Efficient Computation Reduction in Bayesian Neural Networks Through Feature Decomposition and Memorization

Xiaotao Jia, *Member, IEEE*, Jianlei Yang, *Member, IEEE*, Runze Liu, Xueyan Wang, *Member, IEEE*, Sorin Dan Cotofana, *Fellow, IEEE*, and Weisheng Zhao, *Fellow, IEEE*

***Abstract*— The Bayesian method is capable of capturing real-world uncertainties/incompleteness and properly addressing the overfitting issue faced by deep neural networks. In recent years, Bayesian neural networks (BNNs) have drawn tremendous attention to artificial intelligence (AI) researchers and proved to be successful in many applications. However, the required high computation complexity makes BNNs difficult to be deployed in computing systems with a limited power budget. In this article, an efficient BNN inference flow is proposed to reduce the computation cost and then is evaluated using both software and hardware implementations. A feature decomposition and memorization (`DM`) strategy is utilized to reform the BNN inference flow in a reduced manner. About half of the computations could be eliminated compared with the traditional approach that has been proved by theoretical analysis and software validations. Subsequently, in order to resolve the hardware resource limitations, a memory-friendly computing framework is further deployed to reduce the memory overhead introduced by the `DM` strategy. Finally, we implement our approach in Verilog and synthesize it with a 45-nm FreePDK technology. Hardware simulation results on multilayer BNNs demonstrate that, when compared with the traditional BNN inference method, it provides an energy consumption reduction of 73% and a 4× speedup at the expense of 14% area overhead.**

***Index Terms*— Bayesian neural network (BNN), computation reduction, feature decomposition, memory reduction.**

Manuscript received June 21, 2019; revised December 17, 2019; accepted April 2, 2020. Date of publication May 6, 2020; date of current version April 5, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61602022 and Grant 61701013, in part by the State Key Laboratory of Software Development Environment under Grant SKLSDE-2018ZX-07, in part by the National Key Technology Program of China under Grant 2017ZX01032101, in part by the CCF-Tencent IAGR20180101, in part by the State Key Laboratory of Computer Architecture under Grant CARCH201917, and in part by the 111 Talent Program under Grant B16001. *(Corresponding authors: Jianlei Yang; Weisheng Zhao.)*

Xiaotao Jia, Xueyan Wang, and Weisheng Zhao are with the School of Microelectronics, BDBC, Fert Beijing Research Institute, Beihang University, Beijing 100191, China, and also with the Beihang-Goertek Joint Microelectronics Institute, Qingdao Research Institute, Beihang University, Qingdao 266101, China (e-mail: weisheng.zhao@buaa.edu.cn).

Jianlei Yang and Runze Liu are with the School of Computer Science and Engineering, BDBC, Fert Beijing Research Institute, Beihang University, Beijing 100191, China (e-mail: jianlei@buaa.edu.cn).

Sorin Dan Cotofana is with the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2628 CD Delft, The Netherlands.

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2020.2987760

I. Introduction

DEEP learning paradigm has created the premises for the development of several deep neural network (DNN) models [1]–[3]. DNNs have been utilized as underlying implementation tools to boost various applications, e.g., computer vision [4], natural language processing [5], speech recognition [6], and autonomous driving [7]. They made DNN research as one of the most active artificial intelligence (AI) branches. However, even though practical utilization of DNNs provides very promising results, e.g., in some cases, DNNs could even outperform human capabilities [8], their global proliferation is mainly impeded by the lack of proper theoretical justification. DNN training is an optimization procedure, which relies on the maximum likelihood estimation (MLE) of the synaptic weights. However, it is well understood and accepted that MLE is not able to properly handle the inherent weights uncertainties. This implies that, from a practical standpoint, MLE-based training is susceptible to overfitting, as experimentally observed in many DNN behaviors [9]. Furthermore, DNNs have other disadvantages, such as data hungry, lack of solid mathematical foundations, and easy to be fooled [10].

To address these shortcomings, Bayesian methods have been introduced by providing mathematically grounded approaches. Bayesian methods could achieve reasonable learning accuracy from small data sets and exhibit the required robustness to address the overfitting issue [11]. More importantly, they could inherently deal with real-world uncertainty and consider prior knowledge. Thus, the potential combination of the complementary strengths of Bayesian methods and DNNs is receiving increasing interest and Bayesian neural networks (BNNs) have been applied in many different applications [12]–[15]. Several probabilistic programming frameworks have been developed, such as Edward [16], Pyro [17], and Zhusuan [18], which could provide efficient implementations for Bayesian deep learning (BDL).

However, the remarkable DNN capabilities can only be exploited at the expense of a high computation complexity associated with the deep layer structure, which requires the utilization of high-performance computing resources to accelerate the training and inference procedures. To resolve this, numerous approaches have been proposed to reduce the

computation cost using, e.g., network pruning [19], low-rank approximation [20], and structure sparsity learning [21]. However, those have rather limited impact as DNN models have been widely applied in Internet-of-Things (IoT) and embedded systems, which are usually computation resource and power consumption confined. Aiming to enable the DNN technology on such devices, the network inference tasks are usually performed on highly parallel and specialized application-specific integrated circuit (ASIC) hardware [22]. As for BNN, a field-programmable gate array (FPGA)-based accelerator has been recently proposed in [23].

In view of the above, a novel BNN inference approach that could reduce computation complexity is proposed in this article. It relies on a new mathematical formulation that enables computation reuse and consequently has a significant impact on computation latency and energy consumption. This article only focuses on the inference part as we can assume that the training is not done in-place for IoT applications (actually, the Edward framework is adopted for BNN training in this article), and thus, it does not affect the available power and computation resources budget. The main contributions of this article are summarized as follows.

1) We introduce a feature decomposition and memorization (DM) approach applicable for BNNs whose parameters obey the Gaussian distribution. The DM strategy takes advantage of single-layer BNN equation characteristics to perform fast and energy effective inference and theoretically eliminates nearly half of the traditionally required computations.
2) In order to apply DM strategy to multilayer BNNs, two methods are further introduced, which are regarded as Hybrid-BNN and DM-BNN, respectively. Both of them could achieve reasonable accuracy with less energy and less runtime.
3) A memory-friendly computing framework is proposed to mitigate the DM-associated memory overhead without any detrimental effect on the inference computation cost and result quality.
4) The proposed approaches and the traditional BNN inference have been evaluated using both software and hardware implementations. The software implementation indicates that the proposed DM-BNN could reduce the computation cost by 85% at the cost of very slight accuracy loss. The hardware implementation suggests that DM-BNN reduces the energy consumption by 73% and achieves a 4× speedup at the expense of 14% area overhead and a minor accuracy degradation.

The rest of this article is organized as follows. Section II discusses some preliminaries and related works. Section III demonstrates the proposed DM strategy. The memory-friendly computing framework is presented in Section IV. Experimental results are illustrated in Section V. Conclusion is drawn in Section VI.

II. Preliminaries

In this section, the BNN background is reviewed and some related works are briefly discussed.

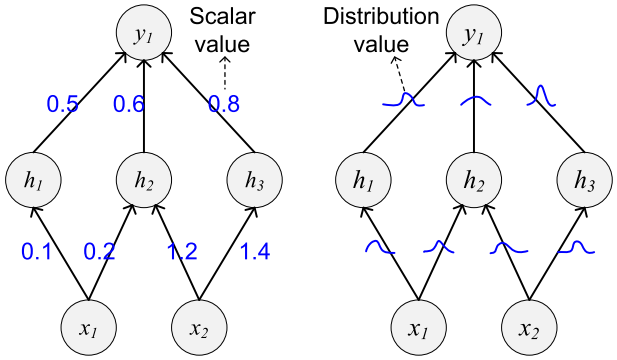

Fig. 1. Different from deterministic neural networks (left) that have scalar weight values and BNNs (right) that the weight values defined by distributions.

By introducing the augmentation of standard neural networks with posterior inference, BNNs aim to create a deep learning framework that can cope with parameter uncertainty. Different from DNNs whose weight values are deterministic, BNNs offer a probabilistic interpretation of deep learning models by inferring weight value distributions, as graphically shown in Fig. 1. BNNs place a prior distribution over each neural network's parameter, and the likelihood (i.e., the training data) is then fed into the network aiming to find the optimal posterior distribution. Usually, BNN posterior distributions fit the Gaussian profile.

To build a BNN model, we denote the BNN by the $f^{\mathcal{W}}(\cdot)$ function, where f represents the network structure and $\mathcal{W}$ is the distribution set of model parameters, i.e., synaptic weights and biases. Given a set of training data sets $\boldsymbol{X} = \{X_1, X_2, \ldots, X_N\}$ and the associated labels $\boldsymbol{y} = \{y_1, y_2, \ldots, y_N\}$, the BNN training aims to find the posterior distribution over model parameters $p(\mathcal{W}|\boldsymbol{X}, \boldsymbol{y})$, which in practice is very difficult to find since the posterior weight distribution is highly complex. Modern BNN research is mainly focused on variational inference methods [24], [25] or Markov chain Monte Carlo approaches [26], [27]. In this article, we mainly focus on the prediction/inference procedure rather than training procedure, and the Edward framework [16] is utilized for software evaluations, which relies on a variational inference method.

BNN prediction procedure starts from instantiating a series of concrete neural networks for forward propagation. The instantiation requires Gaussian random variable sampling for all the posterior distributions that have been identified during the training process. These sampling operations could be implemented both in software and hardware approaches [28]. Assuming that the posterior distribution of one weight ω fits a Gaussian distribution with location (or mean value) of μ and scale (or standard deviation) of σ, i.e., $\omega \sim N(\mu, \sigma^2)$, sampling the weight w requires to select a random number h from the standard Gaussian distribution $N(0, 1)$. Based on the theory that if $U \sim N(0, 1)$, then $X = U\sigma + \mu \sim N(\mu, \sigma^2)$, and the resulted random weight is calculated as $w = \sigma h + \mu$ by the scale–location transformation. Once all the random weights have been sampled, the BNN prediction follows the evaluation paradigm detailed in Section III. We notice that

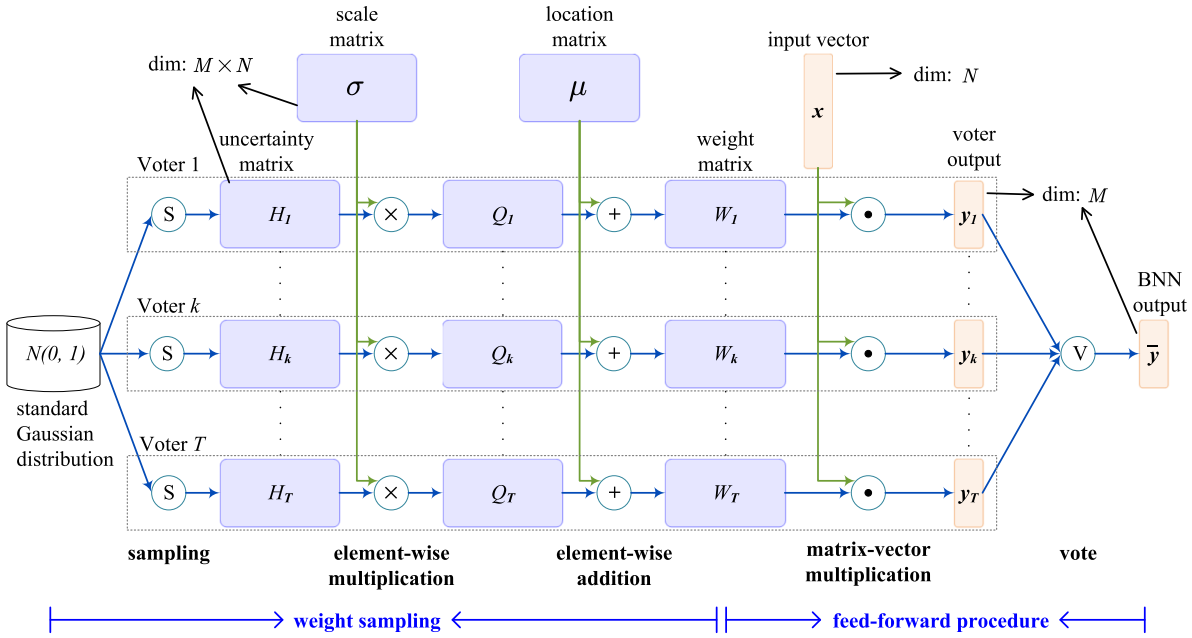

Fig. 2. Single-layer BNN dataflow divided in two steps: 1) Gaussian random number generation and 2) feedforward propagation. For the sake of convenience, the sampling procedure is denoted as Ⓢ and voters evaluation is denoted as Ⓥ. σ and μ are the well-trained BNN weights, and $\boldsymbol{x}$ is the input vector. The bias terms are not considered for simplify.

standard Gaussian random number generation algorithms are classified as inversion, transformation, rejection, and recursion methods [29], among which central limit theorem-based transformation method is most widely used.

As aforementioned, there have been many research works focused on DNN hardware acceleration, while very little has been reported in relation to BNN counterparts with the notable exception of VIBNN [23]. The BNN prediction procedure is accelerated on the FPGA platform in VIBNN by introducing two novel Gaussian random number generators, memory optimization techniques, and a deep pipeline structure. In [23], concrete neural networks are first instantiated based on weights distribution; then, VIBNN performs DNN evaluations repeatedly on them. However, these concrete neural networks are not independent so that VIBNN could not exactly capture the nature of the BNN paradigm. In this article, a novel BNN inference framework is proposed to reduce the computation complexity and BNN-oriented architecture is implemented to improve the hardware efficiency.

III. BNN Computation Reduction

In this section, a novel BNN inference method is demonstrated by taking the advantages of a fact that a certain amount of computations could be shared between its associated neural network instances. We first analyze the single-layer BNN dataflow and propose a feature DM approach to reduce the computation complexity. Subsequently, we extend DM to multilayer BNNs and analyze its computation complexity and memory consumption.

A. BNN Dataflow

Given a well-trained BNN, actual weights and biases sampling are required to instantiate a concrete neural network for inference procedure. However, several concrete neural networks with different parameter values are usually required for instantiation to fully explore the uncertainty using posterior distribution, rather one. Subsequently, the input data (e.g., an image) are fed into all the instantiated neural networks to obtain their predictions, and the actual response is determined by voting. The corresponding BNN dataflow is shown in Fig. 2 for a one-layer fully connected neural network. The involved variables of matrices and vectors are described in Table I, and the defined operators are shown in Table II.

The considered BNN contains N input neurons and M output neurons. μ and σ are $M \times N$ dimensional BNN weight values of posterior distribution parameters, regarded as location matrix and scale matrix, respectively. T is the number of NN samples to be evaluated in order to obtain the appropriate BNN's response. Based on the NN theory, an NN's output is calculated by (1), which means that when deriving BNN's output, (1) will be evaluated for T times

$$\boldsymbol{y} = W\boldsymbol{x} + \boldsymbol{b}. \tag{1}$$

Compared with the computation cost of matrix-vector multiplication between W and x, the computation cost of vector addition between Wx and b could be neglected. Hence, the bias terms are not considered in the following complexity analysis. Fig. 2 graphically shows the standard single-layer BNN dataflow, and Algorithm 1 details the computation procedure as follows.

1) T concrete weight matrices $W_1, W_2, \ldots, W_T$ are sampled according to the weight posterior distributions by exploiting Gaussian random number generators (GRNGs) (Lines 2–4).

TABLE I
BNN NOTATIONS

Symbol	Description	Dimension
N	input neural count	1
M	output neural count	1
T	the number of sampling	1
$\mathcal{L}$	the number of BNN layers	1
$\mathcal{W}$	weight distribution matrix	$M \times N$
W	weight matrix	$M \times N$
H	uncertainty matrix	$M \times N$
σ/μ	scale/location matrix	$M \times N$
$\boldsymbol{x}$	input vector	N
$\boldsymbol{y}$	output vector	M

TABLE II
BNN OPERATIONS

Symbol	Operation
$+$	element-wise addition
$\times$	element-wise multiplication
$\cdot$	matrix-vector multiplication
$<>_L$	line-wise inner product
Ⓢ	random number sample
Ⓥ	vote
Ⓟ	pre-compute
Ⓕ	feed-forward
Ⓣ	scale-location transformation

Algorithm 1 Standard BNN Evaluation

Input: Well-trained BNN with weight parameters μ and σ
Input: Input data $\boldsymbol{x}$
Output: BNN output $\bar{\boldsymbol{y}}$
1: **for** $(k = 1; k \leq T; k = k + 1)$ **do**
2: Sample uncertainty matrix H_k from $N(0, 1)$
3: $Q_k = H_k \times \sigma$
4: $W_k = Q_k + \mu$
5: $\boldsymbol{y_k} = W_k \cdot \boldsymbol{x}$
6: **end for**
7: $\bar{\boldsymbol{y}} = \frac{\sum_{k=1}^{T} \boldsymbol{y_k}}{T}$

2) Matrix-vector multiplication operation is performed between the input $\boldsymbol{x}$ and each weight matrix to generate T outputs $\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_T}$ (Line 5).
3) Output result $\bar{\boldsymbol{y}}$ is computed by averaging voter results of $\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_T}$ (Line 7).

As clearly suggested in Fig. 2, there are T sampling dataflow that could be performed in parallel, and each of them could be regarded as a voter who contributes to the output results' calculation in the final voting stage.

B. Feature Decomposition and Memorization

As previously discussed, the instantiated NNs are evaluated in parallel, and then, the following voting procedure determines the final output. Obviously, improving the number of instantiations could improve the BNN model robustness but will introduce more computation cost. In this section, the BNN inference dataflow is reformulated through the proposed DM strategy to reduce the total computation complexity. It is based on the observations that a certain amount of computations could be actually shared among different instantiated NNs.

1) DM for Single-Layer BNNs: To evaluate BNN's response for a given input, each voting result $\boldsymbol{y_k}$ is calculated by performing multiplication and addition on the input $\boldsymbol{x}$ and sampled weights W_k by (1). Each element of $\boldsymbol{y_k}$ is calculated as follows:

$$y_k^i = \sum_{j=1}^{N} w_k^{ij} x^j = \sum_{j=1}^{N} \left(h_k^{ij} \sigma^{ij} + \mu^{ij} \right) x^j \tag{2a}$$

$$= \sum_{j=1}^{N} h_k^{ij} \underbrace{\sigma^{ij} x^j} + \sum_{j=1}^{N} \underbrace{\mu^{ij} x^j} \tag{2b}$$

where $k = 1, 2, \ldots, T$ and $i = 1, 2, \ldots, M$, w_k^{ij} is sampled according to σ_k^{ij}, and μ_k^{ij}, h_k^{ij} is introduced to represent the uncertainty.

Following (2a), the single-layer BNN inference dataflow is shown in Fig. 2. Note that the input $\boldsymbol{x}$ and posterior distribution parameters σ and μ are the same for different voting evaluations, and we could consider to reuse these computations among them. The features expressed in (2a) could be equivalently decomposed as (2b) in which the computation results of $\sigma \times \boldsymbol{x}$ and $\mu \cdot \boldsymbol{x}$ are the same for all the T voters. Such potential computation sharing is the underlying property utilized in the proposed DM strategy, which memorizes $\sigma^{ij} x^j, (i = 1, 2, \ldots, M; j = 1, 2, \ldots, N)$ and $\sum_{j=1}^{N} \mu^{ij} x^j, (i = 1, 2, \ldots, M)$ so that they could be directly loaded from memory instead of being $T\times$ recomputed during the voters evaluation stage.

Following (2b), the BNN inference dataflow is shown in Fig. 3 with the proposed DM strategy. Compared with the standard dataflow as shown in Fig. 2, the multiplications of $\boldsymbol{x}$ and σ (as well as $\boldsymbol{x}$ and μ) are precomputed and stored in local memory. Hence, the BNN evaluation could be performed directly with the uncertainty matrices H_k (sampled from standard Gaussian distribution) without scale–location transformation. For each BNN evaluation, the voter's response $\boldsymbol{y_k}$ could be obtained according to the precomputed features and sampled uncertainty matrix H_k. The BNN inference dataflow with DM strategy is described in Algorithm 2. In Line 2, "$\times$" means that vector $\boldsymbol{x}$ performs elementwise product with every row of σ, which implies that β has the same dimension as σ. Obviously, the resulted η and β requires additional memory space to store them. In line 5, the operation "$<>_L$" indicates linewise inner product operation, i.e., an inner product is performed on each row of W and β. In Algorithm 2, Lines 1 and 2 and Lines 3–7 correspond to

Algorithm 2 DM-Based BNN Evaluation

Input: Well-trained BNN with weight parameters μ and σ
Input: Input data $\boldsymbol{x}$
Output: BNN output $\bar{\boldsymbol{y}}$
1: $\eta = \mu \cdot \boldsymbol{x}$
2: $\beta = \sigma \times \boldsymbol{x}$
3: **for** $(k = 1; k \leq T; k = k + 1)$ **do**
4: Sample uncertainty matrix H_k from $N(0, 1)$
5: $\boldsymbol{z_k} =< H_k, \beta >_L$
6: $\boldsymbol{y_k} = \boldsymbol{z_k} + \eta$
7: **end for**
8: $\bar{\boldsymbol{y}} = \frac{\sum_{k=1}^{T} \boldsymbol{y_k}}{T}$

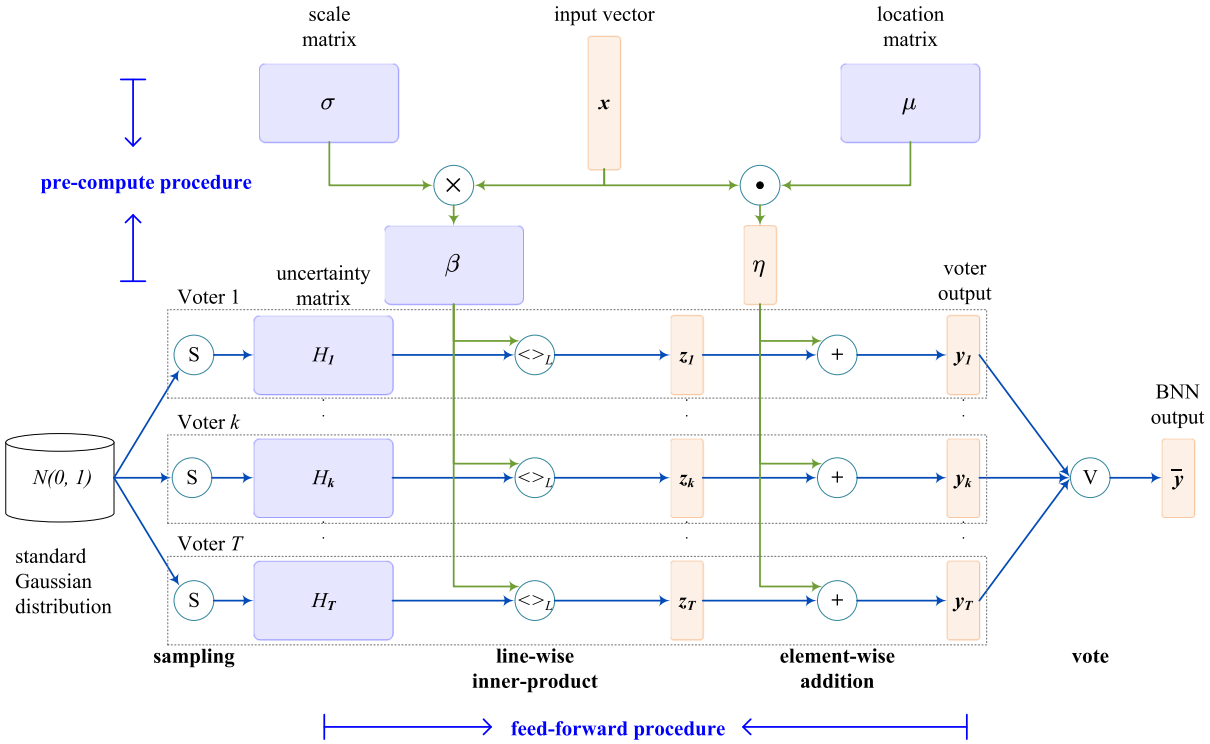

Fig. 3. Single-layer BNN dataflow with feature DM. The bias terms are not considered. β and η is precomputed and stored in local memory.

the precompute and feedforward stage in Fig. 3, respectively. For the sake of convenience, the precompute stage is denoted as Ⓟ and the feedforward evaluation is denoted as Ⓕ.

2) DM for Multilayer BNNs: It is well known that DNNs exhibit more intrinsic computation capabilities than single-layer counterparts when dealing with complex input–output mapping. It is of obvious interest to extend the DM strategy for multilayer Bayesian networks. As shown in Fig. 3, a single-layer network has a 1-to-T relationship between the input vector and the output vector, which means that all voters receive the same input $\boldsymbol{x}$. It is the key condition behind the DM strategy utilization. As for multilayer BNN, we can get T output vectors after the computation of the first layer which are also the input vectors of the second layer, that is to say, there is a T-to-T relationship between the input vector and the output vector rather than 1-to-T. As a result, the proposed DM strategy cannot be directly applied in all layers of multilayer BNNs. In this section, two methods are introduced for multilayer BNNs: Hybrid-BNN and DM-BNN, to take the advantages of DM strategy. A four-voter two-layer BNN (i.e. with one hidden layer) is taken as an example, which has been well trained with the distribution parameters (σ_1, μ_1) and (σ_2, μ_2) for each layer.

The dataflows of these two methods are shown in Fig. 4. For the Hybrid-BNN method, the DM strategy is only applied in the first layer and the corresponding dataflows are shown in Fig. 4(a), which could be derived from Figs. 2 and 3. In the first layer, β_{11} and η_{11} are precomputed using $\boldsymbol{x}$, σ_1, and μ_1, and memorized. Four sampled uncertainty matrices (H_{11}–H_{14}) are processed with β_{11} and η_{11} using the DM strategy yielding four outputs ($\boldsymbol{y}_{11}$–$\boldsymbol{y}_{14}$). In the second layer, four weight matrices (W_{21}–W_{24}) are obtained based on scale–location transformation and operated with the previous four outputs ($\boldsymbol{y}_{11}$–$\boldsymbol{y}_{14}$), respectively. Then, the results ($\boldsymbol{y}_{21}$–$\boldsymbol{y}_{24}$) are obtained for voting to determine the final BNN response $\bar{\boldsymbol{y}}$.

For the DM-BNN method, the DM strategy is applied in all layers, as graphically shown in Fig. 4(b). The one-input to T-outputs relationship now exists not only in the first layer but also in the following layer(s). In the first layer, two uncertainty matrices (H_{11} and H_{12}) are sampled and two corresponding outputs ($\boldsymbol{y}_{11}$ and $\boldsymbol{y}_{12}$) are calculated using the DM strategy. In the second layer, $\boldsymbol{y}_{11}$ is treated as input and two output ($\boldsymbol{y}_{21}$ and $\boldsymbol{y}_{22}$) are generated based on the DM strategy. Similarly, two outputs ($\boldsymbol{y}_{23}$ and $\boldsymbol{y}_{24}$) corresponding to $\boldsymbol{y}_{12}$ are generated.

C. Discussion

The performance of single-layer BNN and multilayer BNN with the DM strategy, as well as the memory overhead introduced by the DM strategy, is analyzed in this section.

1) Single-Layer BNN: To get inside on the implications of our proposal, we further evaluate the required computation complexity in terms of a number of operations, i.e., additions (ADD) and multiplications (MUL), and memory requirements.

Table III summarizes the number of arithmetic operation required by the dataflow in Figs. 2 and 3, i.e., Algorithms 1 and 2, respectively. If we concentrate on the number of multiplications, as they are more time consuming, the two approaches require $2MNT$ and $MN(T+2)$ multiplications, respectively. This indicates that if $T > 2$, which is obviously the case if one would like to deal with uncertainties using a Bayesian approach, the DM approach outperforms standard one. As T increases, the advantage is more substantial, reaching a theoretical maximum of 50%

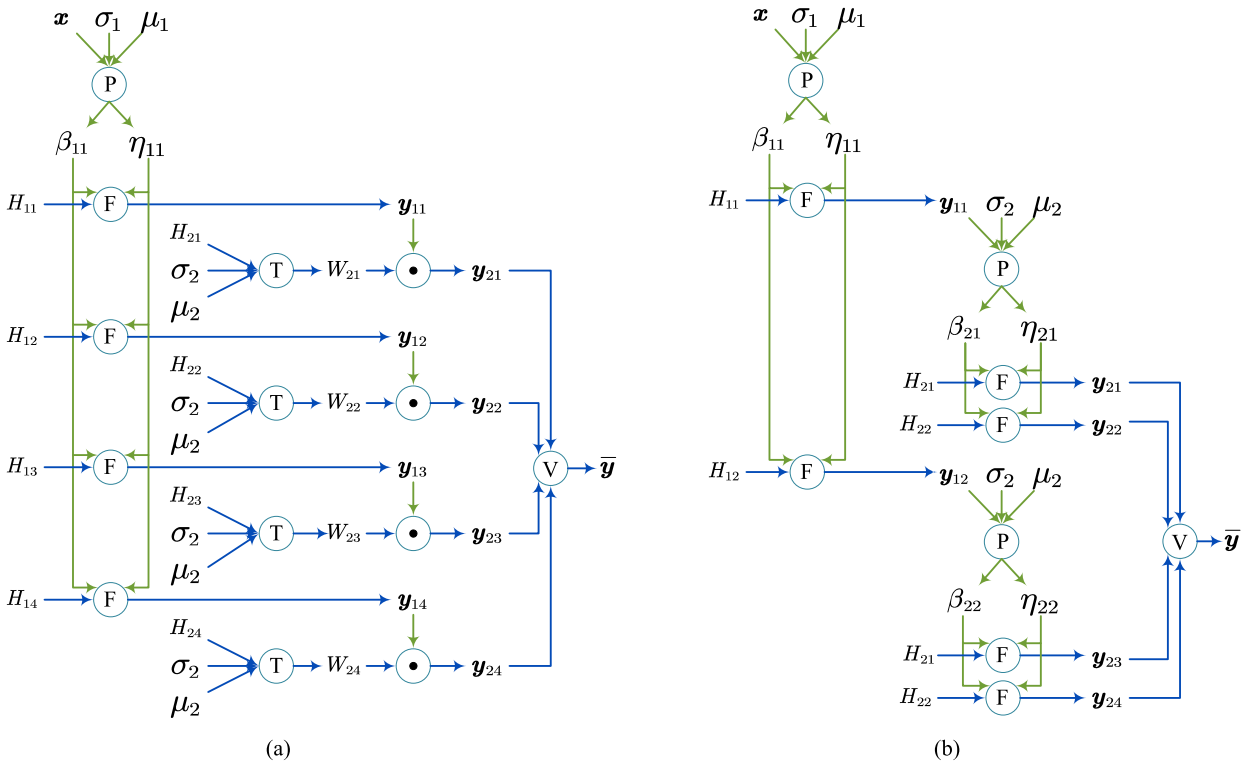

Fig. 4. Simplified dataflow of multilayer BNNs (with one hidden layer), where Ⓟ, Ⓕ, and Ⓣ stand for precompute, feedforward, and scale–location transformation, respectively. (a) Hybrid-BNN: DM strategy is only applied for the first layer. (b) DM-BNN: DM strategy is applied for all layers.

TABLE III
SINGLE-LAYER BNN COMPUTATION COST

Computation cost without DM strategy (Algorithm 1)		
Operation	# MUL	# ADD
$Q_k = H_k \times \sigma$	MNT	0
$W_k = Q_k + \mu$	0	MNT
$\boldsymbol{y_k} = W_k \cdot \boldsymbol{x}$	MNT	$M(N-1)T$
Total	$2MNT$	$\approx 2MNT$
Computation cost with DM strategy (Algorithm 2)		
Operation	# MUL	# ADD
$\eta = \mu \cdot \boldsymbol{x}$	MN	0
$\beta = \sigma \times \boldsymbol{x}$	MN	$M(N-1)$
$\boldsymbol{z_k} =< H_k, \beta >_L$	MNT	$M(N-1)T$
$\boldsymbol{y_k} = \boldsymbol{z_k} + \eta$	0	MT
Total	$MN(T+2)$	$\approx MN(T+1)$

reduction as suggested by the following equation:

$$\lim_{T\to\infty} \frac{MN(T+2)}{2MNT} = \frac{1}{2}. \tag{3}$$

Given that one addition takes one cycle and one multiplication by 2 cycles in state-of-the-art processors, the computation cost in terms of ADD only could be regarded as $\approx 3MNT$ and $\approx 6MNT$ for single-layer BNN evaluation with and without DM strategy, respectively, which results in an overall speedup of ≈ 2.

2) Multilayer BNN: Related to the multilayer BNN dataflows in Fig. 4, the performance of Hybrid-BNN and DM-BNN will be discussed.

In the Hybrid-BNN dataflow, the DM strategy is only applied in the first layer. Thus, it could only reduce the computation cost of the first layer. Usually, the first layer accounts for more than 80% of the total computation, and consequently, the computation cost of the entire flow in Hybrid-BNN could be still reduced by 40%. In the DM-BNN dataflow, the computation cost of all layers could be reduced by about 50%. It can also be found that the required number of uncertainty matrices for each layer in DM-BNN is less than that in Hybrid-BNN. As shown in Fig. 4, eight uncertainty matrices (four for each layer) are sampled in Hybrid-BNN in order to get four-voter results, whereas four uncertainty matrices (two for each layer) are sampled in DM-BNN. In general, if BNN has $\mathcal{L}$ layers, $\sqrt[\mathcal{L}]{T}$ uncertainty matrices are sufficient for each layer to obtain T voter results. Less uncertainty matrices mean less computation cost. Finally, DM-BNN could reduce more than 50% computation cost when compared with standard BNN. Even though some voting outputs inherit the same uncertainty, the experimental results indicate that its influence could be ignored.

3) DM Deployment in Convolutional Layers: Convolutional neural networks (CNNs) are a class of DNNs able to capture spatial and temporal dependences in an image through the application of relevant filters. The advantages of CNNs make them most successful in perspective tasks. Thus, it is important to extend the proposed DM strategy in convolutional layers. Fortunately, this extension could be achieved using convolutional layer unfolding [30], which is a well-known technique that has been utilized in many applications (see [31]–[33]). This technique relies on the creation of convolution matrices such that the convolution computation is transformed into a matrix multiplication. Thus, after applying unfolding on the convolution layers, the DM strategy can be directly applied to them.

4) Memory Overhead: Based on the previous analysis, the proposed DM strategy could effectively reduce the computation

cost at the expense of additional local memory. In the standard BNN dataflow, we only need to store the weight distribution parameters σ and μ, but for DM dataflow, additional storage is required for matrix β who has the same dimension as σ and μ, which means that about 50% memory overhead is introduced.

IV. Memory Reduction for DM Strategy

In this section, the memory overhead issue introduced by the DM strategy is considered and a memory-friendly computing mechanism is proposed. The goal is to minimize the additional allocated memory as much as possible without inducing additional computation overhead.

Generally speaking, due to hardware limitations, one cannot simultaneously evaluate all the T-sampling feedforward neural networks. For some edge devices, a neural network is even divided into several parts and operates in a time-multiplexed manner [34], [35]. When assuming that only αT-sampling feedforward neural networks can be evaluated simultaneously where $0 < \alpha \leq 1$, only αTMN Gaussian random numbers are sampled in each iteration. Thus, αT entire uncertainty matrices ($H \in \mathbb{R}^{M\times N}$) and voting outputs ($\boldsymbol{y} \in \mathbb{R}^{M}$) are generated in each iteration, while α^{-1} iterations are required totally. In order to match the dimension of uncertainty matrix, additional memory with a size of $M \times N$ should be allocated to store β by this way.

If we want to reduce the memory overhead of DM strategy, the dimension of uncertainty matrix involved in each iteration should be shrunken first. Guided by this, a memory-friendly computing mechanism can be derived. In this computing mechanism, αTMN Gaussian random numbers generated in each iteration are redistributed to T submatrices ($H' \in \mathbb{R}^{\alpha M\times N}$). Consequently, β could be partially computed and memorized with the same size as H'. At the end of each iteration, T suboutputs ($\boldsymbol{y}' \in \mathbb{R}^{\alpha M}$) are calculated, and after α^{-1} iterations, all T voter outputs are obtained. With this approach, the introduced memory overhead by the DM strategy could be reduced from 50% to $\alpha\times 50\%$. For the sake of concise and simplicity, the mechanism is shown for $\alpha = 1/2$ in Fig. 5.

It is worth noting that the benefit of the proposed memory friendly approach is determined by computation resources. If only limited computation resources are available, it could reduce the memory overhead as much as possible while maintaining the computation performance. If the computation resources are adequate that all T voters could be evaluated in parallel, the memory-friendly approach could provide a trade-off between computation performance and memory overhead.

V. Experimental Results

The advantages of BNN have been demonstrated by many previous works. The work of [36]–[38] show BNNs ability when dealing with uncertainty. The advantages of BNNs in autonomous vehicle safety are presented in [39] and [40]. The work [23] suggests that BNN performs much better than DNN as training data size shrinks. Rawat *et al.* [41] concluded that BNN can be considered for detecting adversarial examples. In this section, we first demonstrate the strengths of BNN in classification tasks using small data sets. Then, the efficiency of the proposed DM strategy is evaluated.

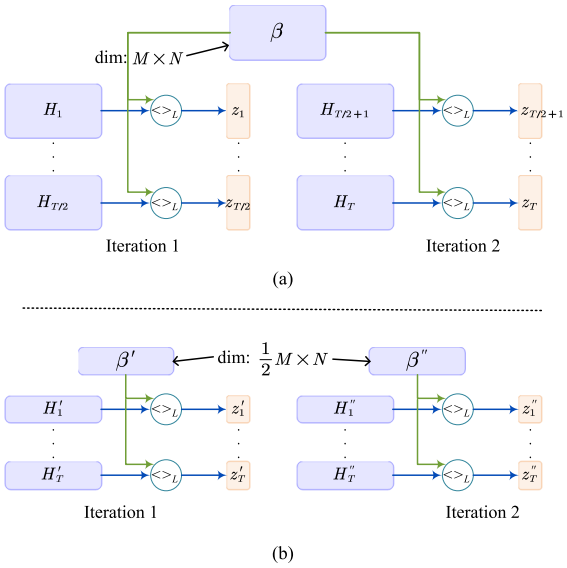

Fig. 5. BNN evaluation with $\alpha = 1/2$ for memory reduction. (a) Standard DM approach. (b) Memory friendly DM approach. The additional memory allocated by (b) is reduced from $M \times N$ to $\alpha M \times N$ (i.e., by $1/2M \times N$). In (b), variable with $'$ indicates the first half part of it and with $''$ indicates the second half of it. For example, $\beta = [\beta'\beta']^T$.

A. BNN Performance on Small-Scale Data Sets

In order to evaluate the BNN performance on small data sets, two famous data sets MNIST [42] and FMNIST [43] are utilized. Both MNIST and FMNIST data sets have 60 000 images for training and 10 000 images for testing. Each image is a gray bitmap whose size is 28 × 28. The MNIST data set contains ten classes of handwritten digits (from "0" to "9"), whereas the FMNIST data set contains ten classes of clothes or shoes. In this experiment, two data sets are reduced to small-scale data sets based on a shrink ratio. In the reduced data sets, the images are randomly selected from the original data set and the number of images for each class is the same. For example with the shrink ratio of 256, each class has about 24 (i.e., $\lceil 60\,000/256/10\rceil$) images to be chosen as a subset. Due to the fact that there are ten classes in MNIST/FMNIST, each subset contains 240 images for training. It is noted that there are still 10 000 images for testing no matter how many training images there are.

For the MNIST dataset, a three-layer (with two hidden layers) fully connected neural network is built with a 784-200-200-10 configuration. For the FMNIST data set that is much more complicated, the LeNet-5 [44] structure is utilized. The non-BNN is trained using TensorFlow [45], whereas the BNN is trained using the Edward framework [16]. The training parameters, such as epochs, batch size, and learning rate, are set to be the same for fairness. An overview for the NN and BNN achieved accuracy on the small-scale data sets is presented in Fig. 6. The horizontal axis represents the shrink ratio of the original data set, and the vertical axis represents the testing accuracy on the 10 000 testing data sets. Fig. 6 clearly indicates that the BNN could achieve better performance than the non-BNN when the training data size shrinks.

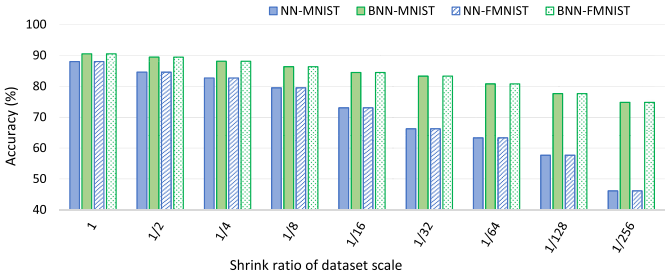

Fig. 6. Accuracy comparison between NN and BNN on MNIST/FMNIST data sets for different training data scales. The accuracy is calculated using 10 000 testing data sets.

TABLE IV
SOFTWARE IMPLEMENTATION RESULTS

Method	Accuracy	# MUL ($\times 10^6$)	# ADD ($\times 10^6$)
Standard BNN	96.73%	39.8	39.7
Hybrid-BNN	96.73%	24.2	24.1
DM-BNN	96.7%	6.9	6.7

B. DM Strategy Evaluation

In this article, two methods are described for multilayer BNNs in Section III, which are theoretically proved to reduce the computation complexity. In this section, both software implementation and hardware implementation are presented in order to evaluate the performance of the proposed DM strategy. The three-layer (with two hidden layers) fully connected neural networks with the MNIST data set are used. The number of sampling T in standard BNN and Hybrid-BNN for each layer is set as 100, and for the three layers in DM-BNN, T is set as 10, 10, and 5, respectively. Consequently, 500 voting results are generated for DM-BNN.

In this experiment, both Hybrid-BNN (the DM strategy is applied only in the first layer) and DM-BNN (the DM strategy is applied in all layers) are implemented for evaluation. Recently, Cai *et al.* [23] proposed an approach to accelerate BNN inference. In their work, two GRNGs and some architecture-level optimizations are proposed. Its dataflow is also reimplemented in our experiments, which is referred to as standard BNN (the DM strategy is not applied at all). In order to make a fair comparison, the energy consumption of GRNGs is not calculated and no architecture-level optimizations are implemented. In this way, the comparison results could well demonstrate the efficiency of the proposed DM strategy.

1) Software Implementation: All methods are implemented using Python language and evaluated on a 64-bit Linux server. Table IV summarizes the accuracy and required number of operations. The second column reports the prediction accuracy for the 10 000 MNIST testing images, whereas the computation complexity in terms of multiplications (# MUL) and additions (# ADD) is given in columns 3 and 4, respectively.

In Hybrid-BNN, the DM strategy is only applied in the first layer, which covers about 79% of total network computation, and Table IV confirms the expected Hybrid-BNN theoretical computation reduction of about 39%. DM-BNN could effectively reduce the computation cost because the feature DM strategy is utilized with all layers. Moreover, because only ten uncertainty matrices are sampled to participate in the first BNN layer, the total computation dropped by 82.5%, which is beyond the 50% upper bound achievable by DM in single-layer BNNs. From Table IV, we can observe that the accuracy of Hybrid-BNN and standard BNN is the same as expected, whereas DM-BNN could significantly reduce the computations only at the cost of very slight accuracy degradation.

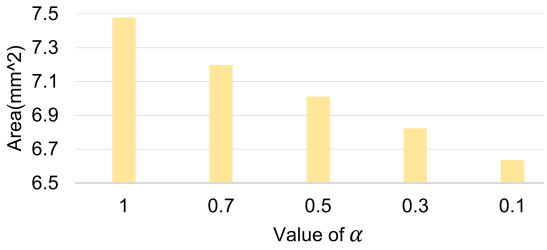

Fig. 7. Evaluation of memory reduction strategy.

2) Hardware Implementation: Hardware implementations of the three methods are also performed for evaluating the BNN inference latency, power consumption, and area efficiency. They are implemented as Verilog and synthesized by Synopsys Design Compiler with a 45-nm FreePDK technology; 8-bit fixed-point number representation is utilized in the designs. Cacti [46] is exploited to estimate the area and energy consumption of the involved memory.

We first evaluate the efficiency of the memory-friendly computing mechanism, which is proposed to reduce the memory overhead introduced by the DM strategy. This mechanism is designed based on the fact that hardware resources are always limited. Therefore, in this experiment, the computing mechanism efficiency is evaluated using the required system area. Assuming that only αT-sampling feedforward neural networks can be evaluated simultaneously, Fig. 7 shows the system area corresponding to different α values. The horizontal axis means the value of α, and the vertical axis means the required hardware system area (mm^2). The experimental results demonstrate that when α decreases, the required hardware system area could also be reduced.

The efficiency of the proposed DM strategy in the BNN inference stage is evaluated with $\alpha = 0.1$. Table V describes the comparison of the three hardware implementations in terms of accuracy, area (mm^2), energy consumption (nJ), and total execution time ($\times 10^3$ s) for BNN inference. One can observe that the accuracy has been slightly diminished when compared with the software implementation, but this is inherent due to the utilization of lower precision, 8-bit fixed-point number instead of 32-bit floating-point number. Compared with the standard BNN, Hybrid-BNN and DM-BNN have about 27% and 14% area overhead, respectively. There are two reasons why standard BNN has the best area efficiency, while Hybrid-BNN has the worst area efficiency. First, both Hybrid-BNN and DM-BNN require extra local memory that is inherent to feature DM strategy utilization. Second, while hardware resources could be shared among different layers in the standard BNN and DM-BNN, because the computing mechanisms of all layers are the same, this is not the case for Hybrid-BNN, in which the first layer requires a different mechanism than the other layers. As reported in column 4, Hybrid-BNN and DM-BNN provide 29% and 73% energy

TABLE V
HARDWARE IMPLEMENTATION RESULTS

Method	Accuracy	Area (mm^2)	Energy (μJ)	Runtime (μs)
Standard BNN	95.42%	5.76	172	392
Hybrid-BNN	95.42%	7.33	122	259
DM-BNN	95.35%	6.63	46	97

consumption reductions, respectively, when compared with standard BNN. Concerning the total execution time required for the evaluation of one MNIST test data, a speedup of 1.5× and 4× is obtained for Hybrid-BNN and DM-BNN over the standard BNN, respectively.

To summarize, Hybrid-BNN reduces the energy consumption by 29% and achieves a 1.5× speedup at the expense of 27% overhead in area, while DM-BNN reduces the energy consumption by 73% and achieves a 4× speedup at the expense 14% overhead with a slight accuracy decreasing.

VI. CONCLUSION

This article addresses the high computation complexity of the BNN inference procedure and introduces a novel computation efficient BNN inference approach, which potentially enables BNNs' utilization in resources-constrained systems, e.g., IoT. We conduct a deep analysis of BNN inference dataflow and introduce a DM strategy that reduces the computation complexity, while having negligible BNN inference accuracy reduction, at the expense of some memory overhead. We further propose the DM strategy utilization in multilayer BNNs and introduce a memory-friendly computing framework that is able to mitigate the DM-induced memory overhead. Finally, our approach is implemented by Verilog and synthesized with a 45-nm FreePDK technology. Evaluation results demonstrate that the proposed strategy provides an energy consumption reduction of 73% and a 4× speedup at the expense of 14% area overhead compared with the standard BNN inference. As a final remark, we note that the reported performance improvement can be further improved using architecture-level optimization (e.g., memory optimization in [23]) or network compression (e.g., pruning in [19]), which constitute future work subjects.

REFERENCES

[1] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA, USA: MIT Press, 1998, pp. 255–258.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS*, 2012, pp. 1097–1105.

[5] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2014, pp. 3104–3112.

[6] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[7] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning affordance for direct perception in autonomous driving," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 2722–2730.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[9] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[10] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 427–436.

[11] Y. Gal and Z. Ghahramani, "Bayesian convolutional neural networks with Bernoulli approximate variational inference," 2015, *arXiv:1506.02158*. [Online]. Available: http://arxiv.org/abs/1506.02158

[12] J. L. Ticknor, "A Bayesian regularized artificial neural network for stock market forecasting," *Expert Syst. Appl.*, vol. 40, no. 14, pp. 5501–5506, Oct. 2013.

[13] X. Jia, J. Yang, P. Dai, R. Liu, Y. Chen, and W. Zhao, "SPINBIS: Spintronics-based Bayesian inference system with stochastic computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 4, pp. 789–802, Apr. 2020.

[14] X. Jia, J. Yang, Z. Wang, Y. Chen, H. H. Li, and W. Zhao, "Spintronics based stochastic computing for efficient Bayesian inference system," in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2018, pp. 580–585.

[15] J.-T. Chien and Y.-C. Ku, "Bayesian recurrent neural network for language modeling," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 2, pp. 361–374, Feb. 2016.

[16] D. Tran, M. D. Hoffman, R. A. Saurous, E. Brevdo, K. Murphy, and D. M. Blei, "Deep probabilistic programming," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–18.

[17] E. Bingham *et al.*, "Pyro: Deep universal probabilistic programming," 2018, *arXiv:1810.09538*. [Online]. Available: http://arxiv.org/abs/1810.09538

[18] J. Shi *et al.*, "ZhuSuan: A library for Bayesian deep learning," 2017, *arXiv:1709.05870*. [Online]. Available: http://arxiv.org/abs/1709.05870

[19] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 1135–1143.

[20] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2014, pp. 1269–1277.

[21] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2016, pp. 2074–2082.

[22] P. Knag, J. K. Kim, T. Chen, and Z. Zhang, "A sparse coding neural network ASIC with on-chip learning for feature extraction and encoding," *IEEE J. Solid-State Circuits*, vol. 50, no. 4, pp. 1070–1079, Apr. 2015.

[23] R. Cai *et al.*, "VIBNN: Hardware acceleration of Bayesian neural networks," in *Proc. ACM Architectural Support Program. Lang. Operating Syst. (ASPLOS)*. New York, NY, USA: ACM, 2018, pp. 476–488.

[24] A. Graves, "Practical variational inference for neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2011, pp. 2348–2356.

[25] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," 2015, *arXiv:1505.05424*. [Online]. Available: http://arxiv.org/abs/1505.05424

[26] T. Chen, E. Fox, and C. Guestrin, "Stochastic gradient Hamiltonian Monte Carlo," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2014, pp. 1683–1691.

[27] A. K. Balan, V. Rathod, K. P. Murphy, and M. Welling, "Bayesian dark knowledge," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 3438–3446.

[28] J. S. Malik and A. Hemani, "Gaussian random number generation: A survey on hardware architectures," *ACM Comput. Surv.*, vol. 49, no. 3, p. 53, Dec. 2016.

[29] D. B. Thomas, W. Luk, P. H. W. Leong, and J. D. Villasenor, "Gaussian random number generators," *ACM Comput. Surv.*, vol. 39, no. 4, p. 11, 2007.

[30] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in *Proc. 10th Int. Workshop Frontiers Handwriting Recognit.*, 2006, pp. 1–7.

[31] S. Chetlur *et al.*, "CuDNN: Efficient primitives for deep learning," 2014, *arXiv:1410.0759*. [Online]. Available: http://arxiv.org/abs/1410.0759

[32] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2010, pp. 253–256.

[33] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, 2011, pp. 1237–1242.
[34] A. Ren, Z. Li, Y. Wang, Q. Qiu, and B. Yuan, "Designing reconfigurable large-scale deep learning systems using stochastic computing," in *Proc. IEEE Int. Conf. Rebooting Comput. (ICRC)*, Oct. 2016, pp. 1–7.
[35] G. Indiveri, F. Corradi, and N. Qiao, "Neuromorphic architectures for spiking deep neural networks," in *IEDM Tech. Dig.*, Dec. 2015, pp. 2–4.
[36] A. Kendall and Y. Gal, "What uncertainties do we need in Bayesian deep learning for computer vision?" in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 5574–5584.
[37] Y. Zhu and N. Zabaras, "Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification," *J. Comput. Phys.*, vol. 366, pp. 415–447, Aug. 2018.
[38] J. van der Westhuizen and J. Lasenby, "Bayesian LSTMs in medicine," 2017, *arXiv:1706.01242*. [Online]. Available: http://arxiv.org/abs/1706.01242
[39] R. McAllister *et al.*, "Concrete problems for autonomous vehicle safety: Advantages of Bayesian deep learning," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, Aug. 2017, pp. 4745–4753.
[40] D. Feng, L. Rosenbaum, and K. Dietmayer, "Towards safe autonomous driving: Capture uncertainty in the deep neural network for lidar 3D vehicle detection," in *Proc. 21st Int. Conf. Intell. Transp. Syst. (ITSC)*, Nov. 2018, pp. 3266–3273.
[41] A. Rawat, M. Wistuba, and M.-I. Nicolae, "Adversarial phenomenon in the eyes of Bayesian deep learning," 2017, *arXiv:1711.08244*. [Online]. Available: http://arxiv.org/abs/1711.08244
[42] C. C. Y. LeCun and C. J. Burges. (2010). *MNIST Handwritten Digit Database*. [Online]. Available: http://yann.lecun.com/exdb/mnist
[43] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*. [Online]. Available: https://arxiv.org/abs/1708.07747
[44] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[45] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*. [Online]. Available: http://arxiv.org/abs/1603.04467
[46] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2007, pp. 3–14.

Xiaotao Jia (Member, IEEE) received the B.S. degree in mathematics from Beijing Jiao Tong University, Beijing, China, in 2011, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, in 2016.

From 2016 to 2019, he was a Post-Doctoral Researcher with the School of Microelectronics, Beihang University, Beijing, where he is currently an Assistant Professor. His current research interests include spintronic circuits, stochastic computing, and Bayesian deep learning.

Jianlei Yang (Member, IEEE) received the B.S. degree in microelectronics from Xidian University, Xi'an, China, in 2009, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2014.

From 2014 to 2016, he was a Post-Doctoral Researcher with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA, USA. He is currently an Associate Professor with the School of Computer Science and Engineering, Beihang University, Beijing. His current research interests include spintronics and neuromorphic computing systems.

Dr. Yang was a recipient of the First Place and Second Place on the ACM TAU Power Grid Simulation Contest in 2011 and 2012, respectively, the IEEE ICCD Best Paper Award in 2013, the IEEE ICESS Best Paper Award in 2017, and the ACM GLSVLSI Best Paper Nomination in 2015.

Runze Liu received the B.S. degree from the School of Computer Science and Engineering, Beihang University, Beijing, China, in 2018.

His research interests include computing architectures for deep learning and machine vision.

Xueyan Wang (Member, IEEE) received the B.S. degree in computer science from Shandong University, Jinan, China, in 2013, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2018.

From 2015 to 2016, she was a Visiting Scholar with the University of Maryland, College Park, MD, USA. She is currently a Post-Doctoral Researcher with the School of Microelectronics, Beihang University, Beijing. Her current research interests include highly efficient processing-in-memory (PIM) architectures and hardware security.

Sorin Dan Cotofana (Fellow, IEEE) received the M.Sc. degree in computer science from Politehnica University of Bucharest, Bucharest, Romania, in 1984, and the Ph.D. degree in electrical engineering from the Delft University of Technology, Delft, The Netherlands, in 1998.

He is currently with the Computer Engineering Laboratory, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, The Netherlands. He has coauthored more than 250 papers in peer-reviewed international journals and conferences. His current research interests include the design and implementation of dependable/reliable systems out of unpredictable/unreliable components, aging assessment/prediction and lifetime reliability-aware resource management, and unconventional computation paradigms and computation with emerging nanodevices.

Dr. Cotofana is a member of HiPEAC. He received 12 best paper awards at international conferences. He is also the Editor-in-Chief of the IEEE TRANSACTIONS ON NANOTECHNOLOGY, an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS, and a Distinguished Lecturer and a member of Board of Governors of the IEEE Circuits and Systems Society (CASS).

Weisheng Zhao (Fellow, IEEE) received the Ph.D. degree in physics from the University of Paris Sud, Paris, France, in 2007.

In 2009, he joined the French National Research Center (CNRS), Paris, as a tenured Research Scientist. Since 2014, he has been a Distinguished Professor with Beihang University, Beijing, China, where he is currently a Professor with the School of Microelectronics. He has published more than 200 scientific articles in leading journals and conferences, such as *Nature Electronics*, *Nature Communications*, *Advanced Materials*, IEEE TRANSACTIONS, the International Symposium on Computer Architecture (ISCA), and the Design Automation Conference (DAC). His current research interests include the hybrid integration of nanodevices with CMOS circuit and new nonvolatile memory (40-nm technology node and below), such as MRAM circuit and architecture design.

Dr. Zhao is also the Editor-In-Chief of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS.