Designing Reconfigurable Large-Scale Deep Learning Systems Using Stochastic Computing

Ao Ren[*], Zhe Li[*], Yanzhi Wang, Qinru Qiu
Dept. of Electrical Engineering & Computer Science
Syracuse University
Syracuse, NY 13244, USA
{aren, zli89, ywang393, qiqiu}@syr.edu

Bo Yuan
Department of Electrical Engineering
City University of New York, City College
New York, NY 10031, USA
byuan@ccny.cuny.edu

***Abstract*—Deep Learning, as an important branch of machine learning and neural network, is playing an increasingly important role in a number of fields like computer vision, natural language processing, etc. However, large-scale deep learning systems mainly operate in high-performance server clusters, thus restricting the application extensions to personal or mobile devices. The solution proposed in this paper is taking advantage of the fantastic features of stochastic computing methods. Stochastic computing is a type of data representation and processing technique, which uses a binary bit stream to represent a probability number (by counting the number of ones in this bit stream). In the stochastic computing area, some key arithmetic operations such as additions or multiplications can be implemented with very simple components like AND gates or multiplexers, respectively. Thus it provides an immense design space for integrating a large amount of neurons and enabling fully parallel and scalable hardware implementations of large-scale deep learning systems. In this paper, we present a reconfigurable large-scale deep learning system based on stochastic computing technologies, including the design of the neuron, the convolution function, the back-propagation function and some other basic operations. And the network-on-chip technique is also proposed in this paper to achieve the goal of implementing a large-scale hardware system. Our experiments validate the functionality of reconfigurable deep learning systems using stochastic computing, and demonstrate that when the bit streams are set to be 8192 bits, classification of MNIST digits by stochastic computing can perform as low error rate as that by normal arithmetic operations.**

Keywords—Stochastic computing; deep learning; neuron; reconfigurable; large-scale.

I. INTRODUCTION

In recent years, the machine learning technology lends the emerging of autonomous systems, such as unmanned vehicles, robotics, and cognitive wearable devices. *Deep Learning*, as a new branch of machine learning research, is representation learning based and capable of overcoming the limitations of processing natural data in their raw form, which are faced by the conventional machine learning techniques. Recently, deep learning has been proven to be an effective technique that is capable of handling unstructured data for both supervised and unsupervised learning [1]-[3]. The deep layered structure significantly improves learning performance, however, also increases memory and computation complexity.

Nowadays, large-scale deep neural networks mainly operate in high-performance server clusters, GPU or FPGA clusters [4]-[11], and have attracted many research attentions on enhancing parallelism and scalability, and reducing power consumptions as well as synchronization overheads [4][6][10]. However, the computing requirement at server or GPU cluster level implies high power/energy consumptions and prohibits the wide application of deep learning systems in personal systems or mobile devices. One promising method to overcome this shortcoming is designing specific hardware-based deep learning systems, in order to exploit the maximum degree of parallelism and achieve significant reduction in power/energy.

IBM TrueNorth neuro-synaptic processor [15], although in essence not for deep learning applications but for spiking neural networks, is a breakthrough effort in this direction. TrueNorth has integrated 4,096 *physical neurons*, each accommodating up to 256 *virtual neurons* in the time-multiplexed manner. This time-multiplexing operating manner has restricted the parallelism degree and performance. Moreover, a desirable hardware-based deep learning system should be able to possess online training/learning capability and re-configurability for different applications, whose properties are typically missing in state-of-the-art hardware implementations of neural networks [12]-[15]. The challenges to design highly scalable and parallel hardware deep learning systems that also provide online learning capability necessitate the investigation on novel computing paradigm spanning hardware, algorithm, and application.

Stochastic Computing (*SC*) [16][17], as a unique data representation and processing technique, has the potential to enable the design of fully parallel and scalable hardware implementations of large-scale deep learning systems, due to the following reasons: First, many complex arithmetic operations can be implemented with very simple hardware logic in stochastic computing framework [16], which offers an immense design space for (i) neuron integrations due to the significantly reduced area per neuron and (ii) performance optimizations with respect to resiliency, power/energy, and speed by trading off the abundant area budget. Second, stochastic computing has inherently strong fault tolerance against transient and soft errors [18]-[20], because it processes data in the form of bit streams that are interpreted as probabilities in contrast to the traditional computing that operates on the positional representation of data. Based on these encouraging characteristics, stochastic computing can potentially trigger a revolutionary reshaping of hardware design of large-scale deep learning systems with orders-of-

[*] Ao Ren and Zhe Li contributed equally to this work

magnitude improvements in scalability, performance, power/energy efficiency, and resiliency.

In this paper, we investigate highly scalable hardware-based deep learning systems using stochastic computing. Our target is to develop a universal platform of hardware deep learning systems for various applications, which supports online learning capability and re-configurability for various types of neural networks such as *deep belief network* (DBN) or *convolutional neural network* (ConvNN) and for different applications. The target platform resembles FPFA for digital circuits and systems. In order to achieve this goal as well as high performance, low energy/power consumption, high scalability and resiliency, we propose novel hardware design of neurons using stochastic computing. The major computing tasks, e.g., inference and learning, of the neuron are performed in the stochastic computing domain based on our novel designs of stochastic computing-based inner product calculation. The proposed neuron achieves online learning capability and re-configurability, both for different types of neural networks such as DBN and ConvNN, and for different applications. It achieves extremely small hardware footprint and power/energy consumptions. At the network-on-chip (NoC) level, which coordinates neuron communications, we propose high-efficient, low-hardware cost, and low-power NoC structure connecting and coordinating a large number of stochastic computing-based neurons.

Extensive experimental results have been conducted on both the hardware design and the testing results on various deep learning benchmarks, e.g., classifying MNIST digits by ConvNN with different lengths of bit streams. Experimental results demonstrate that the error rate can be restricted around 1.54% which is the base line performance of trained ConvNN model when the lengths of bit streams are set to be 8192 (2^{13}) bits.

The rest of the paper is organized as follows: Section 2 discusses the background on deep learning and stochastic computing techniques. Section 3 presents design and optimization of the proposed reconfigurable neuron along with its key operations, using stochastic computing. The proposed NoC structure for coordinating stochastic computing-based neurons in the deep learning network is presented in this section as well. Section 4 presents experimental results. Finally, Section 5 concludes the paper.

II. BACKGROUND ON DEEP LEARNING AND STOCHASTIC COMPUTING

A. *Deep Learning*

In recent years, various deep learning architectures such as deep neural networks, convolutional deep neural networks, deep belief networks and recurrent neural networks have been applied to fields like computer vision, automatic speech recognition, natural language processing, audio recognition and bioinformatics where they have been shown to produce state-of-the-art results on various tasks. Deep learning intrinsically is deeply and hierarchically structured that attempt to model high-level abstractions in data using multiple processing layers composed of multiple non-linear transformations. In this work, we take deep convolutional neural networks as example to show the availability of implementing ConvNN using stochastic computing system.

A ConvNN architecture is formed by a stack of distinct layers that transform the input into an output through a differentiable function. The building blocks of a ConvNN architecture consists of convolutional layer, pooling layer, nonlinear transformation layer, fully connected layer and loss layer. By arranging the topology of above layers, several impressive work built specific architectures such as LeNet [33], and AlexNet [34]. The convolutional layer is the core building block of ConvNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input data. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when they see some specific type of features at some spatial positions in the input.

Another important concept of ConvNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling such as max pooling, average pooling and L2-norm pooling. The pooling layer operates independently on every depth slice of the input and resizes it spatially. The pooling operation provides a form of translation invariance. Nonlinear transformation includes ReLU which is abbreviation of Rectified Linear Units applying neurons the non-saturating activation function $f(x) = \max(0, x)$, hyperbolic tangent (tanh) applying neurons the saturating activation function $f(x) = \tanh(x)$ or $f(x) = |\tanh(x)|$ and sigmoid applying neurons the activation function $f(x) = (1 + e^{-x})^{-1}$. Fully connected layer is a normal neural network layer with its inputs fully connected with its previous layer. The loss layer of ConvNN specifies how the network training penalizes the deviation between the predicted labels and true labels, and it is normally the last layer in the network. Various loss functions, such as softmax loss, sigmoid cross-entropy loss or Euclidean loss, may be used there for different tasks.

B. *Stochastic Computing*

Deviated from the conventional binary computing (referred as *conventional computing*), *stochastic computing* (SC) represents any number using a stream of bits. Here the value of real number x in the unit interval is interpreted by the ratio of bit-1 in the entire bit-stream, i.e., $\mathrm{P}(X = 1)$. For instance, the 8-bit sequence 00100101 containing three 1s denotes $\mathrm{x} = \mathrm{P}(\mathrm{X} = 1) = 3/8 = 0.375$. Since each bit has the same weight, number representation in stochastic computing is unary and hence enables different interpretations for the same value. Besides this unipolar coding format [16], bipolar coding format [16] is another popular number representation scheme in stochastic computing. In the scenario of bipolar coding, the relationship between x and $\mathrm{P}(X = 1)$ becomes $\mathrm{P}(X = 1) = (x + 1)/2$, which enables the stochastic representation for negative number. Notice that for either unipolar or bipolar coding format, the represented number ranges in [0, 1] or [-1, 1]. To represent a number beyond this range, a pre-scaling operation [21] or integer bit-stream based representation [22] can be used to relax this constraint.

A major advantage of stochastic computing is its ultra-low hardware cost: Many complicated arithmetic functions can now be implemented with very simple logic circuits. For instance, as shown in Fig. 1, the real multiplication can be performed with an AND gate in the unipolar coding form since $c = P(C = 1) = P(A = 1)P(B = 1) = ab$ or with an XNOR gate in bipolar coding form since $c = 2P(C = 1) - 1 = 2(P(A = 1)P(B = 1) + P(A = 0)P(B = 0)) - 1 = (2P(A = 1) - 1)(2P(B = 1) - 1) = ab$. Another example is regarding the adder, which can be simply implemented with a multiplexer (see Fig. 2) in the scenario of stochastic computing, for $c = P(C = 1) = 1/2(P(A = 1) + 1/2P(B = 1) = 1/2(a + b)$. Additionally, the addition in the bipolar form uses this multiplexer as well, since $c = 2P(C = 1) - 1 = 2(1/2(P(A = 1) + 1/2P(B = 1)) - 1 = 1/2(2P(A = 1) - 1) + (2P(B = 1) - 1)) = 1/2(a + b)$.

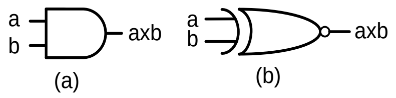


Fig. 1. Stochastic multiplication using: (a) unipolar encoding (b) bipolar encoding.

In general, such significant saving in hardware resource makes stochastic computing circuits well-suited for the area-constrained applications, such as signal sensing and processing in wearable devices. Besides, the abundant budget on area offers immense design space in optimizing hardware performance in terms of power, latency and speed via efficient tradeoffs between area and those metrics, thereby implying the potential application of stochastic computing in large-scale systems that requires massive parallelism for basic computing units.

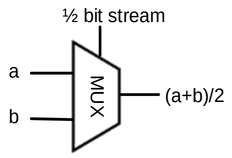


Fig. 2. Scaled addition in stochastic computing.

Another advantage of stochastic computing is its inherent error-resilience. By nature, the redundant representation of stochastic computing translates to the strong capability for tolerating transient error and soft error (bit-flipping) since each bit has the same weight in bit-stream. For instance, as reported in [18]-[20], compared to their conventional computing counterparts, the stochastic digital signal processing component shows much better error-resilience capability, which is attractive for the emerging noise-rich deep nanoscale CMOS era. Inspired by these encouraging characteristics of stochastic computing, prior efforts have investigated stochastic system in different practical applications, including image processing [21][23], control systems [25], etc.

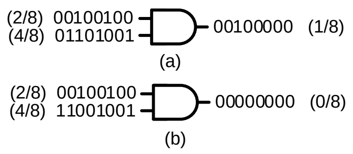


Fig. 3. Stochastic unipolar multiplication (a) expected result (b) unexpected result.

However, despite its advantage on low complexity and high error-resilience, stochastic computing suffers some drawbacks. As Fig. 3 displays, both bit streams 01101001 as well as 11001001 can represent 4/8, but they definitely result in different results. This example illustrates a key problem that how to generate good stochastic numbers to decrease the inaccuracy as possible. According to [30], there are two sources of inaccuracy: random fluctuations in stochastic number representation and correlations among the numbers that participate in the calculation. Reference [30] points out that two bit streams $S1=S1(n)S1(n\text{-}1)\ldots S1(1)$ and $S2=S2(n)S2(n\text{-}1)\ldots S2(1)$ are said to be uncorrelated if and only if:

$$\sum_{i=1}^{n} S1(i)S2(i) = \frac{\sum_{i=1}^{n} S1(i) \times \sum_{i=1}^{n} S2(i)}{n} \quad (1)$$

Thus the two sources of inaccuracy mentioned above are correlated to the randomness and lengths of the stochastic numbers. Therefore, a good random number generator and long enough bit-stream are crucial for reliable stochastic computing. High-efficiency and low-area designs of random number generators, e.g., the Intel random number generator [35], can be utilized to fulfill this goal.

III. DESIGN AND OPTIMIZATION OF HARDWARE NEURON USING STOCHASTIC COMPUTING

A. Stochastic Computing-Based Inner Product Calculation

The kernel function in neurons of deep learning systems is to calculate the inner product of input signals and weight vector of the neuron, and hence it is necessary to investigate the low-complexity and high-efficiency implementation of inner product calculations. Based on its mathematic form, the fully-parallel hardware implementation of size-N inner product requires an array of N multipliers and a depth-$\log_2 N$ adder tree. The huge hardware cost of multiplier arrays and adder tree with a large N (when neuron connectivity is high) prohibits the implementation of inner product calculation with fully parallel style in the conventional computing domain. On the other hand, implementing the inner product calculation using a sequential style will require significantly lower hardware cost (and power consumption) but exhibit higher latency (lower performance).

In this work we develop low-complexity fully-parallel inner product calculation in stochastic computing domain. Fig. 4 shows the hardware architecture for size-N stochastic inner product calculation, which contains an array of XNOR-based multipliers and an adder tree. Here the XNOR gate, instead of AND gate, is used for performing multiplication since bipolar-form of stochastic computing is needed to process negative numbers. For the adder tree, each component adder is the multiplexer shown in Fig. 2.

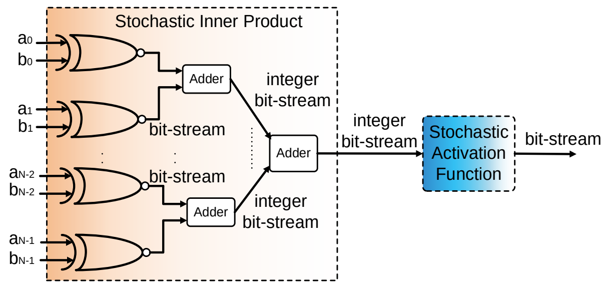


Fig. 4. Hardware architecture of size-N stochastic inner product

B. *Other Basic Operations in Neurons for Deep Learning*

Besides the inner product, some other basic operations, e.g., pooling, activation function, back propagation, are also required in the neurons of different types of deep learning systems. In this work, we investigated the designs of these important operations in stochastic computing.

1) *Pooling*

Pooling is an important operation that enables significant reduction of inter-layer connection in ConvNN as well as retaining the translation invariance of the extracted features. In general, pooling can be performed in the form of *max pooling* ($s_j = max(a_i)$) or *average pooling* ($s_j = mean(a_i)$) for $i \in R_j$ where a_i is the activation result of the i-th element in the feature map, R_j is the current pooling region of feature map, and s_j is the corresponding pooling output.

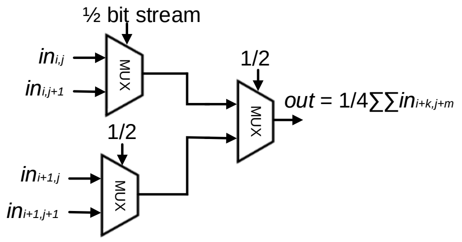


Fig. 5. Stochastic computing circuit for 2x2 average pooling

In the proposed stochastic-computing-based neuron for deep learning systems, average pooling is adopted as the preferred pooling technique since it can be easily implemented using very simple logic circuit. For instance, as seen in Fig. 5, the stochastic arithmetic mean over a 2×2 region only requires three multiplexers, which is much simpler than the implementation in the conventional computing domain. In addition, the use of average pooling does not incur severe accuracy loss as compared to using max pooling. As indicated in [25], for a ConvNN trained for CIFAR 10 dataset, average pooling results in the similar test error (19.24%) compared to the case using max pooling (19.40%) but with lower convergence speed. However, since stochastic computing enables much higher parallelism degree in hardware and stochastic circuits for average pooling is very simple, such drawback on speed can be easily avoided.

2) *Nonlinear Activation*

Nonlinear activation is widely used in the inference and learning phases of different types of deep learning networks [1]-[4]. Usually following inner product calculation, the nonlinear activation is the key operation that enhances the representation capability of neuron/neural network. In general, although numerous functions can offer nonlinear transformation, the most popular activation functions in practice are *sigmoid* as $y = f(x) = 1/(1 + e^{-x})$, *tanh* as $y = f(x) = (e^x - e^{-x})/(e^x + e^{-x})$, and Rectified Linear Unit (ReLU) as $f(x) = max(x, 0)$. Here sigmoid and tanh are used as activation functions in both DBN and ConvNN, while ReLU is only used in ConvNN.

In this work, for the convenience of maximizing reconfigurability of the neuron, the tanh function is selected as the nonlinear activation function for all types of deep learning networks. The reasons are: (i) Replacing sigmoid or ReLU function by tanh function does not cause accuracy loss in DBN or ConvNN. As reported in [26], with different configurations of network size and training/test datasets, the replacement of ReLU by tanh function in ConvNN always results in very close test error. For instance, a four-layer ConvNN achieves 25% classification error on CIFAR-10 dataset with either ReLU or tanh function [26]. (ii) In the scenario of stochastic computing, the tanh function can be easily implemented by a symmetric *K*-state finite state machine (FSM) which is shown in

Fig. 6 [16]. Compared to the piecewise linear approximation (PLAN)-based implementation [27] in conventional computing, the *K*-state FSM design for tanh function in stochastic computing has much lower hardware cost.

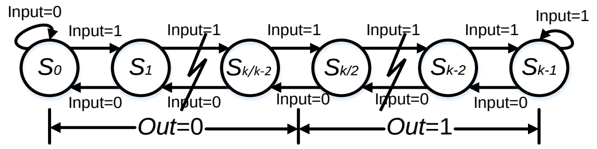


Fig. 6. Stochastic design for tanh function using FSM

According to [17], the *K*-state FSM and tanh have the following relationship:

$$\mathrm{Stanh}(K, x) = \tanh(\frac{K}{2}x) \qquad (2)$$

where *K* is the number of states, x is the input stream sent to the FSM $\mathrm{Stanh}(K, x)$. For a specific stochastic stream x' that we want to calculate $\tanh(x')$, we can set

$$x = 2x'/K. \qquad (3)$$

and thereby $\tanh(x') = \tanh(\frac{K}{2}x)$.

Hence, in order to calculate the activation result of the input stream x', it should be preprocessed by Eqn. (3), and then x is sent into the FSM Stanh . Additionally, the calculation accuracy of the *K*-state FSM is decided by (i) the length of the stochastic bit stream and (ii) the number of states *K*. These two parameters are properly optimized in our experiments which are discussed in Section 4.

3) *Convolution*

Convolution (filtering) is the critical and unique operation in ConvNN. By exploiting local similarity of inputs, the convolution operation enables significant reduction of the required inter-layer connection, especially when the inputs to the networks are the inherently stationary natural images.

This paper used the stochastic inner product circuit to perform the convolution operation, which is essentially just the inner product between the convolution kernel (filter) and the local regions of input signals. Therefore, the convolution operation in ConvNN can also be implemented with very simple stochastic circuits as shown in Fig. 4.

4) *Back-Propagation (BP) for Learning*

To date, the underlying leaning strategy for all deep learning networks is the *back-propagation* (BP) approach [1][2]. Consider any neuron n_c in layer *i*. The BP algorithm first calculates/updates the discrepancy δ_c^i of the neuron n_c as:

$$\delta_c^i \leftarrow (\boldsymbol{w}_c^{(i+1)} \cdot \boldsymbol{\delta}^{(i+1)}) f'(z_c) \qquad (4)$$

where $\boldsymbol{w}_c^{(i+1)}$ is the vector of the connection weights between n_c and all neurons in layer $(i+1)$, and $\boldsymbol{\delta}^{(i+1)}$ is the discrepancy vector of neurons in layer $(i+1)$. In addition, z_c is the input of the activation function $f(x)$ in the n_c, and $f'(x)$ is the derivative of $f(x)$.

After calculating δ_c^i, the neuron n_c will update connection weights with the neurons in layer $(i-1)$. Let w_{bc} denote the weight between n_c and neuron n_b in layer $(i-1)$, then w_{bc} can be updated as follows:

$$\Delta w_{bc} \leftarrow \Delta w_{bc} + a_b \delta_c^i \tag{5}$$

$$w_{bc} \leftarrow w_{bc} - \Delta w_{bc} \tag{6}$$

where a_b is the output from the nonlinear activation function $f(x)$ of n_b.

In the scenario of stochastic computing, all operations in Equations (2)-(4), such as inner product calculation and multiplication, can be implemented in very simple logic circuits that have been analyzed above. In addition, the hardware architectures for $f(x) = \tanh(x)$ and $f'(x) = 1 - \tanh^2(x)$ can also be constructed using stochastic computing. Therefore, the entire back-propagation procedure can be implemented with simple fully parallel stochastic circuits, thereby leading to a significant acceleration of the training procedure of large-scale deep learning systems.

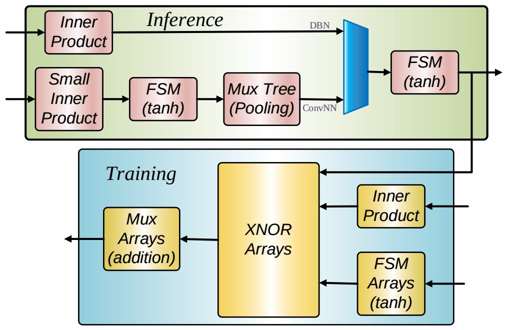


Fig. 7. Reconfigurable neuron diagram

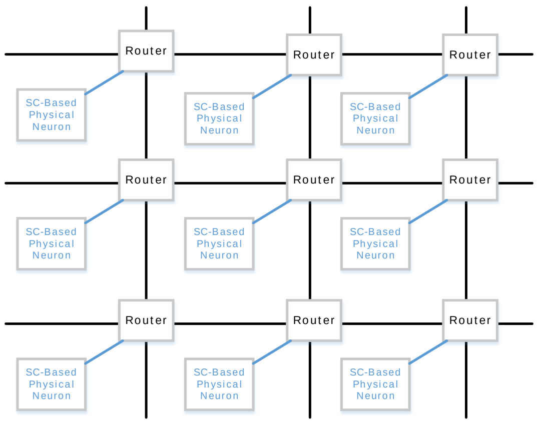


Fig. 8. NoC structure of SC-based neurons

C. Hardware Design of Reconfigurable Neuron Using Stochastic Computing

Based on the designs of fundamental operations, the hardware design of reconfigurable neuron using stochastic computing is developed. Fig. 7 illustrates the overall diagram of the proposed neuron design, integrating inner product calculation, activation function, average pooling, etc., all based on stochastic computing. The proposed neuron could be reconfigured for different types of neural networks (e.g., DBN, ConvNN) or for different layers of neural network if it uses heterogeneous neurons at different layers (e.g., ConvNN). Because we only use tanh function for activation function, enabling reconfigurability for DBN and ConvNN will not cause a much hardware overhead.

D. Design of NoC Structure for Deep Learning Systems

For hardware implementation of stochastic computing-based large-scale deep learning systems, we propose NoC communication structure as shown in Fig. 8 for coordinating and supporting communications among neurons. The reason is, compared with the simpler shared bus structure among neurons, a mesh network-based NoC structure is more suitable for large-scale deep learning systems due to its higher parallelism degree in neuron communications and higher scalability [28][29].

Fig. 8 illustrates that the NoC structure connects adjacent core tiles to form a 2D mesh network. Each core tile comprises one hardware implementation of stochastic computing-based neuron, i.e., the physical neuron, and an NoC router. Please note that due to the extremely small footprint of a physical neuron, a single neuron (at the software/algorithm level) can be mapped to a physical neuron (at hardware level) without using the notion of virtual neuron [15], in which multiple virtual neurons share one physical neuron and operate in a time-multiplexed manner. On the other hand, NoC routers are responsible for routing neuron communications, which are encoded in packets with destination neuron addresses, to nearby routers. A neuron communication packet will eventually be routed from a source neuron to a set of destination neurons according to neural network connectivity.

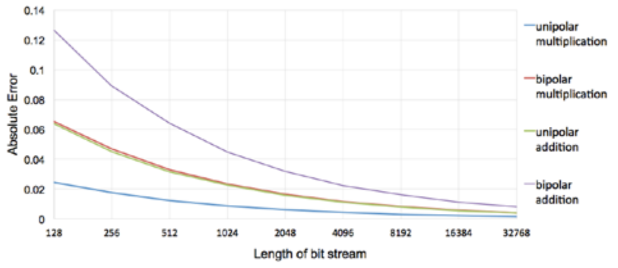
IV. Experiments And Analysis

To investigate the functionality and performance of our proposed stochastic computing-based deep learning system, a set of simulations are conducted. We first do experiments to depict the relations between the accuracy and design parameters of stochastic computing-based components inside a neuron (the adder, multiplier and activation function). Then, these components are organized into neurons to be put into the deep learning system presented in this paper.

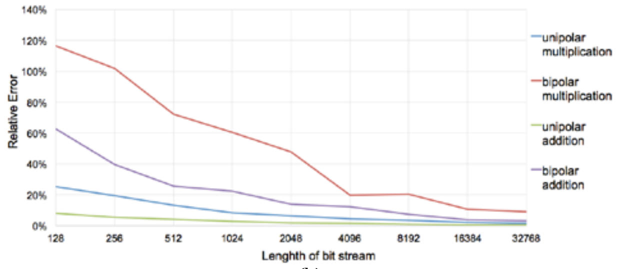
A. Investigations on Stochastic Computing-based Components

The lengths of input bit streams can make a significant effect on the accuracy of stochastic computing. Thus to reach a higher accuracy, it is desirable to increase the bit stream length. However, a longer bit stream will also result in a longer latency. Hence, to resolve this contradiction, we conduct a series of experiments to find out the relation between accuracy and bit stream length. Then, the length of input streams is decided according to the errors.

Fig. 9 displays the relations between computation accuracy and the length of bit streams on some key components in stochastic computing. As expected, both the absolute errors and the relative errors decrease with the increasing of bit stream length, since the randomness of each input bit stream significantly increases as the length becomes longer. Thus, the error of each represented number is reduced. Apparently, bipolar calculations will result in higher error rates than unipolar calculations, and multiplications have higher relative errors but lower absolute errors than additions. Because the multiplication in stochastic computing always results in a smaller result than its inputs, but the addition sometimes can produce a greater result than its inputs. It is obvious that when the result is small, a very small absolute error could result in a big relative error. This figure can also help decide the length of input bit streams sent to neurons. When the length of bit streams is longer than 4096, the relative error of each operation is intuitively acceptable. But its performance when cooperated together and integrated into a deep learning system is still unclear until next subsection.



(a)



(b)

Fig. 9. The length of bit stream versus errors for stochastic computing components (a) for absolute errors (b) for relative errors

Except for the addition and multiplication, the activation function is another important component of a neuron. Eqn. (2) implies that the accuracy of the *K*-state FSM is determined by both the length of input streams and the number of states. In our simulations, we set the length of input streams to 16384 (with the precision of 14 bits), and simulate the accuracy of the FSM with the number of states ranging from 8 to 24.

TABLE I. NUMBER OF STATES VERSUS RELATIVE ERROR FOR TANH CALCULATION

Number of States	Relative Error
8	10.06%
10	8.27%
12	7.43%
14	7.36%
16	7.51%
18	8.07%
20	8.55%
22	9.28%
24	9.95%

The simulation results shown in TABLE I indicate that when the number of states is 14, the most accurate result can be achieved. Please notice that more states not always generate more accurate results, since the pre-processing will reduce the number sent into the FSM by *K*, where *K* is the number of states. As stated above, when the number is small after reduction by *K* with a big *K*, the relative errors will be enlarged. And Fig. 10 depicts the comparison between the FSM-based tanh calculation (Stanh) and actual tanh results (tanh) when the number of states is set to 14 and the input value is in the range of [-1, 1]. From the figure, we know that the FSM-based activation function slightly oscillates around the output curve of tanh, thus this result implies that the Stanh can be a good candidate to replace the tanh in neurons.

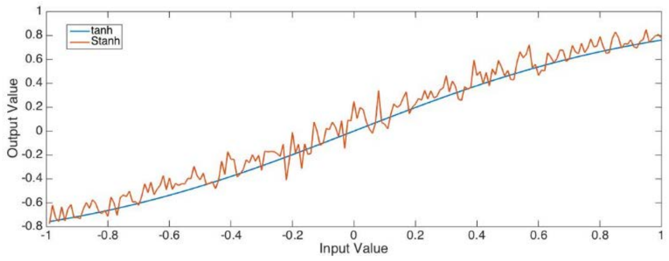


Fig. 10. Illustrating the accuracy of FSM-based tanh activation function (Stanh in the figure) vs. the actual tanh calculation results

B. *Case Study: Classifying MNIST Digits Using Stochastic Computing-Based Convolutional Neural Network*

In this section, we use Theano [31] to solve the MNIST [32] digits classification problem with LeNet5 [33]. The MNIST database has a training set of 60,000 examples, and a test set of 10,000 examples. The images are centered in 28×28 blocks. Since we are not investigating the performance of ConvNN, we just trained the LeNet5 to achieve an acceptable testing error rate. We trained the model 20 epochs with batch size of 500 examples (totally 10000 training examples) and we set the learning rate as 0.1. Then we can achieve a testing error rate of 1.54% with 10000 testing examples. This is our base line performance.

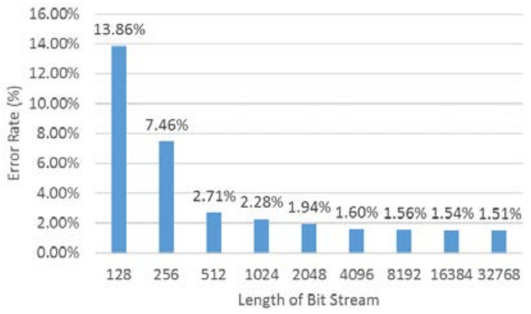


Fig. 11. Simulation result of LeNet5 for MNIST with different lengths of bit streams

Because from Fig. 9, we can see in generally, the bipolar operations' accuracies are worse than unipolar operations. To investigate the worst cases of stochastic-based computing, thus, we replaced addition, multiplication and tanh operators with our stochastic computing-based bipolar operators. Also we scaled the numbers before we use the stochastic operators because they are supposed to handle numbers in the range of [-1,1]. Based on TABLE I. with different lengths of bit streams for addition/multiplication, we achieve different test errors shown in Fig. 11. We can observe that when bit streams representing each number are only 128 bits long, the error rate is very high compared to the base line. And when we increase the lengths of bit streams, the error rates decrease sharply. When each bit stream is about 8000 bits long, the error rate saturates to the baseline error rate of 1.54%. And the error rate of 1.51% with stochastic computing-based operators even outperform the baseline error rate when the bit-stream length is 32768. The reason is that the random errors brought by the stochastic computing-based arithmetic operations compensate themselves during arithmetic operations in a neural network. When a number is represented more randomly, the compensation effect is more visible. Hence the compensation effect leads to the correction to improve the neural network performance when a number is represented extreme randomly (with a very long bit stream).

V. Conclusion

In this paper, we investigate the availability of using stochastic computing technique to implement reconfigurable large-scale deep learning systems. The most important components and operations of a deep learning system, including the neuron, the pooling function, the activation function and the back-propagation function, etc., are all implemented in the form of stochastic computing. Our experiments demonstrate that the error rates decrease with the increasing of the lengths of bit streams. Moreover, when the bit streams are set to be 8192 bits, classification of MNIST digits by stochastic computing can perform as low error rate as that by normal arithmetic operations.

References

[1] Y. Bengio, “Learning deep architectures for AI,” Foundations and trends in Machine Learning, 2009.

[2] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,” in *Proc. of International Conference on Machine Learning* (ICML), 2009.

[3] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Elsevier Neural Networks*, 2015.

[4] R. Raina, A. Madhavan, and A. Y. Ng, “Large-scale deep unsupervised learning using graphics processors,” *Annual International Conference on Machine Learning* (ICML), 2009.

[5] Q. V. Le, “Building high-level features using large scale unsupervised learning,” *Proc. of ICASSP*, 2013.

[6] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, “Project Adam: Building an Efficient and Scalable Deep Learning Training System”, *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.

[7] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, “Large scale distributed deep networks,” *Proc. of Advances in Neural Information Processing Systems* (NIPS), 2012.

[8] A. Coates, B. Huval, T. Wang, D. J. Wu, A. Y. Ng, and B. Catanzaro, “Deep learning with COTS HPC systems,” *Proc. of Advances in Neural Information Processing Systems* (NIPS), 2013.

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” *Proc. of ACM International Conference on Multimedia*, 2014.

[10] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, “Deep, big, simple neural nets for handwritten digital recognition,” *Neural Computation*, 2010.

[11] J. Bergstra, F. Bastien, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, D. Warde-Farley, I. Goodfellow, A. Bergeron, and Y. Bengio, “Theano: Deep learning on GPUs with Python,” *Journal of Machine Learning Research*, 2011.

[12] Y. Wang, T. Tang, L. Xia, B. Li, P. Gu, H. Yang, H. Li, Y. Xie, "Energy Efficient RRAM Spiking Neural Network for Real Time Classification." In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pp. 189-194. ACM, 2015.

[13] O. Bichler, D. Querlioz, S. J. Thorpe, J. P. Bourgoin, C. Gamrat, "Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity." *Neural Networks* 32 (2012): 339-348.

[14] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, A. D. Brown, "Overview of the SpiNNaker System Architecture," in *Computers, IEEE Transactions on*, vol.62, no.12, pp.2454-2467, Dec. 2013.

[15] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface." *Science* 345, no. 6197 (2014): 668-673.

[16] B. Gaines, “Stochastic computing systems,” *Advances in Information Systems Science*, vol. 2, no. 2, pp. 37–172, 1969.

[17] B. D. Brown and H. C. Card, “Stochastic neural computation I: computational elements,” *IEEE Trans. Comput.*, vol. 50, pp. 891-905, Sept. 2001.

[18] W. Qian, X. Li, Marc D. Riedel, K. Bazargan, and D. J. Lilja, “An architecture for fault-tolerant computation with stochastic logic,” *IEEE Trans. on Computers*, vol. 60, no. 1, pp. 93-105, 2011.

[19] Y. Liu and K.K. Parhi, "Lattice FIR Digital Filters using Stochastic Computing," in *Proc. of 2015 IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1027-1031, April 2015

[20] B. Yuan, Y. Wang and Z. Wang, “Area-Efficient Error-Resilient Discrete Fourier Transformation Design using Stochastic Computing,” submitted to *Great Lake Symp. on VLSI (GLSVLSI’2016)*

[21] B. Yuan, C. Zhang and Z. Wang, “Design Space Exploration for Hardware-Efficient Stochastic Computing: A Case Study on Discrete Cosine Transformation”, accepted by IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP’2016)

[22] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu and W. Gross, “VLSI Implementation of Deep Neural Network using Integral Stochastic Computing,”, available in arxiv.org.

[23] P. Li and D. Lilja, “A low power fault-tolerance architecture for the kernel density estimation based image segmentation algorithm,” in *Proc. of IEEE International Conf. on Application-Specific Systems, Architectures and Processors (ASAP)*, pp. 161-168, 2011.

[24] D. Zhang and H. Li, “A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms,” IEEE Trans. Industrial Electronics, vol. 55, no. 2, pp. 551–561, 2008.

[25] M. D. Zeiler and R. Fergus, “Stochastic Pooling for Regularization of Deep Convolutional Neural Networks,” available in arxiv.org

[26] A. Krizhevsky, I. Sutskever and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in Proc. of NIPS, 2012.

[27] D. Larkin, A. Kinane, V. Muresan and N. O’Connor, “An efficient hardware architecture for a neural network activation function generator,” in *Proc. of the ISNN Int. Symp. on Neural Networks*, vol. 144, pp. 1319–1327, 2006.

[28] S. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, Field-Programmable Gate Arrays, Springer, 1992.

[29] A. B. Kahng, B. Li, L. S. Peh, and K. Samadi, “ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration,” Proc. of Design, Automation, and Test in Europe (DATE), 2009.

[30] A. Alaghi and J. P. Hayes, “Survey of stochastic computing,” *ACM Trans. Embed. Comput. Syst*, vol. 12, no. 92, May 2013.

[31] TTD Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas et al. "Theano: A Python framework for fast computation of mathematical expressions." *arXiv preprint arXiv*, 2016.

[32] LeCun, Yann, C. Cortes, "The MNIST database of handwritten digits." , 1998.

[33] LeCun, Yann. "LeNet-5, convolutional neural networks." *URL: http://yann. lecun. com/exdb/lenet*, 2015.

[34] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems*, pp. 1097-1105. 2012.

[35] B. Jun, P. Kocher, “The Intel Random Number Generator.”, *Cryptography Research Inc. White Paper*, 1999.