Fully-Parallel Area-Efficient Deep Neural Network Design using Stochastic Computing

Yi Xie, Siyu Liao, Bo Yuan, Yanzhi Wang, and Zhongfeng Wang, *Fellow, IEEE*

***Abstract*—Deep neural network (DNN) has emerged as a powerful machine learning technique for various artificial intelligence applications. Due to the unique advantages on speed, area and power, specific hardware design has become a very attractive solution for the efficient deployment of DNN. However, the huge resource cost of multipliers makes the fully-parallel implementations of multiplication-intensive DNN still very prohibitive in many real-time resource-constrained embedded applications. This paper proposes a fully-parallel area-efficient stochastic DNN design. By leveraging stochastic computing (SC) technique, the computations of DNN are implemented using very simple stochastic logic, thereby enabling low-complexity fully-parallel DNN design. In addition, to avoid the accuracy loss incurred by the approximation of SC, we propose an accuracy-aware DNN datapath architecture to retain the test accuracy of stochastic DNN. Moreover, we propose novel low-complexity architecture for binary-to-stochastic (B-to-S) interface to drastically reduce the footprint of peripheral B-to-S circuit. Experimental results show that the proposed stochastic DNN design achieves much better hardware performance than non-stochastic design with negligible test accuracy loss.**

***Index Terms*—Deep Neural Network, Stochastic Computing, Fully-parallel, Area-efficient**

I. Introduction

DEEP neural network (DNN) [1], a.k.a. *deep learning*, has emerged as one of the most popular machine learning techniques for various artificial intelligence applications, such as object recognition, scene understanding, speech recognition, machine translation etc. Due to its unprecedentedly powerful knowledge-learning and feature-extracting capability, DNN is expected to become the critical component and technique for building the next-generation intelligent cyber-infrastructure.

The current success of DNNs is actually based on the remarkable progress in several related fields, including more efficient network training approaches, more available training/inference data, more advanced network topology and more powerful computer processors than before. Among those factors, the rapid development of the computing resource is very critical: Driven by the Moore's law, the computing power of modern CPU/GPU processors has increased 200,000X than 1990 [2]. Such extraordinary progress in the underlying computing platform makes the training/inference tasks on the large-scale DNNs, which could not be performed in the earlier years, become possible. Nowadays, very powerful CPU/GPU has become the standard hardware configuration for the deployment of DNNs.

Recently, specific hardware designs of DNNs, such as application-specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs) solutions, are receiving much attention in both academia and industry [3][4]. Because their architectures are customized for the kernel computation in DNNs, these DNN hardware accelerators are very superior to their CPU/GPU-based counterparts in terms of area, speed and power. From the perspective of deployment, such promising advantage on hardware performance is very attractive in many practical applications, especially for those latency-sensitive power-sensitive DNN inference tasks.

Although the state-of-the-art DNN accelerators already exceeded the CPU/GPU-based implementations on hardware performance, they still suffer from ultra-high hardware cost: Consider DNN is computation intensive, especially multiplication intensive, the huge resource cost of multipliers makes the hardware mapping of DNN, even only using a partially parallel processing strategy, very resource-consuming. As a result, all the state-of-the-art DNN accelerators use very limited number of multipliers to meet the budget of overall chip area. For instance, in [3] only 272 multipliers are implemented while the entire number of multiplication for one layer of DNN is usually 10,000 or beyond. Without any doubt, such small degree of parallelism impedes the full utilization of DNN's inherent highly parallel characteristic, thereby further limiting the processing throughput and computing performance. In short, the inevitable huge resource cost of multipliers makes the area-efficient fully-parallel implementations of DNNs prohibitive in many real-time area-limited power-constrained embedded machine learning applications, such as unmanned aerial vehicles, intelligent robotics, etc.

This paper proposes a fully-parallel area-efficient stochastic DNN design. By leveraging stochastic computing (SC) [5], the original resource-consuming multiplication is now performed using very simple stochastic logic. As a result, the entire DNN can be implemented on the hardware using fully-parallel mapping strategy with very low area cost and power consumption.

Manuscript received May 12, 2017. This work was supported in part by the National Science Foundation CNS-1704662, Algorithm in the Field (AitF) and Defense Advanced Research Projects Agency (DARPA) SAGA programs.

Yi Xie, Siyu Liao and Bo Yuan are with the City University of New York, USA. (e-mail: yxie000@citymail.cuny.edu, sliao2@gradcenter.cuny.edu, byuan@ccny.cuny.edu).

Y. Wang is with the Department of Electrical Engineering Computer Science, Syracuse University, USA. (e-mail: ywang393@syr.edu).

Z. Wang is with the School of Electronic Science and Engineering, Nanjing University, China. (e-mail: zfwang@nju.edu.cn).

Besides, joint optimization at both software and hardware level is performed to improve the accuracy and hardware performance of stochastic DNN simultaneously. More specifically, the contribution of this paper is twofold. First, we propose an accuracy-aware DNN datapath architecture that enables significant reduction in the approximation error incurred by the stochastic computing. Second, we propose a novel area-efficient architecture for binary-to-stochastic (B-to-S) interface, which drastically reduces the footprint of peripheral B-to-S circuit. Based on our proposed approaches, a design example for MNIST dataset is developed and implemented. Analysis results show that the proposed stochastic DNN design achieves much better hardware performance than non-stochastic design with negligible test accuracy loss.

The rest of this paper is organized as below. A brief review of DNN and stochastic computing is given in Section II. Section III presents the proposed fully-parallel area-efficient high-accuracy stochastic DNN. The accuracy and hardware performance of an example stochastic DNN are analyzed in Section IV. Section V draws the conclusions.

II. Background

A. Deep Neural Network (DNN)

In general, a DNN consists of multiple layers of neurons. Fig. 1(a) illustrates the structure of a four-layer (with two hidden layers) DNN. From this figure it is seen that, the connections between the neurons only exist for the intra-layer case. In other words, for each neuron in the certain layer it only receives the input data from the neurons in the previous layer, and then sends its processed output to the connected neuron in the next layer. More specifically, consider the *j*-th neuron in the *k*-th layer, its computation can be formulated as below:

$$x_j^{k+1} = \sigma(\mathbf{xw}) = \sigma(\textstyle\sum_{i=0}^{n} x_i^k\, w_{i,j}^k), \tag{1}$$

where x_j^{k+1} is the output from the *j*-th neuron in the current (the *k*-th) layer, x_i^k is the output from the *i*-th neuron in the previous (the (*k*-1)-th) layer, $w_{i,j}^k$ is the weight of the connection between these two neuron in the adjacent layers, and $\sigma(\cdot)$ is the activation function such as sigmoid $f(x) = 1/(1+e^{-x})$, hypertangent function $f(x) = \tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$ etc. Also, notice that the accumulation of $x_i^k w_{i,j}^k$ can be represented as the inner product between vectors **x** and **w**, where x_i^k and $w_{i,j}^k$ are the *i*-th element of **x** and **w**, respectively.

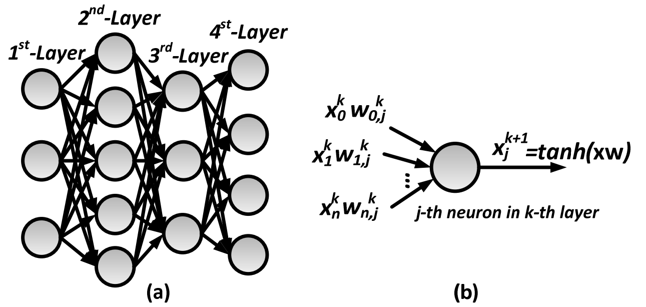

Fig. 1. (a) Example 4-layer DNN. (b) Computation in the neuron.

B. Stochastic Computing (SC)

Stochastic computing [5] is a low-cost probabilistic computing technique. Different from the conventional two's complement weighted computing (referred as *binary computing*), stochastic computing utilizes a stream of equal-weighted bits to represent the number. Here the value of the number is related to the ratio of "1" in the entire bit-stream. More specifically, in the *unipolar format* of SC [5], a length-*L* bit-stream with *Z* bits "1" carries the information of number *z*=*Z*/*L*. Consider the number represented by the unipolar SC is limited to positive value, *bipolar format* of SC [5], which can represent negative number, is more popular for practical applications. As shown in Fig. 2(a), in the bipolar SC scenario, a number *z* within the range [-1, 1] can be represented by any length-*L* bit-stream with *Z* bits "1" as long as *z*=2(*Z*/*L*)-1.Notice that for the numbers beyond the [-1, 1], they can still be represented by bipolar SC after scaling down to this range.

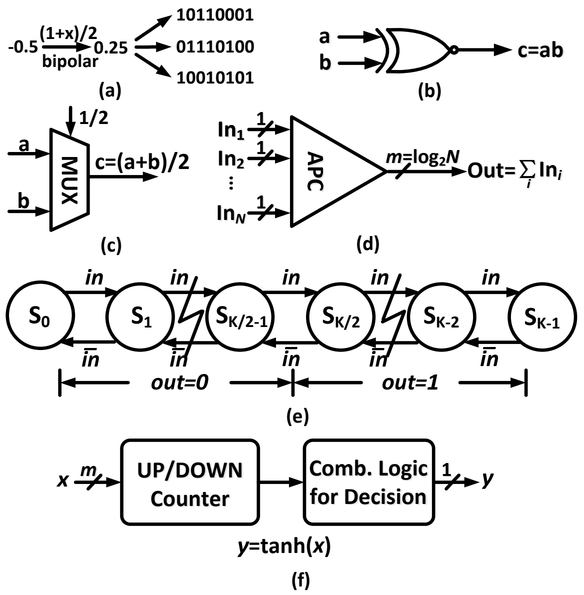

Fig. 2. (a) Bipolar SC representation. (b) SC multiplier. (c) SC scaled adder. (d) APC-based SC non-scaled adder. (e) SC hypertangent module with bit stream input. (f) SC hypertangent module with integer stream input.

One of the most attractive advantages of SC is its ultra-low hardware cost. With the use of probabilistic theory, many basic arithmetic operations can be implemented using very simple stochastic logic. For instance, as shown in Fig. 2(b), in the scenario of bipolar SC the multiplication can be performed using only one XNOR gate. Also, many other operations, such as addition and hypertangent, can also be mapped on simple stochastic logic [5] (see Fig. 2(c)-(f)). Notice that the traditional stochastic adder has to be the scaled-down version (see Fig. 2(c)). In the case of non-scaled results being needed, the accumulator parallel counter (APC) [6] can be used to calculate the sum of multiple stochastic inputs. Since the output of such APC is the integer-stream instead of bit-stream, other corresponding stochastic arithmetic units (like hypertangent) also need to be modified if they are cascaded with the APC (see Fig. 2(f)).

III. The Proposed Stochastic DNN

A. *Challenges of Stochastic DNN Design*

As discussed in Section I, the existing DNN hardware suffers from high complexity problem because of the intensive multiplication. Consider the stochastic multipliers (and many other stochastic arithmetic units) have ultra-low hardware cost, SC seems a very promising solution to build large-scale fully-parallel area-efficient DNN hardware. However, the direct use of stochastic arithmetic units in Fig. 2 cannot result in a high-performance DNN design due to two challenges: First, recall that SC is actually a type of approximation computing technique using probabilistic theory; hence the outputs from the stochastic arithmetic units are the approximated values instead of exact outcomes. Consider the modern DNN models typically have multiple layers with hundreds or thousands of neurons, such accumulation of approximation errors, if not being properly addressed, may cause severe accuracy loss for the entire task. Second, though SC can significantly reduce the hardware cost of datapath, it also needs the extra data interface to convert the source data from binary computing domain to stochastic computing domain (B-to-S). As analyzed and reported in [7], such interface is very area and power consuming since it requires large amount of resource to generate N bit-streams for an N-input DNN, where N is usually very large. Therefore, such large overhead in peripheral circuit, if not being greatly reduced, would eliminate the low-cost benefit provided by the SC technique.

B. *Accuracy-Aware Stochastic DNN Datapath*

To address the aforementioned challenges, we develop novel optimization techniques to improve both the accuracy and hardware performance of stochastic DNNs. In this subsection, we first present the accuracy-aware stochastic DNN datapath.

***Use of APC*:** As indicated in [8], when a stochastic design contains amounts of addition (e.g. extensive accumulation or large adder tree), it is very likely that this design would have significant accuracy loss if the traditional MUX-based stochastic adder (Fig. 2(c)) is used. This is because the down-scaling effect incurred by the stochastic adder in Fig. 2(c), if being greatly accumulated when amounts of additions are needed, can cause very severe precision loss. Clearly, as seen in (1), since n is typically very large for DNNs, building stochastic DNNs on the MUX-based adders would result in inevitably large accuracy drop.

Based on the above analysis and observation, the proposed stochastic DNN utilizes APC as the component adder. Because the addition performed by the APC is the non-scaled version, the potential precision loss can be completely avoided. Though the hardware complexity of APC is larger than MUX, the overall resource cost of stochastic DNN datapath is still quite low since the most resource-consuming multiplier is now implemented by the simple XNOR gate. Also notice that when APC is used as stochastic adder, the cascaded stochastic activation function block, such as tanh(•), also needs to be modified to be compatible for the integer-input bit-output scenario (see Fig. 2(f)).

***Layer-wise Weight-Scaling Scheme*:** Besides the stochastic arithmetic units (such as adder), the range of input is another important factor that affects the accuracy of stochastic circuits. For instance, as theoretically analyzed in [9], the variance of representation error by the bipolar SC varies with the represented number. More specifically, for the number z, the variance of error for using bipolar SC to represent it is $4(Z/L)(1-Z/L)/L$. Clearly, when z is close to 0, the inaccuracy incurred by the approximation of SC tends to achieve maximum.

Unfortunately, when applying stochastic computing to DNN design, the absolute values of much of the numbers that are represented by the bit-streams are just very small. More specifically, from the perspective of statistics, the weights of the well-trained DNN models tend to be close to zero after training. Fig. 3 shows the layer-wise weight distribution of an example trained 784-100-200-10 DNN model on MNIST dataset [10]. As seen from this figure, the values of most of the weights in all the layers locate around zero. Therefore, if we just simply convert the weights of trained DNN from binary computing domain to stochastic computing domain, it would cause severe accuracy loss even the stochastic datapath uses the high-accuracy APC.

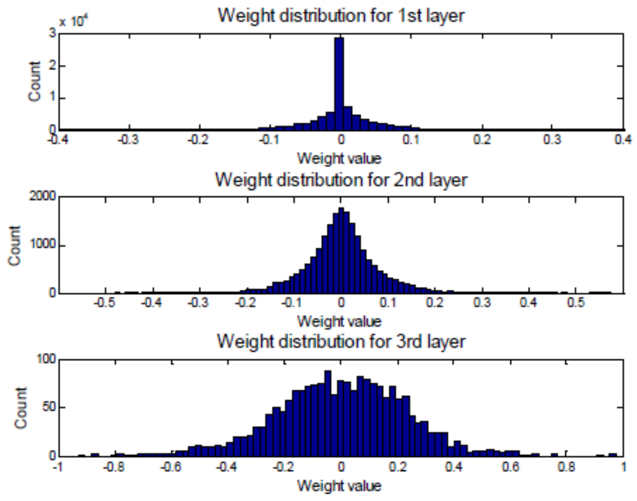

Fig. 3. Weight distributions in different layers of a trained 784-100-200-10 DNN model.

Targeting to this problem, we propose a layer-wise weight-scaling scheme to reduce the computation error of stochastic DNN without extra hardware cost. The key idea here is to scale up the weights of DNN model with the purpose of removing the “near-zero” phenomenon. More specifically, before being converted to bipolar SC format, the weights in the trained DNN model need to be scaled up by certain coefficients first. Such scaling operation can be performed off-line since the DNN models are usually trained on CPU/GPU. Therefore, the bit-streams that represent the weights actually carry the information of the scaled weights. Notice that here the scaling schemes vary for different layers. This is because since the weight distributions vary for different layers (as illustrated in Fig. 3), each layer in the same DNN model may have its specific proper scaling scheme to minimize its computation error.

The simulation results in Table I for the aforementioned example DNN model verify this point. As shown in Table I, the scaling coefficients corresponding to the minimum *standard derivation* of computation error (marked as red) vary for different layers. As a result, different layers should choose their proper scaling coefficients to maximally reduce the approximation errors incurred by the use of SC. Notice that here since the *mean* of computation errors for the outputs of one layer is always very close to zero, the standard derivation of those errors is used to measure the accuracy of the computation in each layer of stochastic DNN. Also notice that the scaling coefficients are usually selected as the power of 2 to facilitate the hardware design.

TABLE I THE STANDARD DEVIATION (σ) OF COMPUTATION ERROR WITH DIFFERENT SCALING COEFFICIENTS

Layer	Scaling Coefficient			
	1	2	4	8
1st	0.598	0.377	0.201	**0.129**
2nd	0.311	0.192	**0.125**	0.147
3rd	0.431	**0.671***	3.388	10.01

*There is no scaled-down compensation for the 3rd layer since it is not involved with activation function. Hence the minimum σ should be the one after divided by the corresponding scaling coefficient (0.671/2<0.431/1).

After using the layer-wise weight-scaling schemes, the datapath of the stochastic DNN need to be modified accordingly. This is because in order to guarantee the validity of computation in each layer, the scaling-up effect should be compensated. More specifically, x_j^{k+1} in (1) should remain the same after scaling up $w_{i,j}^k$. This requirement can be easily satisfied with the slight modification of the combinational logic of the activation function block.

C. Area-efficient B-to-S Interface Module

As analyzed in Section III-A, the high complexity of B-to-S interface is another main challenge for designing efficient stochastic DNN. In general, the resource cost of B-to-S module is proportional to the input-size of stochastic datapath since each input needs its corresponding bit-stream. Consider the number of neurons in the first layer of DNNs is typically very large for various artificial intelligence applications; the B-to-S modules in the stochastic DNN systems have particularly high area- and power-consumption.

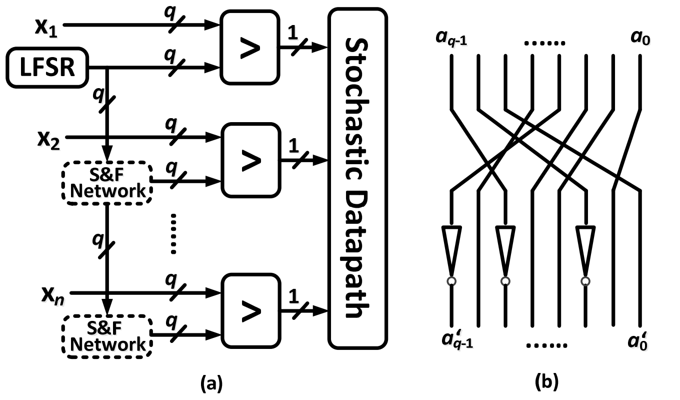

Fig. 4. (a) The proposed B-to-S module with LFSR-sharing scheme and S&F network. (b) The inner architecture of an example S&F network. The shuffling and flipping patterns vary with different inputs.

In this subsection we propose a novel low-complexity architecture for B-to-S module by using linear feedback shift register (LFSR)-sharing scheme and shuffle-flip (S&F) network. Fig. 4(a) shows the overall architecture of the proposed B-to-S module. Here the key idea in this design includes two aspects: 1) First, all the input data ports now share the same LFSR. In other words, the random resource is reduced from n to 1, thereby significantly reducing the resource cost. 2) A dedicated S&F network is designed to guarantee the single LFSR in Fig. 4 can provide *near-independent* randomness to generate different bit-streams. Conventionally, the direct LFSR-sharing strategy in a multi-input SC system would result in severe computation accuracy loss. This is because the functional validity of SC is based on the assumption of using independent input data while sharing the LFSR imposes very strong correlation among the bit-streams that are generated from the same LFSR. To address this problem, we propose the S&F network to eliminate such potential correlation. Fig. 4(b) illustrates the inner architecture of one example S&F network for q-bit input data. From this figure it is seen that the bits of LFSR outputs are irregularly shuffled to transform the original number generated from LFSR to another number. Notice that here irregular shuffling is very necessary since it can maximally avoid any regular patterns that may re-introduce the correlation. Table II shows the standard derivation of computation error for different layers when shuffling the output of LSFR for aforementioned example DNN in Section III-B. It can be seen that the use of shuffling scheme indeed keeps the computation errors in some range, but still makes those errors larger than the case of using n individual LFSRs (see Table I). This is mainly due to the smaller sample space incurred by using shared LFSR instead of individual LFSR for random number generation. More specifically, when n copies of q-bit LFSR are used for providing pseudo random numbers for n input data, the random number for generating each bit-stream has sample space as [0, 2^q-1]. However, when shuffling scheme is used in LFSR-sharing case, for any random number shuffled from LFSR, the number of its possible values is only $\binom{q}{m}$, where m is the number of bit "1" in the current random number generated from LFSR. Clearly, this largely narrowed sample space greatly reduces the randomness, thereby resulting considerable computation error.

TABLE II THE STANDARD DEVIATION (σ) OF COMPUTATION ERROR WITH DIFFERENT SCALING COEFFICIENTS (SHARED LFSR WITH SHUFFLE NETWORK)

Layer	Scaling Coefficient			
	1	2	4	8
1st	0.678	0.573	0.501	0.442
2nd	0.399	0.374	0.303	0.327
3rd	0.473	0.675	3.35	10.08

To address this problem, we propose to flip part of the shuffled bits from LFSR to further improve the randomness. As seen in Fig. 4(b), after the bits from LFSR are shuffled, part of them are flipped by simple inverters. Compared with the previously described shuffle-only scheme, the shuffle-flip scheme enables the bit-streams for different input data being actually generated by using the random numbers from different sample spaces, thereby enhancing the randomness provided by the he B-to-S module. Also, notice that here in order to maximize the randomness, the S&F network that is affiliated with each input data port has different shuffling and bit-flipping patterns. In particular, the positions of bits that need to be flipped vary for different input data ports. As a result, the random numbers that are fed to different comparators are very different and can be viewed as non-correlated. Table III shows the standard deviation of computation error for different layers when shuffling and flipping the output of shared LSFR for aforementioned example DNN in Section III-B. From this table it is seen using the S&F network indeed achieves the same level of computation error as using *n* copies of individual LFSRs (see Table I). Consider the hardware cost of inverter is ultra-low, the proposed LFSR-sharing S&F-based B-to-S module enables significant reduction of resource cost (from hundreds/thousands of LFSR to 1) without increasing correlation, thereby greatly saving hardware while retaining accuracy.

TABLE III THE STANDARD DEVIATION (σ) OF COMPUTATION ERROR WITH DIFFERENT SCALING COEFFICIENTS (SHARED LFSR WITH S&F NETWORK)

Layer	Scaling Coefficient			
	1	2	4	8
1st	0.601	0.361	0.195	0.132
2nd	0.324	0.181	0.120	0.147
3rd	0.439	0.665	3.417	9.873

IV. PERFORMANCE ANALYSIS

A. Test Accuracy of Stochastic DNN

Fig. 5 shows the test accuracy of the example trained 784-100-200-10 DNN on MNIST handwritten digit dataset using different types of implementations. Here BC and SC indicate conventional two's complement binary radix-based computing and stochastic computing, respectively. For fair comparison, the quantization scheme for fixed-point BC and SC designs are set as the same 2^{-10} representation scheme. In addition, shuffle-only SC and shuffle-flip SC represent the stochastic DNNs using shuffle-only network and shuffle-flip network B-to-S interface modules, respectively.

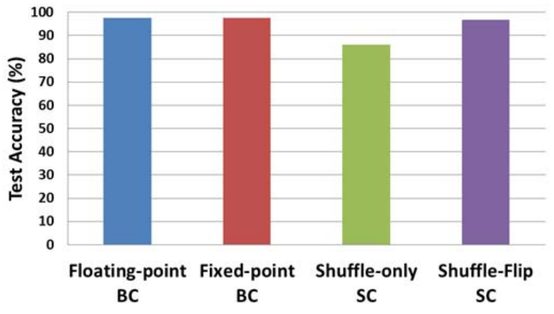

Fig. 5. Comparison of test accuracy among different DNNs using binary radix computing (BC) and stochastic computing (SC) on MNIST dataset. DNN size is 784-100-200-10. Here precision for fixed-point BC and SC is 2^{-10}.

From this figure it is seen that the test accuracy of the proposed stochastic DNN with LFSR-sharing shuffle-flip network B-to-S module is very close to that of the baseline floating-point-based design. Also, notice that when bit-flip strategy is not used (for shuffle-only SC), the stochastic DNN suffers from significant accuracy drop. Hence it is concluded that the proposed shuffle-flip network is essential to guarantee the high accuracy of stochastic DNN designs.

B. Hardware Performance of Stochastic DNN

Based on the proposed optimization approaches for datapath and B-to-S module in Section III, the hardware architecture of the above example stochastic 784-100-200-10 DNN is developed and coded with Verilog HDL. Table IV shows the synthesis results using 45nm CMOS technology. As shown in this table, at 25ºC temperature and 1.1V supply voltage, the proposed non-pipeline stochastic DNN design runs at 450MHz clock frequency while the conventional binary radix-based design needs at least 6-stage pipeline to achieve the same clock speed. In addition, the proposed fully-parallel design consumes only 0.839 million gate counts, thereby enabling 47X reduction in hardware cost than the conventional binary radix-based design with negligible accuracy loss (see Fig. 5). Therefore, the stochastic DNN can be a very promising solution for deploying DNNs in the area-constraint resource-constrained applications, such as edge computing, Internet of things (IoT) etc.

TABLE IV HARDWARE PERFORMANCE OF DNN DESIGNS

Design	Proposed Stochastic DNN	Binary radix-based DNN
DNN Model Size	784-100-200-10	
Data Precision	2^{-10} (length-1024 bit-stream)	2^{-10} (10-bit quantization)
Technology (nm)	CMOS 45nm	
Parallelism	Fully parallel	
Pipeline Stage	Non-pipeline	6
Clock Frequency	450MHz	
Gate Counts	0.839 Million	39.5 Million

V. CONCLUSION

This paper presents a fully-parallel area-efficient deep neural network design using stochastic computing. By using the accuracy-aware datapath and low-complexity randomness-sufficient B-to-S module, the proposed stochastic DNN design achieves very high area efficiency with negligible accuracy loss, thereby paving the way of deploying DNN in the area-constrained resource-constrained applications.

REFERENCES

[1] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.
[2] W. Dally, "High-performance hardware for machine learning," in Proc. of 29th Annual Conference on Neural Information Processing Systems (NIPS), Dec. 2015.
[3] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu and Y. Chen, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. of ACM Intl. Conf. on Architectural Support for Programming Languages and Operating System (ASPLOS)*, March 2014.
[4] N. P. Jouppi, et.al . "In-datacenter performance analysis of a tensor processing unit", to appear in *Proc. of ISCA* 2017.
[5] B. Brown and H. Card, "Stochastic neural computation I: computational elements," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 891-905, Sept. 2001.
[6] T. Pai-Shun and J. P. Hayes, "Stochastic logic realization of matrix operations," in *Proc. of Digital System Design*, pp. 356-364, Oct. 2014.
[7] P. Knag, W. Lu, and Z. Zhang, "A native stochastic computing architecture enabled by memristors," *IEEE Trans. Nanotechnology*, vol. 13, no. 2, pp. 283-293, Mar. 2014.
[8] B. Yuan and Y. Wang, "High-Accuracy FIR Filter Design using Stochastic Computing," in *Proc. of IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2016.
[9] B. Gaines, "Stochastic computing systems," Advances in Information Systems Science, vol. 2, no. 2, pp. 37–172, 1969.
[10] Y. LeCun, C. Cortes and C. J. Burges, "The MNIST database of handwritten digits," 1998.