed a streaming pipelining architecture to increase the inference throughput. For the communications between each layer, we used data streaming mechanism provided by Xilinx, where data samples are transferred in a sequential order starting from the first sample. The precomputed weights which are ready to participate in the inference are allocated in on-chip BRAMs and hence only input and output of the network are stored in off-chip DDR memory connected via AXI streaming interface.

3) PE Design: As shown in the upper part of Fig. 3, a PE consists of three dedicated multiplication and accumulation

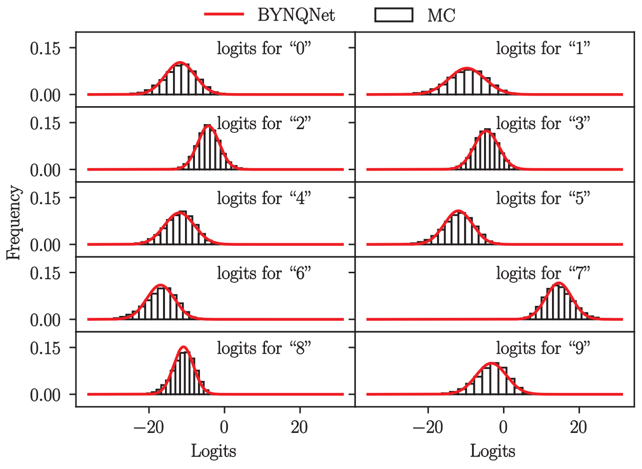

Fig. 4. Distributions of the final layer pre-activations reported by BYNQNet and MC.

units each of which is responsible for computing Eq. (13) and the first and second terms of Eq. (14), respectively. Each PE loads the means and variances of the previous layer activations serially, computes multiplications by the precomputed weights, accumulates the partial sum, and sends the accumulated results to the succeeding PE. Note again that each PE has a dedicated on-chip memory for storing precomputed parameters so that every PE can operate in parallel.

IV. Validation with Software Implementation

A. Experimental Setup

Before implementing BYNQNet on an FPGA platform, we conduct a numerical experiment to compare the accuracy of BYNQNet with MC-BNN employing ReLU activation function. For this purpose, we implement BYNQNet and the MC-BNN by using TensorFlow framework and train both on MNIST dataset [20]. MNIST is a collection of 70k images of 28×28 pixel gray-scale handwritten digits. Among 70k images, 60k images are used to train a network while the rest of 10k images are used to test the trained network. Both BYNQNet and MC-BNN implemented have 784 inputs, 2 hidden layers each of which has 200 neurons, and 10 outputs. All layers are densely connected.

B. Accuracy of Moment-Propagation

As mentioned in Sec. III, BYNQNet is based on CLT, which assumes that (1) the number of neurons in hidden layers (N_l) are large enough and that (2) each post-activation is stochastically independent. However, in the practical situation, a hidden layer typically contains only hundreds of neurons and there are weak correlations among post-activations, which may violate the assumption of CLT.

To investigate the error induced by those factors, we compare the pre-activation distributions of the final layer reported by BYNQNet with that estimated by MC using 10k trials and summarize the result in Fig. 4. We can see the perfect agreement of BYNQNet with MC, which provides evidence that the assumptions of CLT hold for practical situations.

C. Impact of Quadratic Activation Functions on Performance

We further investigate the performance degradation caused by replacing ReLU with quadratic function. Fig. 5(a) compares

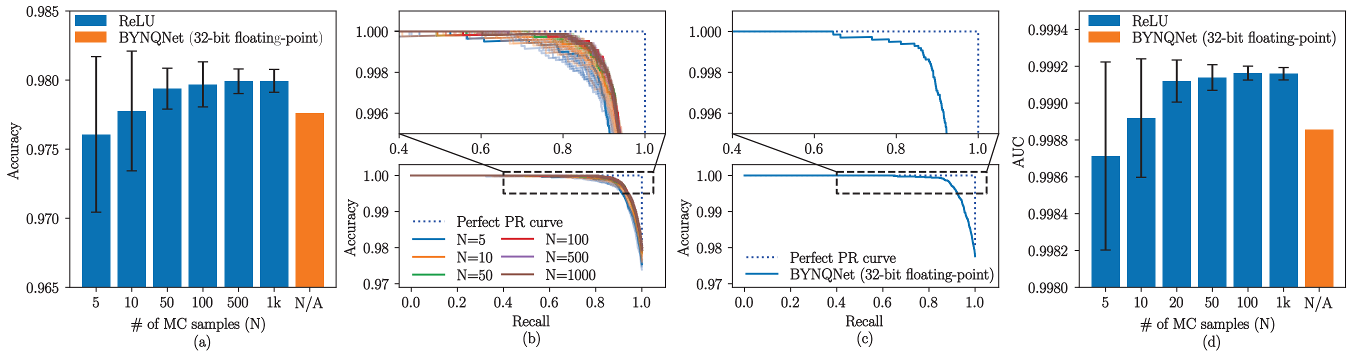

Fig. 5. Performance comparison of MC-BNN employing ReLU and BYNQNet. (a) Testing accuracy comparison. Precision recall curve of MC-BNN (b) and that of BYNQNet (c). (d) Area under the precision-recall curves.

the testing accuracy of BYNQNet with that of the MC-BNN for different N. Note again that, for a fair comparison, both networks have exactly the same structure except the nonlinear activation functions, i.e., BYNQNet has quadratic activation function while MC-BNN has ReLU. For MC-BNN, the same experiment is repeated for 20 times to obtain the 95% confidence intervals which are again shown in the same figure. Fig. 5(a) shows that the testing accuracy improves by increasing N and that small N (e.g. N=10) attains a reasonable classification accuracy, which is consistent with the observation in [11]. We also notice that the accuracy of BYNQNet is comparable to that of MC-BNN when N=10.

We remind the reader that MC-BNN needs to perform N stochastic forwarding through the network with randomly sampled network weights while BYNQNet requires only a single forwarding during which the first and second moments are propagated to obtain the network output as well as the uncertainty. It should be noted that the single forwarding of BYNQNet needs approximately $3\times$ computation compared with MC-BNN since the formulas of BYNQNet, Eqs (13) and (14), involve computations of three matrix multiplications. Hence, in this particular situation, BYNQNet achieved approximately $3.3\times$ reduction in computation while maintaining the testing accuracy.

D. Uncertainty Estimation

We further investigate the reliability of the estimated uncertainty by using the precision-recall (PR) metric [21]. First, we compute the uncertainty for each classification with Eq. (4) and rank images according to the associated uncertainty. Then, the testing accuracy is evaluated only for α-portion of images having lower uncertainty, where the others are discarded. The curve shows how classification accuracy changes as we discard images with uncertainty higher than different percentile thresholds.

Figs. 5(b) and (c) show the PR curves of MC-BNN and BYNQNet. For MC-BNN, the PR curves for 20 experiments are shown. Also, the PR curve of the perfect classifier (corresponding to 100% precision and 100% recall) is shown for the reference. In Fig. 5(b), we notice that by increasing N, the PR curve gets gradually closer to that of the perfect classifier.

To quantitatively compare the reliabilities of reported uncertainties, we compute the area under each PR curve (AUC) and summarize the result in Fig. 5(d). The error bars again show the 95% confidence intervals. A high AUC indicates that a classifier outputs accurate results while suppressing the misclassification and hence the AUC of the perfect classifier will be 1.0. Studying Fig. 5(d), we again see that the AUC of BYNQNet is comparable to that of MC-BNN when N=10. Hence, we can say again that BYNQNet achieves $3.3\times$ reduction in computations while maintaining the quality of uncertainty estimation.

V. Hardware Implementation and Measurement

BYNQNet used in Sec. IV is implemented on Xilinx PYNQ-Z1 board which embeds Zynq XC7Z020 SoC containing 53,200 LUTs, 220 DSP slices, and 630KB BRAMs along with a Cortex-A9 processor.

A. Bit-Length Optimization

To identify an optimal fixed-point bit-length of operands, an exhaustive experiment is conducted. Fig. 6(a) compares the testing accuracies of BYNQNet implemented using different fixed-point bit-lengths with that of the 32-bit floating-point model. We notice that by increasing the bit-length, the testing accuracy quickly reaches to that of the floating-point model and that 7-bit is sufficient to achieve the 0.1% or less error.

Fig. 6(b) compares PR curves of fixed-point BYNQNet with that of the floating-point BYNQNet. We can again see that the PR curve of 7-bit fixed-point model is comparable to that of the floating-point model. Fig. 6(c), which compares AUCs of the fixed-point models and the floating-point model, also demonstrates that the 7-bit fixed-point model is comparable to the floating-point model.

Considering those observations, we choose 7-bit fixed-point model to reduce the hardware footprint while maintaining the accuracy.

B. Measurement

Using Vivado toolchain, we convert the BYNQNet employing 7-bit fixed-point representation into ".bit" file. We measured power consumption of the whole PYNQ-Z1 board using a USB power meter as shown in Fig. 7. During measurement, 10k test images are continuously applied to the PYNQ-Z1 board to observe the averaged power consumption.

We compare BYNQNet with the state-of-the-art MC-BNN accelerator [14] and summarize the result in Tab. I. Since [14] used a Cyclone V FPGA, the direct comparison in resource utilization is difficult and hence we here focus on

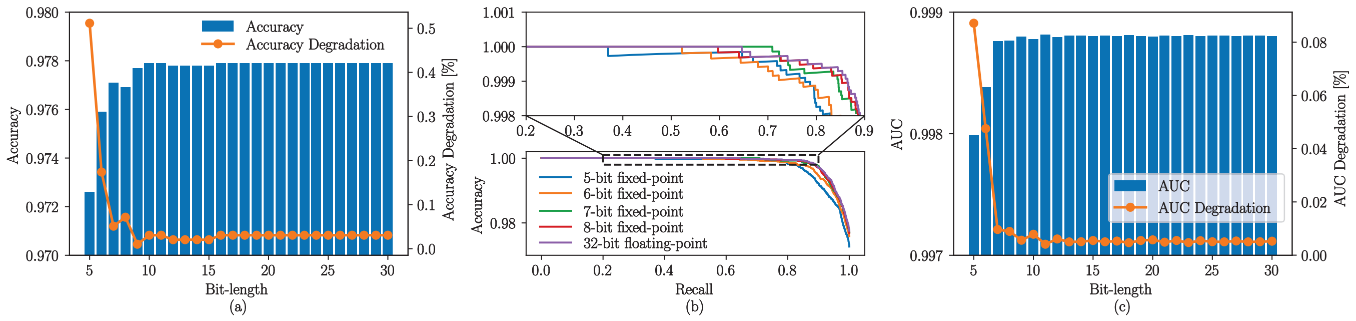

Fig. 6. The impact of fixed-point bit-length on (a) the classification accuracy, (b) PR curve, and (c) AUC.

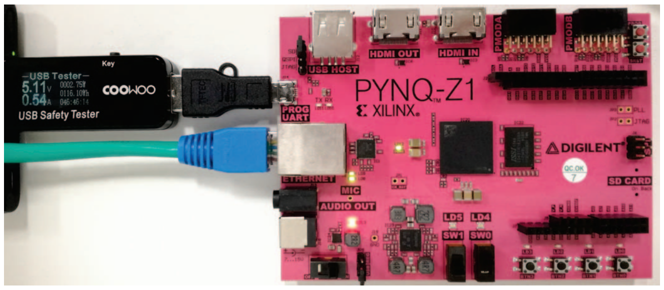

Fig. 7. Power measurement setup.

TABLE I
COMPARISON WITH THE STATE-OF-THE-ART MC-BNN ACCELERATOR.

	VIBNN [14]	BYNQNet
FPGA	Cyclone V 5CGTFD9E5F35C7	Zynq XC7Z020
Clock (MHz)	212.95	200
# of LUTs	–	37102 / 53200 (70.0%)
# of ALMs	98006 / 113560 (86.3%)	–
# of DSPs	342 / 342 (100%)	220 / 220 (100%)
Registers	88720	43268
BRAM [kB]	558.2 / 1525 (36.6%)	220.5 / 630 (35.0%)
Throughput (Images/s)	322×10^3 (N=1) 32.2×10^3 (N=10)	131×10^3
Energy (Images/J)	52.7×10^3 (N=1) 5.27×10^3 (N=10)	47.4×10^3

the throughput and the energy efficiency. As shown in Tab. I, the single forwarding of BYNQNet is more time- and energy-consuming than VIBNN. However, as discussed in Sec. IV, MC-BNN requires at least N=10 MC trials to obtain a reliable uncertainty report, whereas BYNQNet requires only single forwarding. Thus, we can conclude that BYNQNet achieves 4.07× and 8.99× higher throughput and energy efficiency, respectively.

VI. CONCLUSION

BYNQNet to accelerate BNN inference on FPGA was proposed. By replacing ReLU activation functions with quadratic ones, we derived the moment propagation algorithm requiring only polynomial operators, which is suitable for hardware implementation. Our exhaustive experiment revealed that BYNQNet had comparable performance in terms of the classification accuracy and the quality of reported uncertainty with MC-BNN employing ReLU. Furthermore, we implemented BYNQNet on PYNQ-Z1 board, and demonstrated that BYNQNet on FPGA achieved 4.07× and 8.99× higher throughput and energy efficiency compared to the state-of-the-art MC-BNN accelerator [14] without deteriorating reliability of reported uncertainty.

ACKNOWLEDGMENT

This work was supported by JST, PRESTO Grant No. JPMJPR18M1 and JSPS KAKENHI Grant No. 18K13800. The authors also would like to acknowledge Xilinx University Program.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Int. Conf. on Neural Information Process. Syst.*, 2012, pp. 1097–1105.
[2] C. Szegedy, Wei Liu *et al.*, "Going deeper with convolutions," in *Conf. on Comput. Vision and Pattern Recognition*, June 2015, pp. 1–9.
[3] K. He, X. Zhang *et al.*, "Deep Residual Learning for Image Recognition," in *Conf. on Comput. Vision and Pattern Recognit.*, June 2016, pp. 770–778.
[4] D. Silver, A. Huang *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.
[5] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," in *Intell. Vehicles Symp.*, June 2017, pp. 1856–1860.
[6] F. Codevilla, M. Miiller *et al.*, "End-to-End Driving Via Conditional Imitation Learning," in *Int. Conf. on Robotics and Automation*, May 2018, pp. 1–9.
[7] H. Baomar and P. J. Bentley, "Autonomous navigation and landing of large jets using Artificial Neural Networks and learning by imitation," in *Symp. Series on Comput. Intell.*, Nov 2017, pp. 1–10.
[8] M. Bojarski, D. D. Testa *et al.*, "End to End Learning for Self-Driving Cars," *CoRR*, vol. abs/1604.07316, 2016.
[9] J. S. Denker and Y. leCun, "Transforming Neural-net Output Levels to Probability Distributions," in *Neural Information Process. Syst.*, 1990, pp. 853–859.
[10] D. J. C. MacKay, "A Practical Bayesian Framework for Backpropagation Networks," *Neural Comput.*, vol. 4, no. 3, pp. 448–472, May 1992.
[11] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," in *Int. Conf. on Machine Learn.*, 2016, pp. 1050–1059.
[12] Y. Gal and Z. Ghahramani, "Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference," in *Int. Conf. on Learn. Representations*, 2016.
[13] P. Huang, W. T. Hsu *et al.*, "Efficient Uncertainty Estimation for Semantic Segmentation in Videos," in *Eur. Conf. on Comput. Vision*, Sept. 2018.
[14] R. Cai, A. Ren *et al.*, "VIBNN: Hardware Acceleration of Bayesian Neural Networks," in *Int. Conf. on Architectural Support for Program. Lang. and Operating Syst.*, 2018, pp. 476–488.
[15] A. Wu, S. Nowozin *et al.*, "Fixing Variational Bayes: Deterministic Variational Inference for Bayesian Neural Networks," *CoRR*, vol. abs/1810.03958, 2018.
[16] M. Haußmann, F. A. Hamprecht, and M. Kandemir, "Sampling-Free Variational Inference of Bayesian Neural Networks by Variance Backpropagation," in *Conf. on Uncertainty in Artif. Intell.*, 2019.
[17] N. Dowlin, R. Gilad-Bachrach *et al.*, "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy," in *Int. Conf. on Machine Learn.*, 2016, pp. 201–210.
[18] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *Int. Conf. on Learn. Representations*, 2014.
[19] K. Guo, S. Zeng *et al.*, "A survey of FPGA based neural network accelerator," *CoRR*, vol. abs/1712.08934, 2017.
[20] Y. LeCun, C. Cortes, and C. J. Burges, "Mnist handwritten digit database." [Online]. Available: http://yann.lecun.com/exdb/mnist/
[21] A. Kendall and Y. Gal, "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" in *Neural Information Process. Syst.*, 2017, pp. 5574–5584.