Dynamic Energy-Accuracy Trade-off Using Stochastic Computing in Deep Neural Networks

Kyounghoon Kim[1], Jungki Kim[1], Joonsang Yu[1], Jungwoo Seo[1], Jongeun Lee[2], and Kiyoung Choi[1]
[1] Dept. of Electrical and Computer Engineering, Seoul National University, Seoul, Korea
{khkim, jungki.kim, joonsang.yu, jungwoo.seo, kchoi}@dal.snu.ac.kr
[2] School of Electrical and Computer Engineering, UNIST, Ulsan, Korea
jlee@unist.ac.kr

ABSTRACT
This paper presents an efficient DNN design with stochastic computing. Observing that directly adopting stochastic computing to DNN has some challenges including random error fluctuation, range limitation, and overhead in accumulation, we address these problems by removing near-zero weights, applying weight-scaling, and integrating the activation function with the accumulator. The approach allows an easy implementation of early decision termination with a fixed hardware design by exploiting the progressive precision characteristics of stochastic computing, which was not easy with existing approaches. Experimental results show that our approach outperforms the conventional binary logic in terms of gate area, latency, and power consumption.

Keywords
Deep Learning; Deep Neural Networks; Stochastic Computing; Energy Efficiency.

1. INTRODUCTION
Deep neural networks (DNNs) dramatically improve the accuracy of machine learning applications such as object detection [2] and speech recognition [3] that need the intelligence of human. However, compared with other machine learning techniques such as support vector machine (SVM), decision tree, and k-nearest neighbor (KNN), DNNs typically require a lot more computations due to many layers and many neurons comprising the network. Moreover, the industrial and academic needs tend to increase the size and complicate the topology of DNNs [4]. Because of this, using high performance computers with accelerators such as GPUs and/or clustering a bunch of machines is regarded as a practical solution to implementing DNNs [5]. Considering, however, that machine learning has also been rapidly adopted in mobile and embedded systems such as self-driving car [6] and patient data analysis [7] with limited resources, researchers have paid great attention to finding possible ways of efficiently executing DNNs including minimizing the required precision [8] and reducing the size of network [9].

In contrast to those studies based on conventional binary arithmetic computing, a different type of computing such as stochastic computing (SC) can be an attractive solution. SC uses the probability of 1s in a random bit stream to represent a number. For example, six 1s in an eight-bit stream in unipolar encoding represent 6/8 as shown in Figure 1 (a). SC can implement a circuit with smaller hardware footprint, lower power, and shorter critical path delay compared with conventional binary logic. It also has advantages in error tolerance and bit-level parallelism. Considering that the majority operations of s DNN is multiplication, SC has great advantage in implementing a DNN because a single AND gate in SC can execute multiplication as shown in Figure 1 (a) with the unipolar encoding having range [0 1].

DAC '16, June 05-09, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897937.2898011

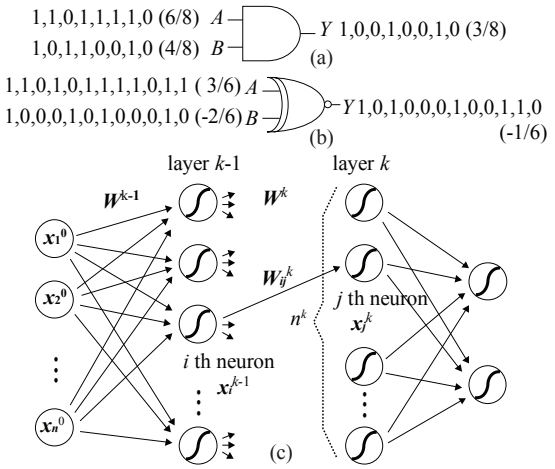


Figure 1. Deep neural network (DNN) using stochastic computing. (a) Stochastic multiplication in unipolar encoding with the range [0 1]. (b) Bipolar stochastic multiplication with [−1 1] range. (c) DNN layers with weight vector W^k in layer k.

However, compared with conventional binary arithmetic, SC has some limitations such as small operational range [−1 1] in bipolar encoding (or [0 1] in unipolar encoding), random error fluctuation, and inefficiency of accumulation. In this paper, we present how we can alleviate the problems that we encounter with when designing a DNN using SC.

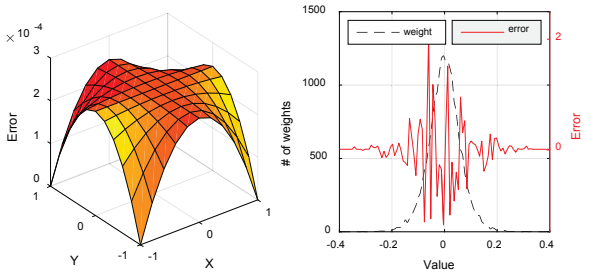
Neural network exploiting SC was first introduced in the noticeable research of [10], where a state-machine based approach was used for implementing an activation function. However, it was not a DNN but a two-layer autoencoder. Although a research for hardware implementation of SC for deep belief network was studied recently [11], only the multiplication part was implemented with SC.

2. BACKGROUND
Because DNN commonly requires negative numbers as well as positive numbers, we use bipolar encoding in this paper. Bipolar stochastic number can be calculated from unipolar number as $p_{bipolar} = 2*p_{unipolar} - 1$. For example, as shown in Figure 1 (b), because nine 1s in a 12-bit stream is 9/12 in unipolar, it is 3/6 in bipolar encoding. An XNOR gate can be used in bipolar encoding to perform multiplication such as $(3/6) \times (-2/6) = (-1/6)$.

A DNN consists of neurons executing multiplication, accumulation, and activation function. Sigmoid or hyperbolic tangent is a typical form of the activation function. In Figure 1 (c), the operation of the j^{th} neuron in layer k (i.e., $\boldsymbol{x}_j^k$) can be defined as follows,

$$\boldsymbol{x}_j^k = af\left(\sum_{i=0}^{n^k-1} \boldsymbol{W}_{ij}^k \boldsymbol{x}_i^{k-1}\right), \quad (1)$$

where $\boldsymbol{W}_{ij}^k$ is a synaptic weight between $\boldsymbol{x}_i^{k-1}$ and $\boldsymbol{x}_j^k$ in layer k; n^k is the number of neurons in layer k; $af(\cdot)$ is the activate function.



(a) Error of XNOR gate (b) Error of near-zero weights

Figure 2. Random error problem occurs when applying SC to DNN. (a) Random error of XNOR gate in 1024-bit stream as absolute value. (b) 20000 weights distribution in 200x100 networks (left Y-axis) and error multiplying by zero (right Y-axis).

2.1 Challenges to Apply SC to DNN

When adopting SC to DNNs, some obstacles should be solved in order to reach the accuracy level achieved by conventional floating-point or fixed point arithmetic. We found that directly applying SC to DNN lead to severe accuracy degradation which is not acceptable in common cases. It happens when synaptic weights are initialized to random numbers with a normal distribution around zero (mean is zero and standard deviation is given by $m^{-1/2}$ where m is the number of inputs to a neuron) as recommended in [12]. In addition, the weights are close to zero due to L1-, L2-regularization which gives penalties to non-zero parameters in order to prevent overfitting [13]. As a result, synaptic weights are aggregated near zero. However, as shown in Figure 2 (a), the XNOR(X, Y) operation representing multiplication in bipolar encoding generates large random errors near the zero values of X and Y. Unfortunately, because many synaptic weights exist near zero, accumulating the products of synaptic weights and inputs increases the error to an acceptable level in our investigation. Figure 2 (b) shows the distribution of 20000 synaptic weights concentrated near zero (dashed line and left Y-axis) and the error of the sum of the products[1] (solid line and right Y-axis). In terms of signal-to-noise ratio (SNR), the results get worse, because the signal values (i.e., the products) tend to be small for near-zero weights.

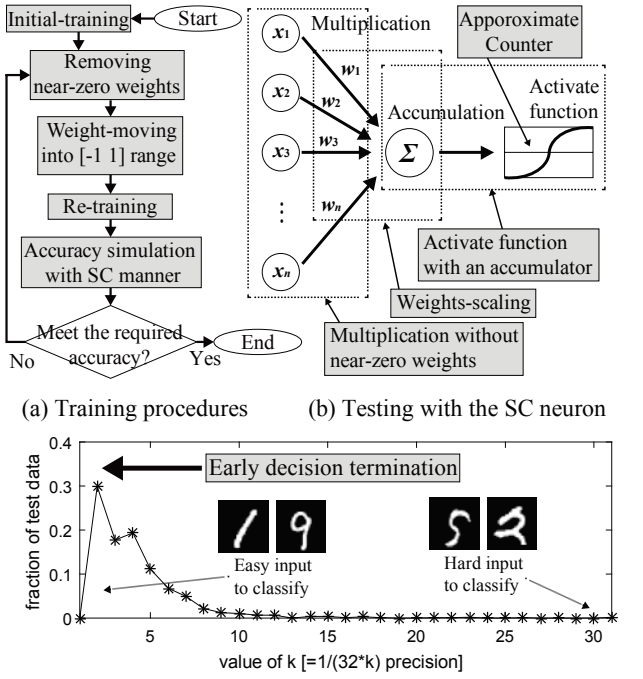
The second problem in applying SC to DNN is the accumulation of many products. In SC, accumulation can be implemented with a multiplexer (MUX) known as scaled addition or an OR gate known as saturated addition. The former sacrifices the precision due to scaling down (or taking average of inputs) while the latter is too sensitive to the correlation of inputs. Another problem is that SC has range limitation from −1 to 1 in bipolar encoding. Thus arithmetic operations in SC can be easily saturated if special care is not taken.

3. DNN USING STOCHASTIC CIRCUIT

3.1 Overview of the Proposed DNN using SC

To use a DNN, we first need to train it (training phase) using training data and then test it with new test inputs (testing phase). We apply SC to testing phase because DNN is mostly operated in testing phase once it is trained in a high-performance system such as a super-computer. Figure 3 shows the proposed training procedure that supports our approach of using SC for DNN. The training performed in a 32-bit floating-point system consists of

[1] To calculate the error of the sum of products at each weight value, the products were obtained by multiplying all the weights for the same value with 0 input. This is to show the tendency of errors appearing at the output of the accumulator.



(a) Training procedures (b) Testing with the SC neuron

(c) Early decision termination

Figure 3. Overview of the proposed procedures and main idea. (a) Training procedure for DNN using SC with 32-bit floating-point computation. (b) SC neurons are operated with SC exploiting the suggested solutions in testing phase. (c) Early decision termination by using progressive precision of SC.

initial-training, removing near-zero weights, incremental weight-moving into [−1 1] range, and re-training as shown in Figure 3 (a); accuracy simulation is conducted in SC after the re-training. The two training steps, initial- and re-training, are identical to the conventional DNN training method using back-propagation algorithm [14]. There are typically many near-zero weights because weights have a tendency to become zero due to regularization as mentioned in Section 2.1. And we find that the technique to remove such near-zero weights is very effective.

In the testing phase, as shown in Figure 3, four methods are applied: 1) multiplications are done without near-zero weights to minimize random errors, 2) weights are scaled to improve signal intensity (i.e., SNR), 3) activation functions are implemented with an accumulator, and 4) approximate counters are used to reduce the size of hardware circuit.

SC can adjust the computation precision without hardware modification. This cannot be realized in other computing systems. For example, an SC system can use 32 and 1024 bits (or any bit-length) for 1/32 and 1/1024 precision, respectively, whereas 10-bit fixed-point system is fixed to 1/1024 precision only. By using this property of SC for the operation at low-precision, we can reduce energy-consumption. Fortunately, in most time, a large fraction of input data can be easily classified because many classification problems are far from decision boundaries [15]. We investigate MNIST dataset for handwritten digit images [16], as shown in Figure 3 (c), where classifying 78.1% of input data needs 1/128 precision while classifying 93.3% needs 1/256 precision. Thus, we suggest early decision termination (EDT) to finish the computation for easily classified inputs in an early stage with low precision (i.e., a relatively small number of bits).

3.2 Removing Near-Zero Weights

Because the existence of near-zero weights is a main source of generating errors when using SC for a DNN as mentioned in

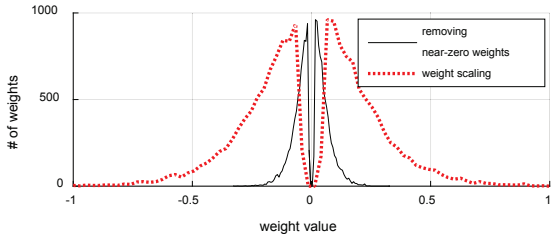


Figure 4. The distribution of weights after removing near-zero weights and weight-scaling.

Section 2.1, we propose to remove near-zero weights in the training phase. In the literature of machine learning, discarding zero weights is recently suggested in order to reduce the size of network [13]. On the other hand, we remove near-zero weights to reduce random fluctuation errors. Re-training is necessarily required because we find that the accuracy after pruning near-zero weights severely decreases and recovers after re-training. We remove near-zero weights under the threshold proportional to standard deviation of weights in a layer as follows,

$$Th_{near-zero} = std(\boldsymbol{W}^k) \cdot (\alpha + \beta \cdot i), \quad (2)$$

where $\boldsymbol{W}^k$ is all synaptic weights in a layer k; α and β are parameters decided empirically; i represents the number of iterations in Figure 3 (a). Thus, as i increases, more near-zero weights are removed. In our case, α and β are set to 0.2 and 0.01, respectively; re-training epoch is ten. The distribution of weights after removing near-zero weights is shown as black solid line in Figure 4.

3.3 Applying Weight Scaling

In order to minimize random fluctuation error and maximize SNR, we propose weight scaling technique. As shown in Figure 2 (a), error increases as the weight becomes close to zero and decreases as it becomes close to 1 or −1 while signals are changed in the opposite way. Thus, in (1), if we scale up the weights $\boldsymbol{W}^k$ before the multiplication and then scale back down the result after accumulation, the SNR can be improved. Suppose that the weights $\boldsymbol{W}^k$ are limited to a range $[-\frac{1}{s}\ \frac{1}{s}]$ where $s > 1$, (1) can be rewritten as follows,

$$\boldsymbol{x}_j^k = af\left(\frac{1}{s} \cdot \sum_{i=0}^{n^{k-1}} s \cdot \boldsymbol{W}_{ij}^k \boldsymbol{x}_i^{k-1}\right) \quad (3)$$

Because $s \cdot \boldsymbol{W}^k > \boldsymbol{W}^k$, the signal level increases whereas the error decreases. For example, if $\boldsymbol{W}^k$ are (0.10, −0.15, 0.20), i.e., limited to [−0.2 0.2], then the weights can be scaled up five times (s=5) to become (0.50, −0.75, 1.00). The red dotted line in Figure 4 shows the advantages of the proposed weight-scaling technique. Note that the number of near-zero weights also decreases because they become far from zero center. The overhead of scaling is negligible because the scaling operation can be applied to synaptic weights in binary format only once after finishing the training phase.

The problem of this method is that it needs a scaled activation function as follows,

$$\boldsymbol{x}_j^k = af\left(\frac{1}{s}t\right), \text{ where } t = \sum_{i=0}^{n^{k-1}} s \cdot \boldsymbol{W}_{ij}^k \boldsymbol{x}_i^{k-1}. \quad (4)$$

In the next section, we suggest a scaled active function.

3.4 Activation Function with Accumulation

We present a state-machine based hyperbolic tangent activation function (i.e., $tanh(\cdot)$) to solve the accumulation problem mentioned in Section 2.1 and provide the scaled functionality $tanh\left(\frac{1}{s}t\right)$ for weight-scaling. State-machine based hyperbolic tangent was introduced in [17] and [18] for a single bit stream and multiple bit streams, respectively. The hyperbolic tangent

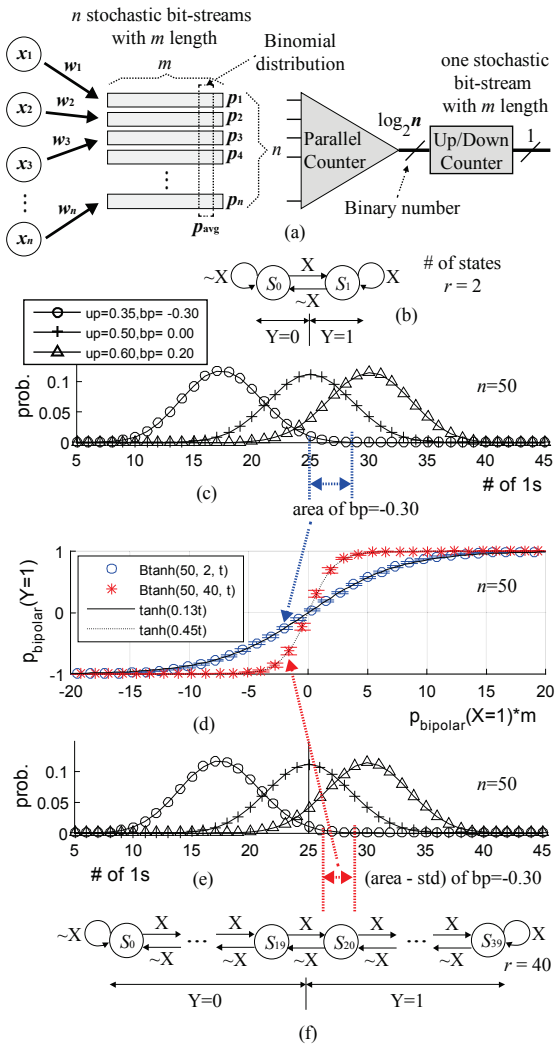
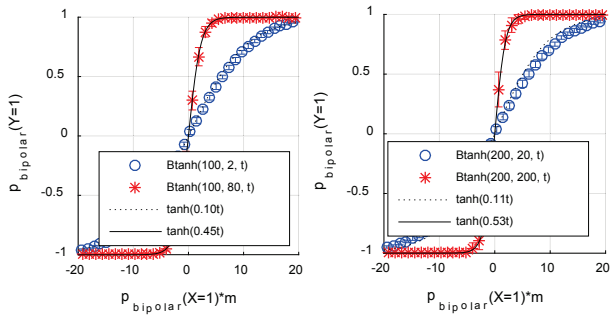


Figure 5. A stochastic neuron and the mechanism of state-machine based activate function. (a) A single neuron using SC. (b) state-machine having two states in an up/down counter. (c) Using binomial distribution for the logistic function. (d) The proposed activate function. (e) Binomial distribution with many states. (f) state-machine having 40 states.

activation functions proposed in the previous work only support $tanh(vt)$, where $v \geq 2$ and a natural number; whereas, our $tanh\left(\frac{1}{s}t\right)$ supports a small coefficient (i.e., $1/s < 1$) as well as multiple bit streams. As shown in Figure 5 (a), given n bit-streams with m bit length generated by the n multiplications of inputs (x_i's) and weights (w_i's), we count the number of 1s in each column by using a parallel counter. The counted value is used by the following saturated up/down counter as the amount of increase or decrease. The resulting binary value of the up/down counter is regarded as its state. Given r states in the up/down counter, one half of the r states generates 0 bit while the other half generates 1 bit; the generated bits approximate outputs of $tanh(\cdot)$ in the form of stochastic number.

3.4.1 Proposed Stochastic Hyperbolic Tangent

Given n input bit-streams with average probability p_{avg} for any bit to be 1, the probability P_{one} of having b 1s in a column of the input bit-streams becomes binomial distribution as follows,



(a) (b)

Figure 6. The result comparison between the proposed hyperbolic tangent *Btanh*(·) and the original *tanh*(·). (a) The number of states is two and 80 for 100 bit-streams. (b) 20 and 200 states for 200 bit-streams.

$$P_{one} = \binom{n}{b}(p_{avg})^{b}(1-p_{avg})^{n-b}. \quad (5)$$

Because cumulative distribution function (CDF) of binomial distribution mimics the logistic function, we construct *tanh*(·) by using CDF of (5). In case of two state up/down counter shown in Figure 5 (b), CDF of binomial distribution shown in Figure 5 (c) for p_{avg} = 0.00 follows *tanh*(0.13t) shown in Figure 5 (d). Note that state S_1 and S_0 in Figure 5 (b) generates 1s and 0s, respectively. As the number of states increases as shown in (f), it becomes a bounded random walk problem [19] and the results are affected by the variation of walking. From this property, we find the relationship between $tanh\left(\frac{1}{s}t\right)$ and the proposed *Btanh*(*n*, *r*, *t*) as follows,

$$Btanh(n,r,t) \cong tanh\left(\frac{1}{s}t\right), \quad (6)$$

$$\frac{1}{s} = \frac{1-q}{2(n-1)}(r'-2n)+1, \text{ and } q = 1.835(2n)^{-0.5552} \quad (7)$$

$$r' = \frac{2(1-s)(n-1)}{s(1-q)} + 2n \quad (8)$$

$$r = nearest\_multiple\_of\_two(r') \quad (9)$$

where *n* is the number of bit-streams and *r* is the number of states and multiple of two. Figure 6 shows that both the proposed *Btanh*(·) and the corresponding *tanh*(·) are almost identical to each other. Figure 7 shows the algorithm for *Btanh*(·). In Line 1, the maximum state is set to *r*−1; the state index starts from 0 and the number of states is *r*. The offset *V* jumping between states is calculated as bipolar encoding in Line 5; it is zero if the number of 1s in a column is a half of *n*. Note that the SC simulation step uses *Btanh*(*n*, *r*, *t*) representing $tanh\left(\frac{1}{s}t\right)$ while intial- and re-training steps in Figure 3 (a) use $tanh(a \cdot x)$, where $t = s \cdot a \cdot x$; *s* and *a* are real numbers.

3.4.2 Using Approximate Counter

In the stochastic neuron shown in Figure 5 (a), the biggest component in terms of area and power is the parallel counter. Since SC calculates in an inaccurate manner anyway, we do not need to stick to a conventional accurate parallel. [20] presents an approximate parallel counter with very small error and no bias. We reduce the area and power by using the same approach.

4. EARLY DECISION TERMINATION

Because the bits fed in at different times are independent of each other in stochastic bit-streams, the precision can be adjusted without hardware modification; it is known as progressive precision [21]. It is a salient advantage of SC over conventional logic using binary arithmetic. As mentioned in Section 3.1, early decision termination (EDT) is useful in terms of energy consumption and decision speed. Our current implementation has

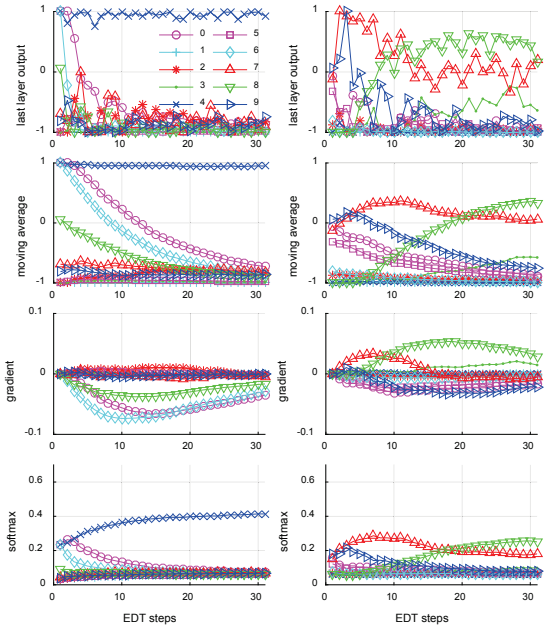
BTANH (*n*, *r*, *t*)

// *n* is the number of bit-stream as shown in Figure 5 (a).
// *m* is the length of bit-stream.
// *r* is the number of states in up/down counter.
// *t* is the bipolar SC input, *n* x *m* size.
// t_i is the *i*th column of *t*, $t = [t_1, \ldots, t_m]$.

Output: Y_i is the *i*th stochastic output bit for *Btanh*(·)

1: $S_{max} = r - 1$ //max state
2: $S_{half} = r/2$ //half state
3: $S \leftarrow S_{half}$ //current state
4: **for** $i = 1$ **to** m
5: $V = \text{Count}(t_i) * 2 - n$ //bipolar 1s counting
6: $S \leftarrow S + V$
7: **if** $S > S_{max}$ **then** $S \leftarrow S_{max}$
8: **if** $S < 0$ **then** $S \leftarrow 0$
9: **if** $S > S_{half}$
10: $Y_i \leftarrow 1$
11: **else**
12: $Y_i \leftarrow 0$

Figure 7. Pseudo-code for the proposed *Btanh*(·).



(a) (b)

Figure 8. The intermediate procedures of early decision termination. Ground truths are 4 and 8, respectively.

32 bits as the precision granularity. That is, it processes 32 bits to make a decision, and if it fails, it continues processing the next 32 bits.

4.1 Moving Average Tracking Output Trends

As shown in the first row of Figure 8[2], we find that the outputs of the last layer from SC circuit severely fluctuate as EDT steps proceed, where every EDT step processes 32 stochastic bits. In order to monitor the trend of decisions as a time series, the following moving average is used as a low-pass filter,

$$MV_{c,i} = \alpha \cdot Y_{c,i} + (1-\alpha) \cdot MV_{c,(i-1)} \quad (10)$$

[2] The example uses MNIST handwritten image dataset, where the number of classes is ten from zero to nine.

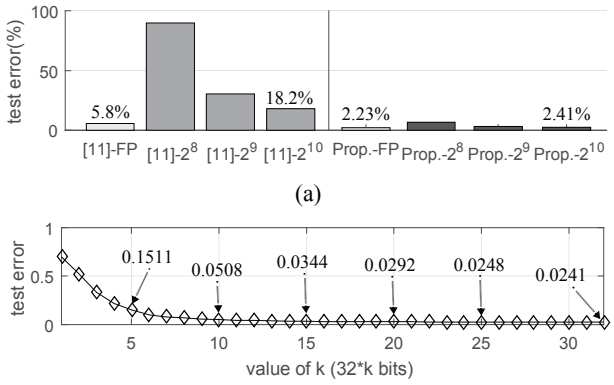


Figure 9. Comparison misclassification error. MNIST test data error in 32-bit floating point is 2.23%. The proposed method is 2.41% while the previous work [11] is 18.2% in 2^{10}-bit stream.

where c is the class; $Y_{c,i}$ is the output of the last layer for class c in EDT step i and $MV_{c,1}=Y_{c,1}$ and α is empirically set to 0.1. As shown in the second row of Figure 8, the result of moving average better shows the trend. We find two important components for EDT: 1) output value of the last layer and 2) the gradient of current step. Using the former is natural because the largest output value is selected in general classification domain. The latter can be an indicator to notify the possibility of changing the current decision in the future. For example, class 7 and 8 are swapped in the moving average of Figure 8 (b) (the second row), which can be informed early by examining the gradient (the third row). Thus, the objective value of individual class c in ith EDT step is defined as follows,

$$O_{c,i} = MV_{c,i} + \beta \cdot Grad_{c,i} \quad (11)$$

where $Grad_{c,i}$ is the gradient value of class c in ith step; β is a weight factor. Finally, by using softmax functions, the objective values of individual classes are normalized depending on the categorical probability, which is commonly used in multiclass classification problems.

$$SM_{c,i} = \frac{e^{O_{c,i}}}{\sum_k e^{O_{k,i}}} \quad (12)$$

As shown in the fourth row of Figure 8, the normalized value represents the current status of a class candidate with absolute value. For example, because class 4 dominates other classes in case of Figure 8 (a), it can be selected as the final decision in early EDT step.

5. EXPERIMENTAL RESULTS

For the experiment, we use MNIST handwritten digit image dataset [16] consisting of 60000 training data and 10000 testing data with 28x28 grayscale image and 10 classes. The networks in this experiments have two hidden layers with a 784-100-200-10 configuration.

5.1 Accuracy of DNN Using SC

The accuracy of DNN is a very important metric, because usability of DNN totally depends on it. We compare our proposed method with the previous methods using SC [11], and a 32-bit floating-point system. As shown in Figure 9 (a), misclassification error of test data in MNIST dataset in 32-bits floating-point is 2.23%, while test error for our proposed DNN using SC is 2.41% with 2^{10}-bit stream. Considering that the previous work [11] using SC and floating-point are 18.2% and 5.8%, respectively, Our work dramatically improves in terms of accuracy. Because fixed-point arithmetic generally has bigger error compared with floating-point arithmetic and the proposed approach is almost similar to floating-

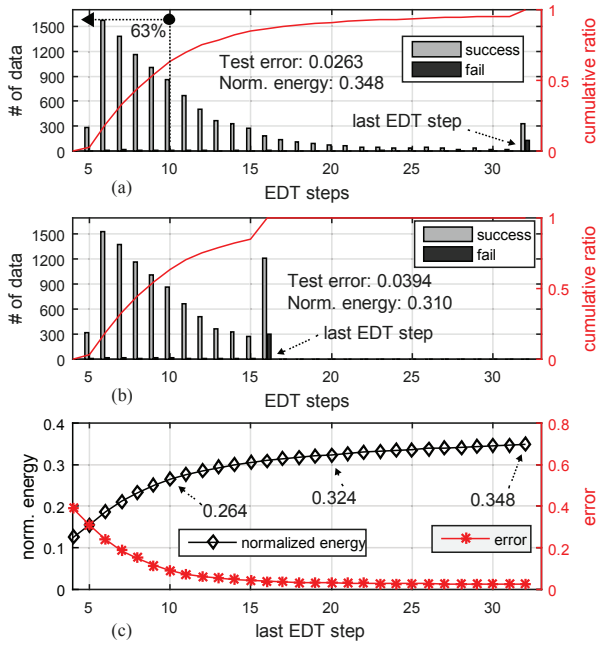


Figure 10. Experimental result for early decision termination (EDT) where one EDT step use 32 stochastic bits. (a) Applying EDT to 1024 bits. (b) The last step of EDT is set to the 16th step (i.e., 16x32=512 bits). (c) Normalized energy reduction between using and not using EDT and test error according to the last EDP step.

point result, we conclude that the improvement is significant. Figure 9 (b) shows test error in the proposed method when using progressive precision mentioned in Section 4, where each step uses 32-bit parallel stochastic circuit. Note that the difference of test error between the 15th step and the 32th step is about 1% error, which means that reducing energy as well as improving decision speed is possible with sacrificing only 1% error rate.

5.2 Effectiveness of Early Decision Termination

Early decision termination (EDT) exploits progressive precision of SC, which can adjust the required precision without hardware modification. In our experiment for EDT, the baseline SC circuit uses 1024 stochastic bit-stream; since one EDT step uses 32 bits, 32 EDT steps are identical to the baseline SC circuit (i.e., 32x32=1024). EDT can reduce energy and improve decision speed. Figure 10 (a) shows that the tests are finished earlier compared to the baseline SC circuit. For example, 63% among 10000 tests are finished before the 10th step (i.e., 320 bits); compared with the baseline SC with 2.41% error, EDT decreases energy by 65.2% with 2.63% error as shown in (a). If we move the last EDT step to earlier ones, we can save more energy as shown in (b). By setting the last EDT step to 16, energy decreases by 69% while sacrificing accuracy by 1.53%, compared with the baseline SC circuit. (c) shows this trade-off relationship between the last EDT step, test error, and normalized energy reduction using EDT compared with not using EDT.

5.3 Comparison of Synthesis Results

We synthesize one SC neuron with 200 inputs, which is compared with 9-bit fixed point (FIX) because SC circuit using 512(=2^9) bits shows reasonable test error rate 0.031. Stochastic bits for each synaptic weight are generated with stochastic number generators (SNG) proposed in [22], where linear feedback shift registers (LFSRs) are shared among parallel SC circuits without generating correlation. They are implemented as combinational

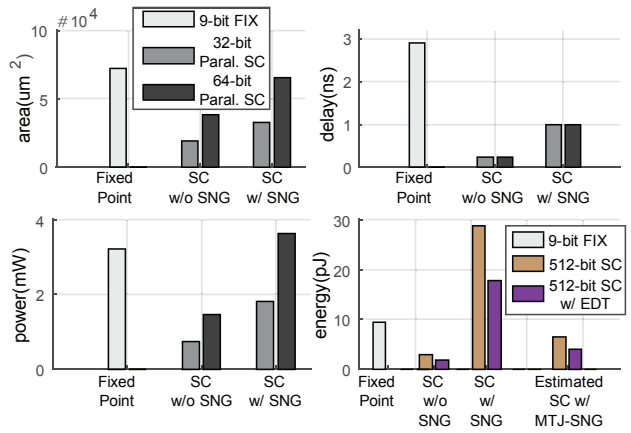


Figure 11. Synthesis results. All cases are compared with 9-bit fixed-point (9-bit FIX). In case of area, critical path delay, and power, 32- and 64-bit parallel SC circuits are used. In case of energy, SC circuit executes 2^9(=512) bits.

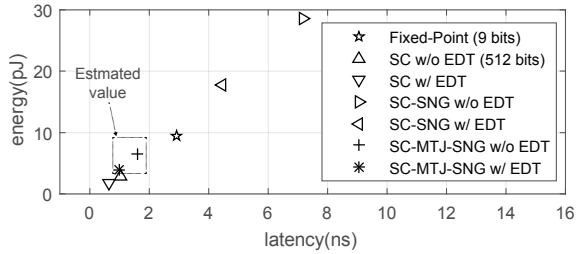


Figure 12. Iso-area performance comparison. Energy and latency for each case are compared under same area. The values for SC circuits using MTJ-SNG are estimated according to [1]. Fixed-point computes with 9-bit width while all SC circuits compute 2^9(=512) bits.

logic in TSMC 45nm technology library with Synopsys Design Compiler using Verilog HDL. The fixed-point is implemented with 3-stage pipelines. The recent work [1] reports that spintronic SNG using magnetic tunneling junctions (MTJs) improves energy efficiency about seven times compared with CMOS SNG. Thus, we also add the estimated value for MTJ-SNG.

Figure 11 shows the synthesis results in terms of area, critical path delay, power, and energy, where the delay of the fixed-point circuit is multiplied by three due to 3-stage pipeline. SC circuit can be implemented with a serial unit up to 512 parallel units, and we select 32- and 64- parallel SC circuit for area, critical path delay and power investigation. Note that the area and power increase in proportion to the parallelism while critical path delay does not vary. As a result, we find that SNG overhead is very significant; it takes 41.50%, 59.58%, and 75.76% of SC circuit (SC w/ SNG) in terms of area, power, and critical path delay. However, regardless of the parallelism of SC circuit, energy consumption is identical in all cases. Compared with 9-bit width fixed-point, SC with SNG increases energy by 3.0 times while SC without SNG decreases energy by 70.0%; we also estimate SNG with MTJ-SNG decreasing energy by 30.0% from the result of [1]. Due to EDT, energy decrease by about 34% compared to basic SC in all cases. Figure 12 shows iso-area performance where all circuits are set to the area of 9-bit fixed-point which is 72104 um^2. In case of latency with EDT[3], SC without SNG is 4.61 times faster while SC with SNG is 1.53 times slower compared with 9-bit fixed-point. It is because the two cases have 120x and 70x parallelisms, respectively, under iso-area and the critical path delay of the former is 4.125 shorter than that of the latter.

[3] The latency is the time to calculate a 9-bit binary number in a fixed-point arithmetic and to calculate a 512-bit stream in SC.

6. CONCLUSION

In this paper, we address the problems in directly adopting stochastic computing to DNN by removing near-zero weights, applying weight-scaling, and using state machine based activation function integrated with the accumulator. We also suggest the early decision termination which is very useful in terms of energy and decision speed. The experimental results demonstrate that the accuracy of DNN using SC is close to that of the conventional floating-point system while reducing the area, power, critical path delay, and energy.

7. ACKNOWLEDGEMENTS

This work was supported by Samsung Research Funding Center of Samsung Electronics under Project Number SRFC-IT1501-08.

8. REFERENCES

[1] R. Venkatesan, *et al.*, "Spintastic: spin-based stochastic logic for energy-efficient computing," Proc. DATE, pp. 1575-1578, 2015.
[2] A. Krizhevsky, *et al.*, "Imagenet classification with deep convolutional neural networks," Proc. NIPS, pp. 1097-1105, 2012.
[3] 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, pp. 82-97, 2012.
[4] C. Szegedy, *et al.*, "Going deeper with convolutions," arXiv preprint arXiv:1409.4842, 2014.
[5] A. Coates, *et al.*, "Deep learning with COTS HPC systems," Proc. ICML, pp. 1337-1345, 2013.
[6] M. Montemerlo, *et al.*, "Junior: The stanford entry in the urban challenge," J. Field Robot., vol. 25, pp. 569-597, 2008.
[7] K. H. Lee, *et al.*, "A low-power processor with configurable embedded machine-learning accelerators for high-order and adaptive analysis of medical-sensor signals," IEEE J. Solid-State Circuit, vol. 48, pp. 1625-1637, 2013.
[8] M. Courbariaux, *et al.*, "Training deep neural networks with low precision multiplications," Proc. workshop contribution at ICLR, 2015.
[9] H. Geoffrey, *et al.*, "Distilling the knowledge in a neural network," Proc. NIPS workshop, 2014.
[10] B. D. Brown, *et al.*, "Stochastic neural computation. II. Soft competitive learning," IEEE Trans. Comput., vol. 50, pp. 906-920, 2001.
[11] K. Sanni, *et al.*, "FPGA implementation of a Deep Belief Network architecture for character recognition using stochastic computation," Proc. CISS, pp. 1-5, 2015.
[12] Y. A. LeCun, *et al.*, "Efficient backprop," in *Neural networks: Tricks of the trade*, ed: Springer, 2012, pp. 9-48.
[13] S. Han, *et al.*, "Learning both weights and connections for efficient neural networks," Proc. NIPS, 2015.
[14] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural Networks, vol. 61, pp. 85-117, 2015.
[15] S. Venkataramani, *et al.*, "Scalable-effort classifiers for energy-efficient machine learning," Proc. DAC, p. 67, 2015.
[16] L. Deng, "The MNIST database of handwritten digit images for machine learning research," IEEE Signal Process. Mag., vol. 29, pp. 141-142, 2012.
[17] B. D. Brown, *et al.*, "Stochastic neural computation. I. Computational elements," IEEE Trans. Comput., vol. 50, pp. 891-905, 2001.
[18] A. Ardakani, *et al.*, "VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing," arXiv preprint arXiv:1509.08972, 2015.
[19] O. C. Ibe, *Elements of Random Walk and Diffusion Processes*: John Wiley & Sons, 2013.
[20] K. Kim, *et al.*, "Approximate De-randomizer for Stochastic Circuits," Proc. ISOCC, 2015.
[21] A. Alaghi, *et al.*, "Survey of stochastic computing," ACM Trans. Embed. Comput. Syst., vol. 12, p. 92, 2013.
[22] K. Kim, *et al.*, "An Energy-Efficient Random Number Generator for Stochastic Circuits," Proc. ASP-DAC, 2016.