Sign-Magnitude Stochastic Computing

Aidyn Zhakatayev

Computer Science & Engineering
Graduate school of UNIST

Sign-Magnitude Stochastic Computing

A thesis
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Aidyn Zhakatayev

06.14.2018

Approved by

이종
은인

Major Advisor
Jongeun Lee

Sign-Magnitude Stochastic Computing

Aidyn Zhakatayev

This certifies that the thesis of Aidyn Zhakatayev is approved.

06.14.2018

이종
은인

Thesis Supervisor: Jongeun Lee

[illegible]

Woongki Baek: Thesis Committee Member #1

[illegible]

Seokhyeong Kang: Thesis Committee Member #2

Abstract

Stochastic computing (SC) is a promising computing paradigm for applications with low precision requirement, stringent cost and power restriction. One known problem with SC, however, is the low accuracy especially with multiplication. In this work we propose a simple, yet very effective solution to the low-accuracy SC-multiplication problem, which is critical in many applications such as deep neural networks (DNNs). Unlike previous solutions, our method is scalable, flexible, and does not require *a priori* knowledge of input values. It is based on an old concept of *sign-magnitude*, which, when applied to SC, has unique advantages. Our experimental results using multiple DNN applications demonstrate that our technique can improve the efficiency of SC-based DNNs by about 32X in terms of latency over using bipolar SC, with very little area overhead (about 1%). Moreover, we analyze the existing SC encodings and propose mathematical formulation of SC multiplication, which can be used for analysis and to model the behavior of SC multiplication, instead of performing bit-level simulation.

Contents

List of Figures

List of Tables

LIST OF TABLES

CHAPTER I

INTRODUCTION

Stochastic computing (SC), in which numbers are represented using the probability of ones in a bitstream, has several advantages over both conventional binary implementations (e.g., lower cost and lower power consumption as well as inherent fault tolerance) and analog designs (e.g., higher integration density and lower variability). As such, stochastic computing has been applied to many applications including deep neural networks (DNNs) [3, 7, 9–11, 13–15, 17, 20], where the large number of arithmetic operations makes SC very attractive. Also, the fact that not all DNN applications require exact result, and that in SC one can dynamically adjust arithmetic precision without any hardware overhead [2, 10, 21] can help bolster the case for SC-based DNNs. Further, with SC it may be possible to fit an entire DNN in one chip without needing an external memory for intermediate results due to small footprint of SC-based neurons, leading to better energy efficiency compared with von-Neumann-style central memory model.

However, today's SC-DNNs suffer from a low accuracy problem, especially when it comes to SC multiplication [10]. Authors of [10], noticing that the inaccuracy is maximal when the multiplication result is zero, suggest skipping multiplication when one of the operands is near zero. However this solution works only if one knows the value of the operands, which is the case only if the SC-DNN design dedicates hardware to each neuron, and even in that case, only for the weight parameter operand, since the other operand, i.e., input activation, is data dependent and cannot be known *a priori*. In other words the scheme works only if the design is fully-parallel, which would not be scalable for large DNNs.

In this work we propose an alternative, much improved method, which consists of a new encoding scheme for stochastic numbers (which we call *sign-magnitude SC*, or *SM-SC* for short) and a set of SC computing elements for it. Our sign-magnitude SC, though simple in concept, can vastly improve the accuracy of SC in SC-multiplication and derived functionalities such as SC-MAC (Stochastic Computing Multiply-and-Accumulator) by 4X $\sim$ 9.5X depending on the input data range, at the cost of negligible hardware overhead. This in turn enables great improvements in recognition accuracy and efficiency of SC-DNNs, allowing otherwise the same SC-DNN architecture to reach the level of accuracy that has never been achieved by previous SC-DNNs employing the same architecture.

The basic idea of our proposal is to handle sign and magnitude of an SC number separately. This forfeits the elegance of the bipolar encoding, which is used as the *de facto* standard in SC designs dealing with signed numbers. It also goes against the idea of SC, which is to make hardware simple even by sacrificing some accuracy. However our work shows that the overhead of sign-magnitude can be made very small in a certain class of applications and the benefit can be quite dramatic.

This work makes the following contributions. First, we present a mathematical formulation of error in unipolar and bipolar SC multiplication. Second we present an in-depth analysis and compare unipolar vs. bipolar encoding in SC in terms of accuracy and efficiency. Third we propose a new representation for stochastic numbers (SM-SC) along with a set of operators including multiplier and adder for SM-SC. Though the format itself closely resembles that of sign-magnitude in *conventional binary arithmetic*, SM-SC has a number of unexpected advantages over conventional SC that are not found in the original sign-magnitude format. Fourth we apply our SM-SC to various SC-DNN accelerator architectures, including CNN (convolution neural network) and RNN (recurrent neural network), and demonstrate its effectiveness.

Our experimental results show that not only can our SM-SC improve the accuracy of SC-MAC operations significantly compared with using bipolar-SC, but also the SC-DNNs based on our SM-SC can be much more efficient, by about 32X (from 1024-bit to 32-bit) in terms of bitstream length over the bipolar-based SC-DNNs, with very little area overhead (about 1% in the 64-bit parallel case). Our technique is applicable to various DNNs, including CNNs and RNNs, and has similar fault tolerance as bipolar SC.

CHAPTER II

Analysis of SC multiplication

One way to explain stochastic computing is that the frequency (or probability) of ones in a bitstream matches the value represented by the bitstream. Thus a N-bit bitstream, which can have $N+1$ different values in SC, can be used to represent a value in $\{i/N \,|\, 0 \leq i \leq N\}$. This is called *unipolar encoding*, and works for unsigned data only. For signed data, one may use *bipolar encoding*, which is to use the encoding(s) for unipolar value u to represent bipolar value $2u-1$.

The power of SC is in being able to perform bit-level simple computations. For example, multiplication can be performed with a single AND gate. Suppose $a = P(A=1)$ and $b = P(B = 1)$ are inputs of AND gate and $c = P(C=1)$ is the output. For each output bit to be 1 both input bits must be one, which is $P(C=1) = P(A=1 \textit{ and } B=1)$. If inputs are independent from each other $P(C=1) = P(A=1) \cdot P(B=1)$. In other words, $c = a \cdot b$ and multiplication of two numbers can be performed using AND in stochastic domain. In bipolar encoding XNOR gate can be used for SC multiplication.

The accuracy of SC result has two aspects: mean and standard deviation, but the mean usually matches the correct result unless there is some systematic error (or bias) in the SC algorithm. On the other hand, standard deviation is about how much error or deviation in the result one will see on a single trial, and therefore will be nonzero unless the SC result is guaranteed to match the correct result. The longer the bitstream is, the lower the standard deviation. Thus we use the standard deviation of SC result as the metric for SC accuracy.

2.1 Mathematical formulation

The behavior of original stochastic computing has been mathematically formulated in [6], where stochastic inputs are assumed to be Bernoulli sequence. However, stochastic computing applications use pseudo-random bitstream for SC, which is far different than original stochastic computing. Stochastic number generators (SNG) generate $\lfloor N * a \rfloor$ number of 1s, where N is the bitstream length and a is encoding value.

[21] proposed efficient SNG which uses shuffling circuit to generate SC bitstream. By improving the quality of random shuffling they achieved similar result as software randomization. Thus, in proposed mathematical formulation is based on randomly shuffled SC bitstreams.

We derived mathematical formulation of variance for both unipolar (which is the same as SM-SC except for the sign bit) and bipolar SC multiplications. Here N denotes the length of stochastic bitstream, p_0 the output of SC multiplication, and a and b the number of 1s of two input SC numbers.

$$\sigma^2(p_0) = \frac{\sum_i \left[\left(\frac{i}{N} - p_1 \cdot p_2\right)^2 \cdot \binom{a}{i}\binom{N-a}{b-i} \right]}{\binom{N}{b}}, \tag{II.1}$$

is the variance of unipolar SC multiplication, where $p_1 = \frac{a}{N}$, $p_2 = \frac{b}{N}$, and i ranges from $\max(0, a+b-N)$ to $\min(a,b)$.

$$\sigma^2(p_0) = \frac{\sum_i \left[\left(\frac{2i-N}{N} - p_1 \cdot p_2\right)^2 \cdot \binom{a}{y}\binom{N-a}{b-y} \right]}{\binom{N}{b}}, \tag{II.2}$$

is the variance of bipolar SC multiplication, where $y = \frac{i+a+b-N}{2}$, $p_1 = \frac{(2a-N)}{N}$, $p_2 = \frac{(2b-N)}{N}$, and i increases by 2 $(i \leftarrow i+2)$ in range from $|a+b-N|$ to $N - |a-b|$.

The idea of derivation is to consider all possible outcomes of SC multiplication. i is the number of 1s generatable from two input SC number. For example, in equation II.1 if $a+b-N < 0$, then the minimum number of 1s generatable is $a+b-N$. In other words, there are too many 1s in input bitstreams and they can't avoid to meet each other. Maximum number of 1s generatable is $min(a,b)$, because in order to generate 1 both inputs must be 1 (AND gate). For each i we calculate all possible cases to generate i number of 1s. To make it simpler, we fix input bitsream a and change order of input bitstream b. $\binom{a}{i}\binom{N-a}{b-i}$ is all possible orderings of b to generate i number of 1s. $\left(\frac{i}{N} - p_1 \cdot p_2\right)^2$ is squared error from the definition of variance. $\binom{N}{b}$ is total number of possible orderings. Basically, the equation considers all possibles orderings and their squared errors and averages them by dividing to the total number of possible orderings. Similar approach is used for bipolar SC multiplication in equation II.2.

Proposed equations can be used in the analyzing accuracy and performance of SC multiplication, instead of building bit-level simulation, which can be extremely slow for large applications. For example, one can use proposed equation to find the length of SC bitstream to achieve certain accuracy. The validation results of proposed equations are presented in Chapter V.

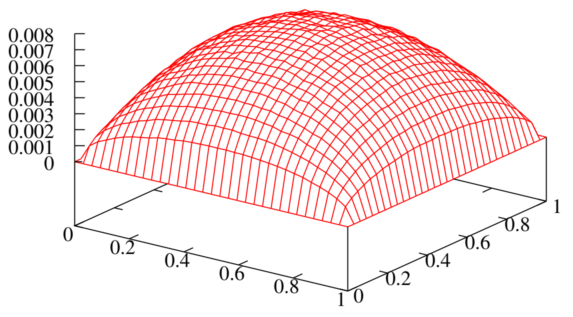
2.2 Comparison of unipolar and bipolar SC multiplication

Figure 2.1 compares the standard deviation of SC multiplication, one of the most frequently used operations in SC applications, using bipolar vs. unipolar for different values of input operands. As pointed out previously [10] and confirmed in Figure 2.1b, the absolute error of bipolar SC-multiplier is highest on median input values. And this is exactly the same for unipolar as well. However, what is different between unipolar and bipolar is the trend of the *relative error*, which we define as the standard deviation divided by the correct SC result, as shown in the lower graphs. For these graphs we use the same input range ([0, 1]) and ensure that the number of data points is also the same between unipolar and bipolar, so that the two graphs can be directly compared. The data points where the expected SC result is zero are omitted.

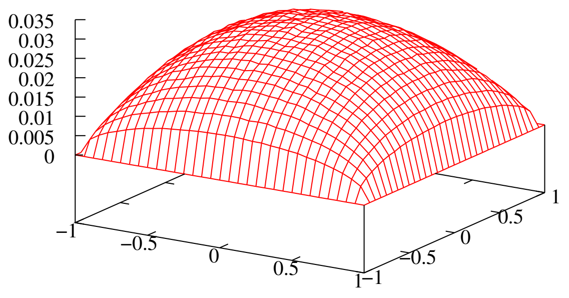
There are two salient points about the graphs. First is the different impact of the maximum absolute error. For bipolar the maximum absolute error happens when the expected SC result is also the lowest (which is at $(0, 0)$), amplifying the impact of error. For unipolar, by contrast, the maximum (absolute) error happens (at $(0.5, 0.5)$) when the expected SC result is a somewhat median value, dampening the impact.

Second difference is the magnitude of the maximum relative error. In both cases the maximum relative error is reported when at least one of the operands is near zero. But whereas in bipolar the maximum relative error happens when both operands are near zero, in unipolar it happens when one operand is near zero and the other is 0.5 (see Figure 2.1a). If both operands are near zero, the absolute error converges to zero, putting a limit on the magnitude of relative error. In consequence, the maximum relative error for unipolar is much lower than for bipolar, as shown in the graphs.

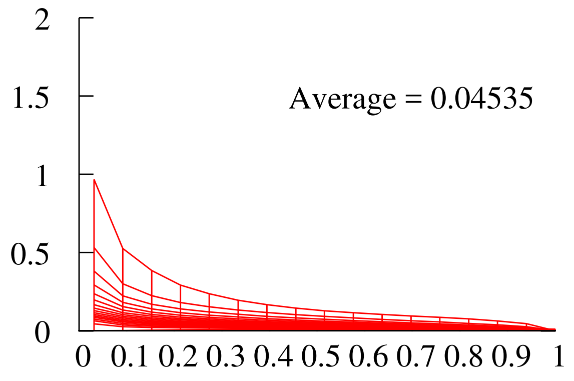
These two factors can explain why the *average* of the relative errors across input combinations (the area under the graph divided by the number of data points) is 6.36X less in the unipolar case compared with the bipolar case. This also unequivocally shows that unipolar encoding is far more accurate than bipolar as far as SC multiplication is concerned.



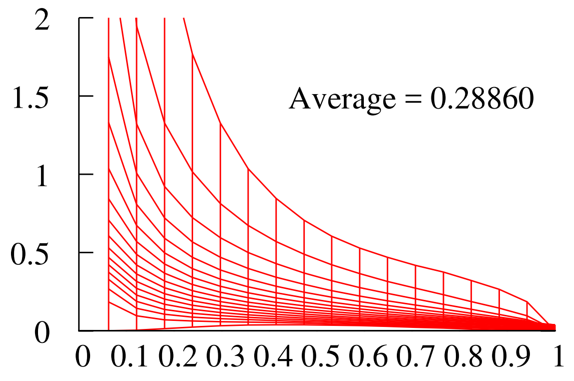
(a) Absolute error, unipolar



(b) Absolute error, bipolar



(c) Relative error, unipolar



(d) Relative error, bipolar

Figure 2.1: SC multiplication error (as measured in standard deviation) vs. input value. Bit-stream length is 1024 bits. In the bottom figures, the x-axis represents the value of one operand while the value of the other operand is represented by different line graphs (Note: different y-axis scale).

CHAPTER III

Proposed SC representation

3.1 Sign-Magnitude SC

Since unipolar encoding cannot handle signed values directly, we propose sign-magnitude SC (SM-SC). To be able to represent negative values SM encoding sacrifices one bit to store sign bit and dedicates $N-1$ bits to store the magnitude. Magnitude part of SM-SC uses same encoding as unipolar SC, which is probability of 1s in the bitstream equals to the value of SC number ($y = P(Y)$). By using sign-magnitude SC (SM-SC) we solve sign problem and take advantage of unipolar encoding accuracy. Table 3.1 shows examples and differences of unipolar, bipolar and sign-magnitude representations, where bitstream length is 6. $P(Y)$ is the probability of each bit to be 1, equals to the number of 1s over total number of bits in the bitstream. In SM-SC negative zero is defined to be the same as positive zero.

Table 3.1: SC encoding examples (N=6)

	Unipolar		Bipolar		SM	
Value	$y = P(Y)$	#1s	$y = 2P(Y) - 1$	#1s	(sign, $y = P(Y)$)	#1s
-1	N.A.	N.A.	000000	0	(1, 11111)	6
0	000000	0	010101	3	(0, 00000)	0
1	111111	6	111111	6	(0, 11111)	5

Table 3.2: Examples of representable numbers (N=4)

Unipolar	0000 = 0	0010 = 0.25	1100 = 0.5	1011 = 0.75	1111 = 1
Bipolar	0000 = -1	0100 = -0.5	1010 = 0	1110 = 0.5	1111 = 1
Sign-magnitude	0,000 = 0 1,101 = -0.66	0,001 = 0.33 1,111 = -1	0,110 = 0.66	0,111 = 1	1,010 = -0.33

3.2 Encoding Efficiency

Let $L_U(n)$ be the number of different values that can be represented with n-bit unipolar encoding. $L_B(n)$ and $L_{SM}(n)$ are defined similarly for bipolar and SM encodings, respectively. Given an N-bit bitstream, it is easy to see the following equalities hold.

$$\begin{aligned} L_U(N) &= N+1 && \text{(III.1)} \\ L_B(N) &= N+1 && \text{(III.2)} \\ L_{SM}(N) &= 2 \cdot L_U(N-1) - 1 = 2N - 1 && \text{(III.3)} \end{aligned}$$

The "minus one" in (III.3) is due to the duplicate zeros. Equation III.3 suggests that the SM format in SC has roughly twice the encoding efficiency as that of the conventional SC. Note that this encoding efficiency advantage is unique in SM-SC only, and the sign-magnitude format in *conventional binary arithmetic* does not have any encoding advantage over the 2's-complement format. Table 3.2 shows an example of representable numbers using 4-bit SC number.

3.3 Computation with Sign-Magnitude SC

Figure 3.1 shows a set of SC operations implemented using SM-SC numbers. Figure 3.1a illustrates an SC multiplier taking two SM-SC numbers. It is simpler than a bipolar SC multiplier as it uses AND gates instead of XNOR gates for data bits.

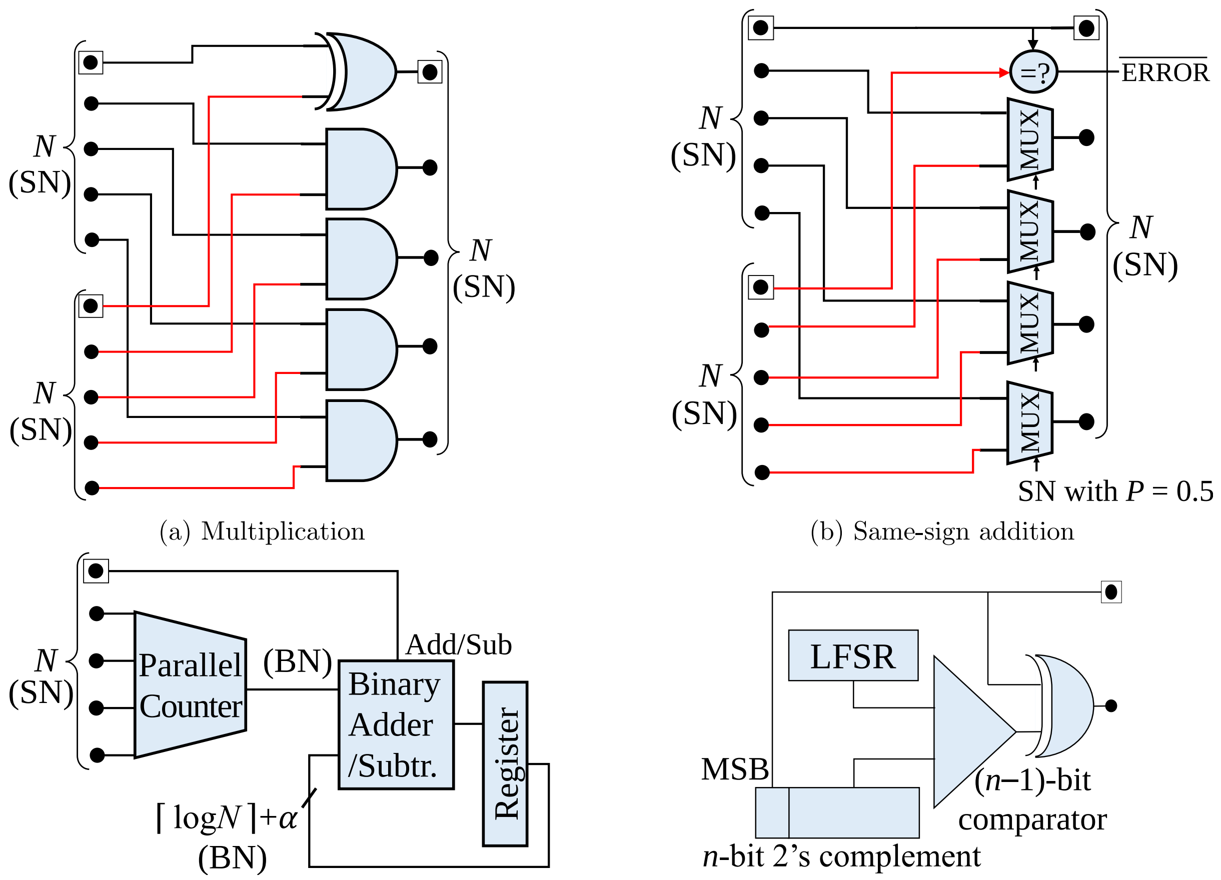
The addition of two SM numbers, on the other hand, is difficult in the general case. But if the sign bits are known to be the same, addition is no more complex than in the case of unipolar or bipolar SC, as shown in Figure 3.1b.

Accumulating a sequence of SNs into a binary-number register can be done efficiently using a parallel counter and a binary accumulator, which is also known as accumulative parallel counter (APC). Figure 3.1c shows the circuit. The only difference from APC is that this circuit requires that the output of the parallel counter be either added to or subtracted from the accumulation register depending on the input sign bit (as opposed to being always added to), but it can be implemented easily as shown in the figure.

Finally, Figure 3.1d is an SNG for SM-SC numbers. It takes an n-bit 2's complement number as input and generates an SM-SC bitstream. A straightforward implementation would be to first convert the 2's complement input into a sign-magnitude binary, which requires n XOR gates and an n-bit adder. Instead our circuit achieves it with just one XOR gate.

The correctness of our method can be shown as follows. Let x denote the input value. If $x \geq 0$, our SNG degenerates into the conventional SNG, and is therefore correct. For the $x < 0$ case, let's consider $x = -10$ and $n = 5$, as an example. The binary representation X is 10110 and its magnitude is $-x = \bar{X} + 1$ where $\bar{X}$ is 1's complement of X. The comparator will generate a bitstream for the lower $n - 1$ bits (0110), which is $-10 + 16$, or $x + 2^{(n-1)}$ in general due to the definition of 2's complement. The bitstream generated by the comparator is inverted by XOR, the effect of which is complementing it to $2^{(n-1)}$. In other words, the output of XOR gate corresponds to $2^{(n-1)} - (x + 2^{(n-1)}) = -x$. Thus it is correct in the $x < 0$ case as well.

In summary, while some operations such as general addition may be complicated in SM-SC than in the conventional SC, other operations have an efficient counterpart in SM-SC.



(a) Multiplication

(b) Same-sign addition

(c) Accumulation with binary output

(d) Stochastic number generator (SNG)

Figure 3.1: Multiplication, addition, accumulation, and SNG for sign-magnitude stochastic computing (SM-SC). Note: sign bit is marked by a small box around it.

CHAPTER IV

DNN Accelerator Based on SM-SC

As shown in the previous section there are different ways to implement even the same arithmetic operation in SM-SC, with very different cost and complexity ramifications. Therefore it is important to design the architecture taking cost and accuracy into consideration. Most critically it is better to avoid the general SC-adder with stochastic output.

We use a recently proposed SC-DNN accelerator architecture [16], which is based on an SC-MAC array of a fixed size ($M \times M \times M$). The core computation engine of this architecture is an SC-MAC array, where each SC-MAC performs dot-product between a sequence of input activations and a sequence of weight parameters. These input activation and weight parameter values are given in conventional binary, as they should be stored in memory until processed by the SC-MAC array. (SC-MAC array has a limited size and cannot handle in general all activation/weight data simultaneously.) A series of SNGs are also necessary for each SC-MAC. Figure 4.1 illustrates the structure of an SC-MAC.

This SC-MAC architecture performs accumulation in the binary domain, thus avoiding the limited range problem of SC. As such, it is a good match with our SM-SC that can provide an efficient implementation of SC-input binary-output MAC, by combining circuits in Figure 3.1a and Figure 3.1c.

One issue in implementing hardware for our SM-SC is the degree of bit-parallelism. For efficiency reasons, it is much better to set the bitstream length for SM-SC modules to $N = 2^k + 1$ bits, where 1 bit is for the sign. Since the sign bit is needed whether the SM-SC module is

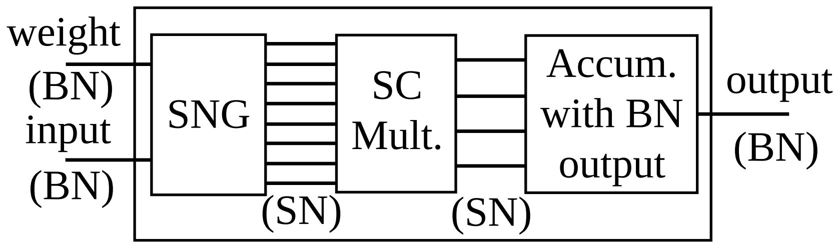


Figure 4.1: SC-MAC structure

fully bit-parallel, fully bit-serial, or something in between, it is advantageous to use higher bit-parallelism, which can amortize the overhead of the sign bit and associated logic. Higher bit-parallelism can also help amortize the overhead of, if any, binary circuits (e.g., binary adder, accumulation register). On the other hand, too much bit-parallelism can increase the critical path delay and adversely affect area due to aggressive timing optimization of synthesis tools.

CHAPTER V

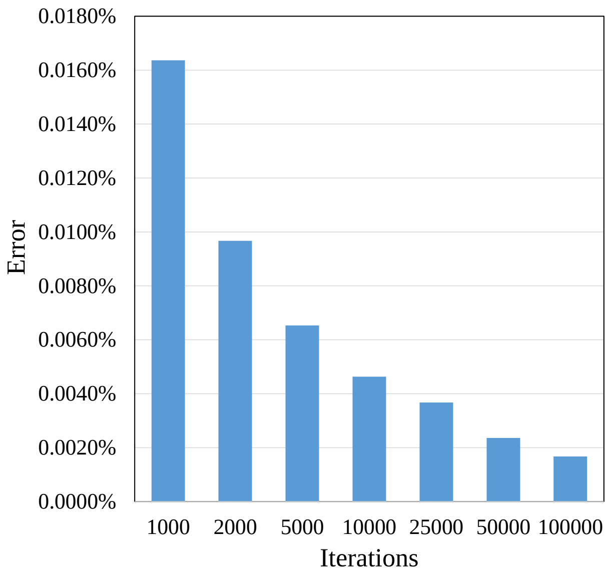
Experiments

To demonstrate the efficacy of our proposed technique, we compare our **SM-SC** scheme against **bipolar SC**, which is the *de facto* standard used in all prior SC designs requiring handling of signed numbers [3, 10, 11, 13–15].

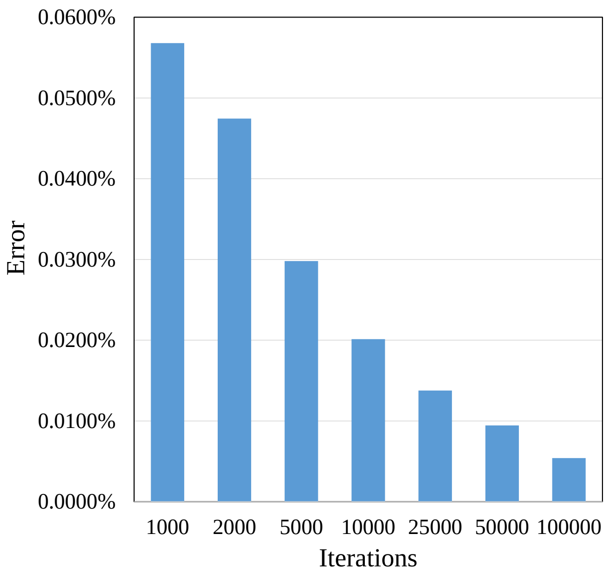
We perform our evaluations across multiple levels. First at the operation level, we compare SM-SC's accuracy on SC multiplication against that of bipolar SC. Second at the macro block level, we evaluate an SC-MAC array in depth through accuracy simulation and RTL synthesis. An SC-MAC array is a key building block used in many applications across different domains. Finally we present our results of applying SM-SC to various SC-DNN accelerators in terms of accuracy and hardware synthesis. The SC-DNNs that we evaluate cover two CNNs designed for different datasets and one RNN, of the type called LSTM (Long Short-Term Memory), but they all share the same SC-MAC array as the key SC building block.

For the evaluation at the operation and macro block levels, we use standard deviation as the metric for error. Specifically we repeat the same experiment with the same input 10,000 times to get the standard deviation of the output, which is referred to as absolute error. Then we repeat this by changing the input data.

For the evaluation at the application level (ie., SC-DNNs), we extend Caffe [8], a well-known deep learning evaluation framework, to support SM-SC as well as bipolar SC. Our accuracy metric here is the *test accuracy* reported by the tool. This setup is quite realistic as our acceleration model is that all the input and output of an SC-MAC is in conventional



(a) Unipolar



(b) Bipolar

Figure 5.1: Average error between proposed mathematical formulation and bit-level simulation

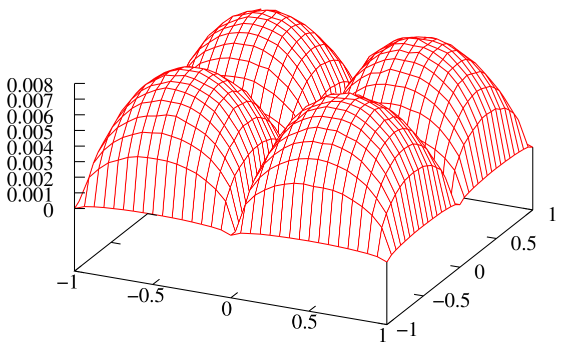
binary. This setup also allows us to do not only inference but also SC-in-the-loop retraining, which is known to boost the accuracy quite significantly.

5.1 Accuracy of mathematical formulation

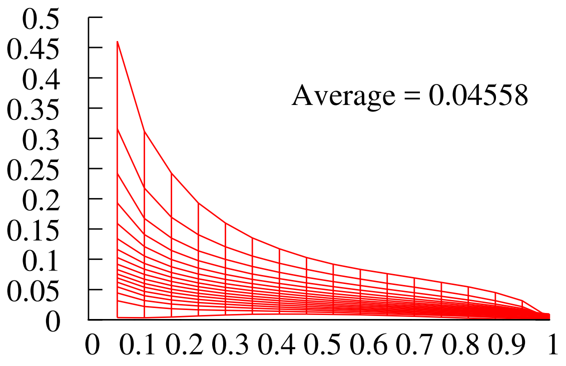
The accuracy of proposed mathematical formulation of SC multiplication against bit-level SC multiplication simulation is shown in Figure 5.1. Random bitstream length and SC input values are used for this comparison. Several different numbers of iteration are performed to get variation in bit-level simulation. To get the accurate behavior of bit-level simulation more iterations need to be performed, which gets closer to the proposed mathematical formulation. The difference in variance between the proposed equation and bit-level simulation is less than 0.002% for unipolar and 0.003% for bipolar SC multiplication, when 10,000 iterations of bit-level simulation is performed.

5.2 SC Multiplication Accuracy

Figure 5.2 shows the SC multiplication error for different input combinations when using our SM-SC. Figure 5.2a shows that the absolute error graph has the desired four peaks instead of one, with nine minima at all integer coordinate points (i.e., $(\pm1,\pm1)$, $(\pm1,0)$, $(0,\pm1)$, $(0,0)$).



(a) Absolute error, sign-magnitude



(b) Relative error, sign-magnitude

Figure 5.2: Multiplication error of our SM-SC for different input values. Bitstream length is 1024 bits including one sign bit. Result of 10,000 trials for each input combination.

Compared to bipolar case SM-SC shows overall better accuracy and can handle both positive and negative numbers.

Figure 5.2b confirms that our SM-SC also greatly reduce the relative error as compared with the bipolar case. In fact ours is very similar to the unipolar case, including the maximum relative error and average relative error values, except that ours can also handle negative values. This suggests that our SM-SC achieves its stated goal of realizing unipolar's performance on signed numbers for the multiplication operation. This is significant as we achieve it *without increasing bitstream length* and at the same time extend the dynamic range by 2X.

As compared with the bipolar case, our SM-SC is 6.3X better in terms of average relative error. There are two factors contributing to this. The first is that the (absolute) error is very low when either operand is near zero. This so-called near-zero problem (how to minimize error at near zero) can be addressed by other means too (e.g., near-zero weight removal in [10]), but we solve the problem through a simple change in the encoding without introducing any manipulation of the input values. Thus ours is more flexible and robust. In addition, our scheme works well for either operand being near zero, which is another advantage compared with the previous work. The other contributing factor is that due to the increased encoding efficiency, our scheme effectively uses 2X longer bitstreams, which helps reduce error while covering the full input range.

Also note that this level of accuracy enhancement cannot be achieved even if one uses twice longer bitstreams for bipolar. Overall, this result demonstrates that our SM-SC can achieve much better accuracy than bipolar encoding SC for multiplication.

5.3 MAC Array: Accuracy and Area Impact of SM-SC

Figure 5.3a shows the *relative* error of SC-MAC averaged across 100 different input vector combinations. Each input vector is 100-element sized, and drawn randomly from a uniform distribution. As input distributions may not follow uniform (e.g., weight parameters in DNNs), we limit the input range differently as shown on the x-axis of the graph. Even when we reduce the input range, the hardware is not redesigned for the reduced range; we use the same hardware for different data distributions. Stochastic bitstream length is 256-bit.

The graph shows a clear advantage of our SM-SC MAC over bipolar. The gap between them increases from 4X (for full-range) to 5.5X and to 9.5X (for quarter range) as we reduce the input range. The fact that a simple change in the encoding of SC numbers can give nearly 10X improvement in accuracy essentially for free is impressive, especially when we consider that 10X accuracy improvement in SC requires at least 10X increase in bitstream length and usually more.[1]

We have implemented bit-serial and 64-bit parallel versions in Verilog RTL and synthesized them using Synopsys Design Compiler with TSMC 45nm technology. Figure 5.3b shows our synthesis result of one SC-MAC comparing bipolar-SC and our SM-SC. The SNG part is not included as it is common in both cases and also because SNGs are typically shared across multiple SC-MACs in an array organization (Section 5.5 shows results including SNGs).

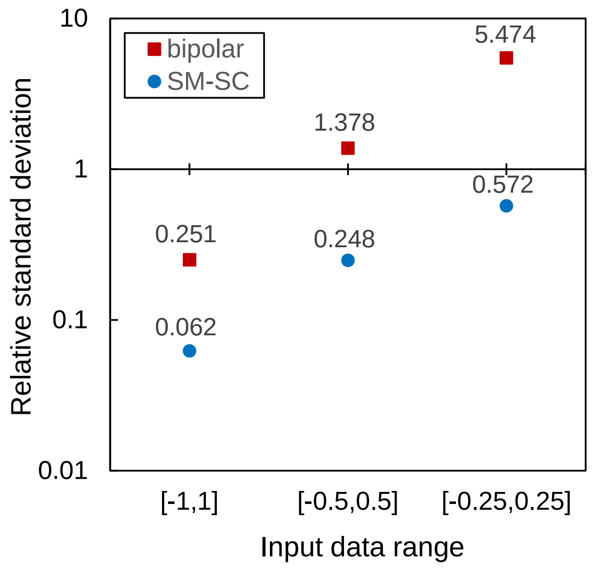
For the bit-serial version, we expected the overhead of the sign bit to be significant. However as it turns out, most of the area is due to the binary accumulator that includes saturating logic. As a result the SM-SC version adds only 8% overhead in area as compared to the bipolar-SC version. For the 64-bit version, which is more practical and far more cost-effective than the bit-serial version,[2] ours adds mere 1.3% overhead to the bipolar-SC version, without increasing the operating frequency.

5.4 SC-DNNs: Accuracy Comparison

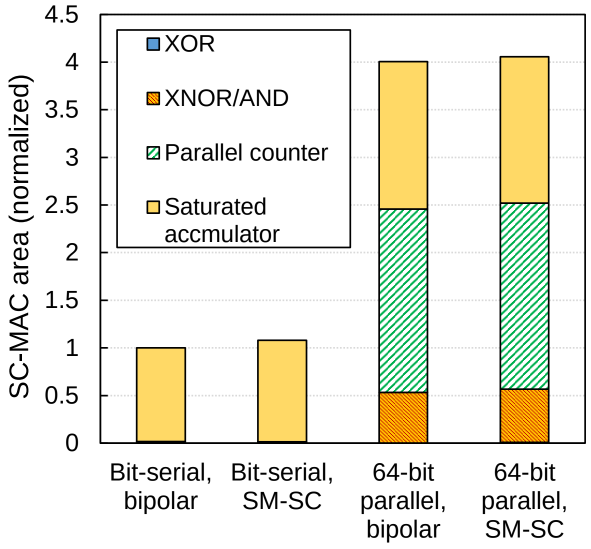
For SC-DNN evaluation we use two CNNs and one RNN. The CNN models are from the ones included in the Caffe distribution, targeting MNIST and Cifar-10 datasets. We apply SC to convolution layers only, which account for the majority of computation in CNNs. The RNN model is based on an example in [1], which consists of one LSTM layer and one fully-connected

[1] We also varied the bitstream length for SC-MAC array, which shows that to achieve the level of accuracy of our SM-SC at 32-bit, bipolar-SC requires about 1024-bit streams, agreeing with our result for SC-DNN applications in Section 5.4.

[2] The 64-bit versions are only about 4X larger in area though they can reduce latency by 64X compared to the bit-serial version.



(a) SC-MAC accuracy



(b) SC-MAC area

Figure 5.3: (a) Relative error (averaged over 100 input vector combinations using geometric mean) vs. input data range. (b) SC-MAC area synthesis result.

(FC) layer. We apply SC to all inner-product layers inside LSTM, which constitutes most of the computation in the RNN. The characteristics of used DNNs are summarized in Table 5.1.

5.4.1 MNIST-CNN

Figure 5.4a shows our test result using SC versions of MNIST-CNN. It is retrained by running additional 1 epoch in SC mode. The MNIST dataset is relatively easy, and the bipolar SC model also achieves good accuracy after retraining. But ours, even before retraining, is more accurate than retrained bipolar-SC model at 32-bit SC, and, after retaining, is virtually indistinguishable from the floating-point model. More significantly, if we compare the two retained SC models, one can see that our SM-SC's accuracy at 32-bit is the same as that of bipolar-SC at 1024-bit, which is 32X improvement in efficiency without accuracy degradation.

5.4.2 Cifar10-CNN

The Cifar-10 dataset is much more challenging than MNIST, and even the floating-point version's accuracy is about 81% without augmentation. We follow the default training procedure provided, which runs for 140 epochs, after which we fine-tune the model in SC for 20 more epochs.

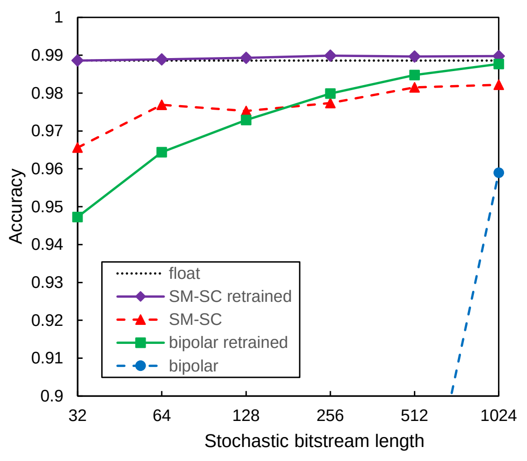
Figure 5.4b shows the advantage of SM-SC. The best accuracy using the conventional SC is about 5% point lower than that of the floating-point case, which is consistent with a recent

Table 5.1: DNN architectures

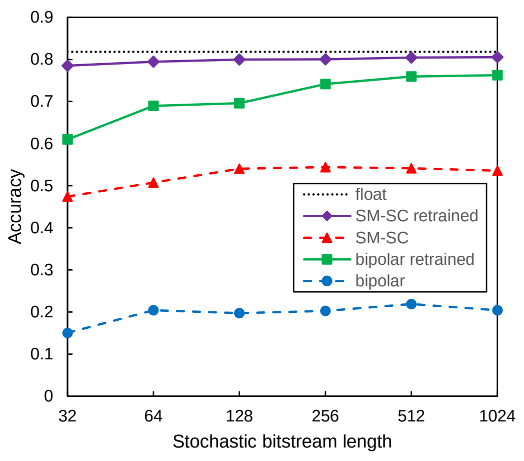
MNIST-CNN			
Layer	Ouput size	Filter / Stride	#Parameters
data	28x28x1		
conv1	24x24x20	5x5 / 1	520
pool1	12x12x20	2x2 / 2	
conv2	8x8x50	5x5 / 1	25,050
pool2	4x4x50	2x2 / 2	
ip1	500		400,500
ip2	10		5,010

Cifar10-CNN			
Layer	Ouput size	Filter / Stride	#Parameters
data	32x32x3		
conv1	32x32x32	5x5 / 1	2,432
pool1	16x16x32	3x3 / 2	
norm1	16x16x32		
conv2	16x16x32	5x5 / 1	25,632
pool2	8x8x32	3x3 / 2	
norm2	8x8x32		
conv3	8x8x64	5x5 / 1	51,264
pool3	4x4x64	3x3 / 2	
ip1	10		10,250

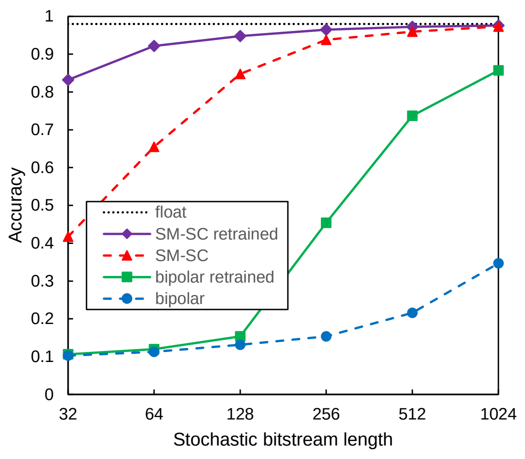
MNIST-LSTM			
Layer	Output size	#Timesteps	#Parameters
data	28	28	
lstm1	128	28	80,384
ip1	10		1,290



(a) CNN accuracy with MNIST dataset



(b) CNN accuracy with Cifar-10 dataset



(c) LSTM accuracy with MNIST dataset

Figure 5.4: SC-DNN accuracy.

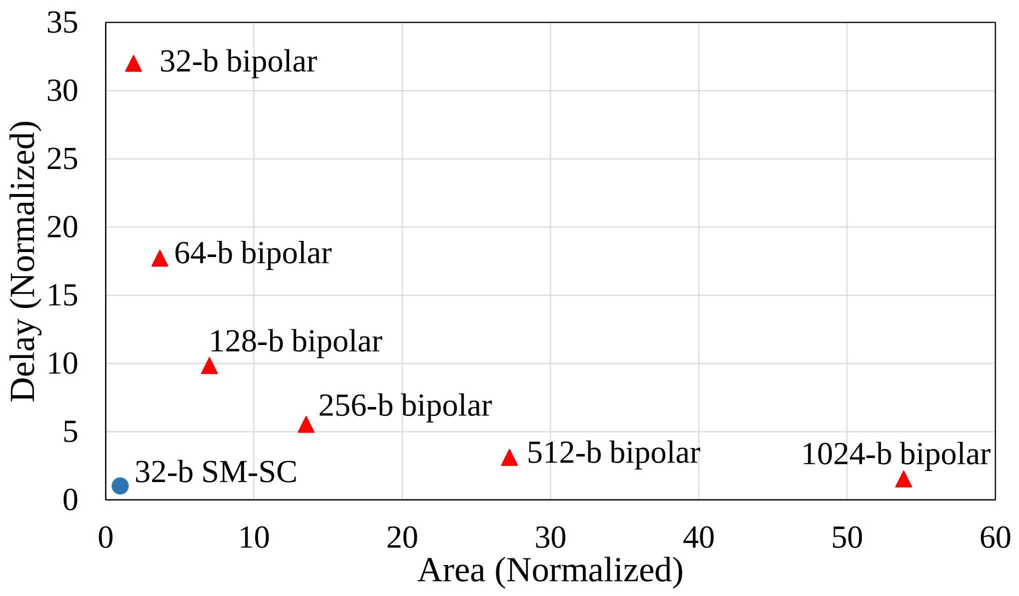


Figure 5.5: Area and delay comparison between 1024-bit bipolar SC vs. 32-bit SM-SC. Different MAC designs are considered for bipolar SC.

result [19] obtained with an optimized training method for a roughly similar platform. Once again we observe that our SM-SC's result at 32-bit precision is even better than that of bipolar-SC at 1024-bit, which suggests that our encoding can give more than 32X improvement in efficiency. Since the x-axis is in log-scale, the bitstream length must be much longer in order for bipolar-SC to reach the 80% accuracy.

5.4.3 MNIST-LSTM

The LSTM cell we use has three control gates, each of which needs a fully-connected layer between input (plus states) and hidden neurons, and therefore has high computational complexity [18]. We use a single LSTM layer with 128 hidden neurons. To provide input for the network, each 28x28 pixel image is split row-by-row, and each row is fed to the network at each time step. The network is trained in floating-point for 100 epochs with learning rate of 0.001, and retrained for additional 20 epochs in SC mode, with the learning rate reduced by 10X.

Figure 5.4c shows the result. This network is more challenging to SC, and the bipolar-SC can't recover even 90% of the floating-point accuracy even at 1024-bit. But our SM-SC outperforms bipolar-SC regardless of retraining. Comparing the two retrained SC models, the conventional SC's accuracy at 1024-bit can be nearly achieved by our SM-SC at 32-bit, again demonstrating the 32X improvement in efficiency by our technique.

Table 5.2: Comparison with previous neural network accelerators

		Frequency	Area*	Power*	Tech.
Binary	GLSVLSI'15 [5]	700 MHz	0.98	236.59	65nm
SC	Arxiv'15 [3]	400 MHz	0.09	14.90	65nm
	DAC'16 [10]	1000 MHz	0.06	3.60	45nm
	ASPDAC'17 [16] (without SNG)	1540 MHz	0.43	279.31	45nm
	SM-SC 32 bit (without SNG)	1666 MHz	0.18	115.94	45nm
	SM-SC 32 bit (with SNG)	1563 MHz	1.09	2619.83	45nm

		GOPS	GOPS/mm^2	GOPS/W	Scope for area & power
Binary	GLSVLSI'15 [5]	274.00	278.85	1158.11	SoP ($\simeq$ MAC) units only
SC	Arxiv'15 [3]	1.01	11.91	67.93	One neuron
	DAC'16 [10]	75.74	1262.33	21038.79	One neuron with 200 inputs
	ASPDAC'17 [16] (without SNG)	1575.38	3697.81	5640.23	MAC array (size: 8x8x8)
	SM-SC 32 bit (without SNG)	1706.66	9290.28	14719.95	MAC array (size: 8x8x8)
	SM-SC 32 bit (with SNG)	1600	1458.81	610.72	MAC array (size: 8x8x8)

*Note 1: For the scope for area (in mm^2) and power (in mW), see the rightmost column.

5.5 SC-DNNs: Area and Delay

Figure 5.5 compares 1024-bit bipolar SC vs. our 32-bit SM-SC in terms of area and delay of the main computing MAC array. The bitstream lengths are chosen based on the accuracy result in Figure 5.4. The area and delay of SNG are included in the results. For bipolar SC, we vary the bit-parallelism to get different Pareto-optimal points.[1] The result confirms the extreme efficiency of SM-SC compared with bipolar SC. One interesting fact in this comparison is the area reduction even between 32-bit bipolar vs. 32-bit SM-SC, which is due to the use of larger LFSRs and comparators (10-bit used) in bipolar to satisfy 1024-cycle period.

5.6 SC-DNNs: Fault Tolerance

Figure 5.6 compares fault tolerance results among SC schemes for the Cifar-10 CNN. We have implemented the fault model of [12] in Caffe, which is to randomly flip input bits of a MAC array based on a set probability. The x-axis shows this probability. Errors are injected to convolution layers only. The graph shows that our proposed SM-SC is more accurate at low fault rates than bipolar SC, but becomes worse as the fault rate increases. This is due to the sign bit, which is special and arguably has a higher weight than other bits of SM-SC. However,

[1]We have also compared power of each design, which was similar to the area result and thus was omitted.

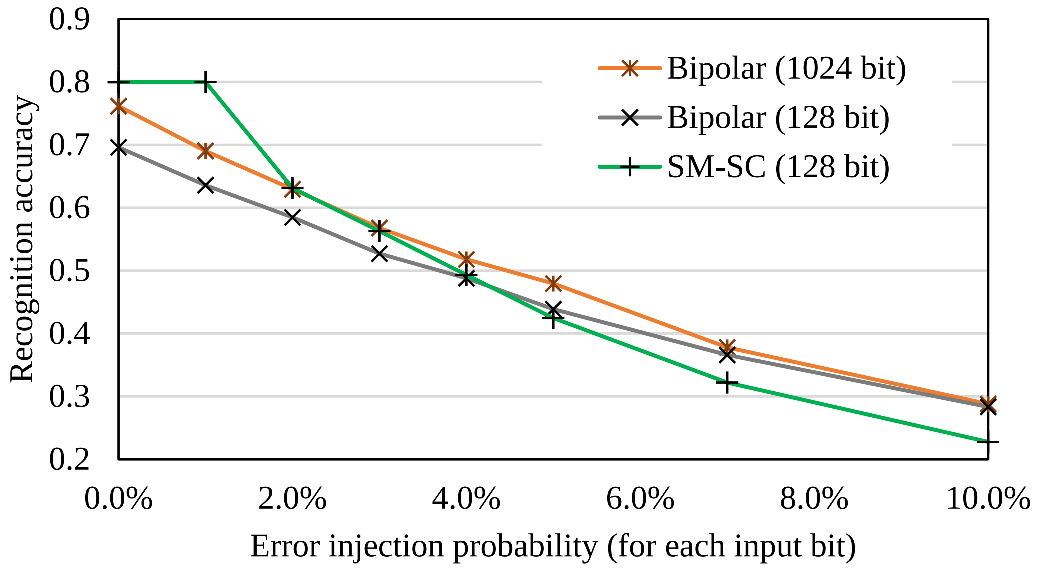


Figure 5.6: Fault tolerance result.

the difference in accuracy is within about 5% point, which is very small considering the huge area and delay differences between the two.

CHAPTER VI

Related Work

There is a body of research that applies SC to various neural network implementations [3, 4, 7, 10, 11, 13–15]. [4] is one of the earliest work applying SC to neural networks. Recently, SC has been applied to deep belief networks [14] and radial basis function neural networks [7] as well.

Table 5.2 compares our SC-DNNs with other SC-DNNs and one non-SC DNN design. Compared to other SC-DNNs ours is very efficient in terms of operation-per-area. Ours is also better than the conventional binary design in terms of operation-per-area even when including SNG overhead. This is mainly due to the accuracy improvement, which enables us to use much shorter bitstreams to do essentially the same computation. Note that the SNG is implemented using LFSR and comparator, which can be improved using even distribution SNG [21].

[15] has recently proposed a new SC multiplier that is very accurate and efficient. However, it is applicable to convolution layers only, and not applicable to fully-connected layers. Ours is applicable to any SC-MAC and therefore any layers including fully-connected layers, as exemplified by RNNs in our experiments.

Authors in [10] first observed the low accuracy problem of SC-multiplication near zero but their solution is hard-wired, and not scalable. As compared with [10], our method has the following merits when it comes to the inaccurate SC-multiplication problem. First, ours can support a very scalable architecture. Second, ours does not require *a priori* knowledge of operand values; consequently, it can be applied to whichever operand (or both operands) is near zero. Finally, no special retraining is necessary to use our method.

The architecture in [11] uses bipolar by default but also uses unipolar partially, in order to use OR-based SC-addition. Thus, not only is the motivation different from ours, but they also assume *a priori* knowledge of input signs similar to [10] in order to divide the adder tree into positive and negative subtrees, limiting its scalability and applicability to more general DNNs.

CHAPTER VII

Conclusion

In this work we proposed SM-SC, a new encoding scheme for stochastic numbers, along with associated SC operations. Our idea is very simple, yet it achieves great improvement in accuracy and efficiency of SC where SC-multiplication is used. Unlike previous methods, ours does not require any sign-based regrouping of input, which complicates hardware design with added overhead, or value-based filtering of input, which limits applicability and effectiveness. Our SM-SC shows about 10X improvement in accuracy in our SC-MAC accuracy evaluation, and consistent 32X improvement (32-bit vs. 1024-bit) in our DNN evaluation. With so much improvement in accuracy and efficiency, SM-SC is definitely a worthwhile optimization to have in designer's toolbox.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. Tensorflow: A system for large-scale machine learning. *CoRR*, abs/1605.08695, 2016. 16

[2] Armin Alaghi and John P. Hayes. Fast and accurate computation using stochastic circuits. In *DATE '14*, pages 76:1–76:4, 2014. 1

[3] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross. Vlsi implementation of deep neural networks using integral stochastic computing. In *ISTC*, pages 216–220, Sept 2016. 1, 13, 21, 23

[4] B. D. Brown and H. C. Card. Stochastic neural computation. ii. soft competitive learning. *IEEE Transactions on Computers*, 50(9):906–920, Sep 2001. 23

[5] Lukas Cavigelli, David Gschwend, Christoph Mayer, Samuel Willi, Beat Muheim, and Luca Benini. Origami: A convolutional network accelerator. In *GLSVLSI '15*, pages 199–204, New York, NY, USA, 2015. ACM. 21

[6] B.R Gaines. Stochastic computing systems. *Advances in Information Systems Science 2*, pages 37–172, 1969. 4

[7] Y. Ji, F. Ran, C. Ma, and D. J. Lilja. A hardware implementation of a radial basis function neural network using stochastic logic. In *DATE*, pages 880–883, March 2015. 1, 23

[8] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014. 13

[9] D. Kim, M. S. Moghaddam, H. Moradian, H. Sim, J. Lee, and K. Choi. Fpga implementation of convolutional neural network based on stochastic computing. In *ICFPT*, pages 287–290, Dec 2017. 1

[10] Kyounghoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoung Choi. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. In *DAC '16*, pages 124:1–124:6, New York, NY, USA, 2016. ACM. 1, 5, 13, 15, 21, 23, 24

[11] Bingzhe Li, M. Hassan Najafi, and David J. Lilja. Using stochastic computing to reduce the hardware requirements for a restricted boltzmann machine classifier. In *FPGA '16*, pages 36–41. ACM, 2016. 1, 13, 23, 24

[12] W. Qian, X. Li, K. Bazargan, and D. J. Lilja. An architecture for fault-tolerant computation with stochastic logic. *IEEE Transactions on Computers*, 60(1), 93-105, 2011. 21

[13] Ao Ren, Ji Li, Zhe Li, Caiwen Ding, Xuehai Qian, Qinru Qiu, Bo Yuan, and Yanzhi Wang. SC-DCNN: highly-scalable deep convolutional neural network using stochastic computing. *CoRR*, abs/1611.05939, 2016. 1, 13, 23

[14] K. Sanni, G. Garreau, J. L. Molin, and A. G. Andreou. Fpga implementation of a deep belief network architecture for character recognition using stochastic computation. In *CISS*, pages 1–5, March 2015. 1, 13, 23

[15] H. Sim and J. Lee. A new stochastic computing multiplier with application to deep convolutional neural networks. In *2017 54th ACM/EDAC/IEEE DAC*, pages 1–6, June 2017. 1, 13, 23

[16] H. Sim, D. Nguyen, J. Lee, and K. Choi. Scalable stochastic-computing accelerator for convolutional neural networks. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 696–701, Jan 2017. 11, 21

[17] Hyeonuk Sim, Saken Kenzhegulov, and Jongeun Lee. Dps: Dynamic precision scaling for stochastic computing-based deep neural networks. In *DAC'18*, 2018. 1

[18] Shuohang Wang and Jing Jiang. Learning natural language inference with LSTM. *CoRR*, abs/1512.08849, 2015. 20

[19] Yandan Wang, Wei Wen, Linghao Song, and Hai (Helen) Li. Classification accuracy improvement for neuromorphic computing systems with one-level precision synapses. *CoRR*, abs/1701.01791, 2017. 20

[20] J. Yu, K. Kim, J. Lee, and K. Choi. Accurate and efficient stochastic computing hardware for convolutional neural networks. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 105–112, Nov 2017. 1

[21] A. Zhakatayev, K. Kim, K. Choi, and J. Lee. An efficient and accurate stochastic number generator using even-distribution coding. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, PP(99):1–1, 2018. 1, 4, 23