Quantized Neural Networks with New Stochastic Multipliers

Bingzhe Li*, M. Hassan Najafi*, Bo Yuan†, and David J. Lilja*

* Department of Electrical and Computer Engineering, University of Minnesota–Twin Cities
† Department of Electrical Engineering, City University of New York
{lixx1743, najaf011, lilja}@umn.edu, byuan@ccny.cuny.edu

***Abstract*—With increased interests of neural networks, hardware implementations of neural networks have been investigated. Researchers pursue low hardware cost by using different technologies such as stochastic computing and quantization. For example, the quantization is able to reduce total number of trained weights and results in low hardware cost. Stochastic computing aims to lower hardware costs substantially by using simple gates instead of complex arithmetic operations. In this paper, we propose a new stochastic multiplier with shifted unary code adders (SUC-Adder) for quantized neural networks. The new design uses the characteristic of quantized weights and tremendously reduces the hardware cost of neural networks. Experimental results indicate that our stochastic design achieves about 10x energy reduction compared to its counterpart binary implementation while maintaining slightly higher recognition error rates than the binary implementation.**

***Keywords*-Neural networks, Stochastic computing, Quantization, Multiplier**

1. Introduction

Neural networks as a computational model based on neurons, are becoming a prevalent method in many areas. Researchers start to investigate neural networks with hardware implementation, rather than restricting to software implementations. In the hardware implementations, two major directions are explored. One aims to accelerate neural network performance by using FPGA or VLSI designs [1][2]. The other targets to low-power applications such as mobile devices and Internet-of-things (IoT).

To achieve low hardware cost, quantization and stochastic computing have been investigated, respectively. First, stochastic computing [3] is known as a low-cost and fault-tolerate technology used in approximate computation areas. By using simple gates to implement complex arithmetic operations, stochastic computing achieves extremely low hardware cost. For example, several finite state machines are able to implement exponential function or tanh function. Thus, stochastic computing is a promising approach to reduce hardware cost in many applications. Second, the quantization is an approximate method in digital designs. In terms of smaller quantization levels, quantization introduces larger quantization errors while reducing complexity of applications significantly. Therefore, to investigate low-cost neural networks, people applied those two approaches into neural network implementations, respectively.

On one hand, previous works have investigated neural networks by using stochastic computing. For example, the stochastic RBM implementation was proposed by Li *et al.* [4][5]. Another work [6] implemented a tanh based neuron and improved error rates of stochastic neural network substantially. Two works [7][8] implemented deep stochastic CNNs with approximate parallel counters using bipolar and unipolar encoding formats, respectively. On the other hand, some researchers studied efficient hardware implementations of neural networks with the quantization technology. For example, Hwang *et al.* [9] studied fixed-point feedforward neural networks with different quantization levels. According to their results, they achieved similar recognition error rates compared to floating-point neural networks. Another work [10] exploited sparseness in vocabulary speech recognition to reduce the model size and execution time. However, the advantages of both quantization and stochastic computing in neural networks are not well investigated. Therefore, there is an opportunity to design a quantized neural network by using stochastic computing in order to reduce hardware cost of neural networks further.

In this paper, we proposed a new stochastic neural network architecture for quantized neural networks. The goal of this work is to combine two technologies, quantization and stochastic computing in neural networks and then to reduce hardware cost of neural networks further. Regarding neural network implementations, first we retrain a fully trained neural network with the back-propagation algorithm in order to obtain neural networks with quantized weights. Second, according to the quantized weights, a stochastic matrix multiplication is implemented with a new component called shifted unary code adder (SUC-Adder). The SUC-Adder is capable of efficiently summing up products of matrix elements and quantized weights. As a result, the SUC-Adder can decrease the number of inputs of parallel counters and then reduce hardware cost of the whole neural network.

The main contributions of this paper are summarized as followed:

- Retrained neural networks with different quantization levels are implemented by stochastic computing.
- A stochastic quantized matrix multiplier is proposed with SUC-Adders for the quantized neural networks. The method can efficiently and accurately achieve high accuracy of partial matrix multiplication by only using several AND and OR gates.
- The stochastic neural networks with proposed multipliers achieve much lower hardware costs compared to previous works and binary implementations while maintaining very close recognition error rates to binary neural networks.

The remainder of this paper is organized as follows: Section 2 demonstrates the motivation of this work and also introduces the background of stochastic computing and quantized neural networks. Section 3 provides the neural networks stochastic implementation. The experimental results of neural network comparisons are discussed in Section 4. The conclusions are given in Section 5.

2. Stochastic Computing and Quantized Neural Networks

2.1. *Motivation*

A deep neural network normally has thousands of weights, which result in long execution time within CPU/GPUs or large hardware cost for neural network accelerators. Therefore, previous researchers reduced the model size of deep neural networks by quantizing all trained weights. Consequently, the quantized weights decreased complexity of neural networks and finally lowered execution time and hardware cost in both conventional hardware and CPU/GPU implementations.

Similar to binary implementations, one advantage of quantization in stochastic computing is that the number of non-zero weights is reduced because because near-zero weights are quantized to zeros. As seen in Fig. 1, a large portion of weights in neural networks are around zero. Therefore, removing near-zero weights can reduce a large number of operations. Another advantage is that original weights in binary implementations are quantized to their nearby levels and thus thousands of weights are replaced by several constant values. As a result, the quantized weights will further reduce resource utilization.

Nevertheless, directly applying stochastic computing to quantized neural network may only obtain limited benefit. As the second advantage mentioned above, the benefit of quantized neural networks for stochastic computing is reducing the number of stochastic number generators (SNG). However, because of correlation between bit-streams, for the same value, we cannot encode them by the same SNG when they are operated by each other in conventional stochastic implementations. In addition, some researchers [11][12] have investigated low-cost SNGs. As a result, the benefit of reducing number of SNGs will become much smaller in future. Therefore, how to efficiently apply stochastic computing to the quantized neural networks motivates us to design a new stochastic neuron architecture.

2.2. *Stochastic Computing*

Stochastic computing is performed with random bit-streams. In stochastic computing domain, input data is encoded by two ways, unipolar and bipolar formats. These two formats are encoded by $Pr(X) = x$ and $Pr(X) = (x + 1)/2$ [13], respectively. Therefore, based on two encoding formats, their corresponding operations are investigated. For example, regarding addition, Dickson *et al.* used a simple OR gate to perform unipolar addition and Qian *et al.* [14] used a simple MUX to implement both unipolar and bipolar format addition. Moreover, for multiplication, simple AND [15] and XNOR gate [16] are implemented for unipolar and bipolar formats, respectively.

In this work, we use unipolar encoding format throughout stochastic neural network implementations. Thus, AND gates and OR gates are used for some basic operations like addition and multiplication in the stochastic neural networks. Moreover, another important part is the activation function in neural networks. We use the restricted Boltzmann machine (RBM) with sigmoid activation function. Some previous works have investigated stochastic sigmoid implementation. For example, Li *el at.* [4] proposed a stochastic sigmoid function based on stochastic tanh() function using finite state machines. Another work [8] used approximate parallel counters connecting finite-state machines to implement the sigmoid function. In our work, the sigmoid activation function is integrated into the neuron architecture with the multiplier. The details are introduced in Section 3.

2.3. *Neural Network Retraining*

In general, a neuron implements a function of $p = \sigma(\mathbf{A} * x + \mathbf{B})$, where σ is the activation function, x is input of the neuron, **A** is the coefficient for matrix multiplication, and **B** is the bias. Binary neural networks quantize all weights (**A** and **B**) and inputs (x) in retraining process [9] in order to decrease sizes of neural networks and then reduce resource utilization. However, the trade-off is that the quantized neural networks increase recognition error rates.

For stochastic quantized neural networks, we only quantize the coefficient **A**. The reasons are given as follows: First, our new proposed architecture obtains benefits from the weight **A** which is used for multiplications. Thus, it is not necessary to quantize inputs and other weights **B** in our implementations. Second, we use the same un-quantized input images as previous works in order that our work is designed for the same input datasets as the previous works. Third, since quantization reduces precision of neural networks, quantizing unnecessary weights results in increasing error rates. As seen in Fig. 2, the results indicate the neural network with partially quantized weights has better recognition error rates than the one with fully quantized weights. Therefore, to obtain lower recognition error rates, we only quantized coefficient **A**.

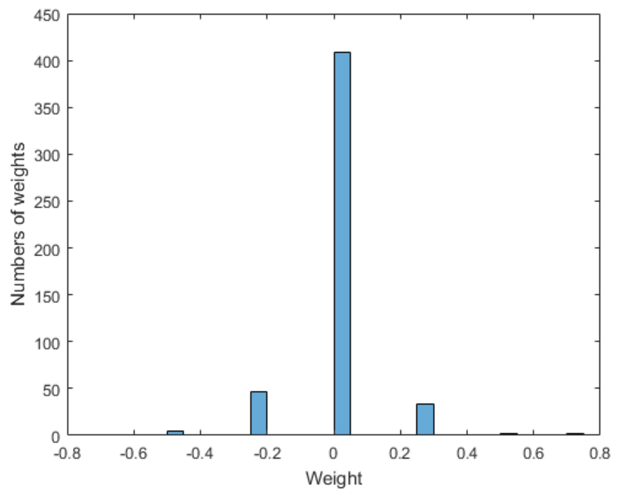
In our training process, we use the following steps retraining neural networks in order to obtain the quantized weights:

1) Fully train neural networks with floating-point weights.
2) Choose number of iterations to minimize the output error.
3) Quantize weights used for multiplications in hidden layers.
4) Remove near-zero biases of hidden layers.
5) Perform back-propagation algorithm [17] to update weights.
6) Repeat Step 3 to Step 5 to minimize the output error and quantize partial weights.

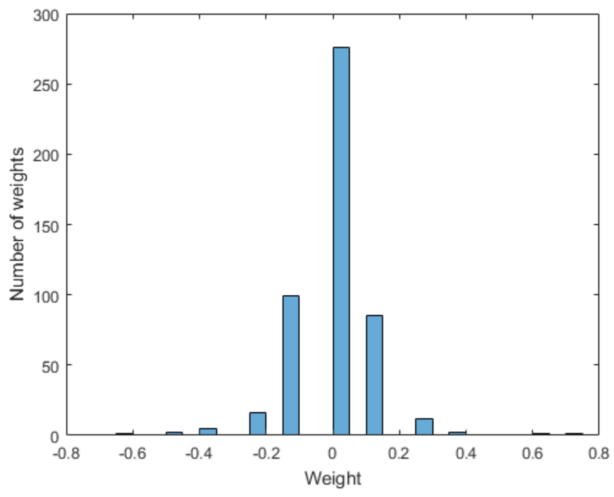
After retraining the neural networks, we obtain the quantized neural networks and all weights are regarded as constant values for stochastic neural network classifiers. In addition, the recognition error rates can be found in Table II. Compared to the floating-point neural network, the quantized neural network has slightly higher recognition error rates.

3. Stochastic Neural Network Implementations

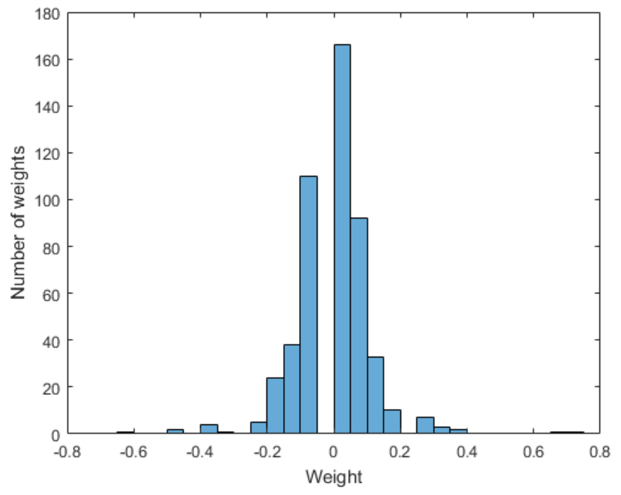
In this section, we introduce stochastic implementations for quantized neural networks. We use 2-bit quantization as an



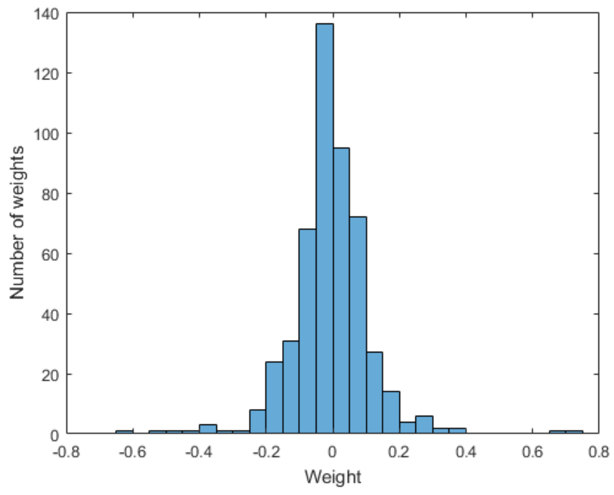
(a) 2-bit



(b) 3-bit



(c) 4-bit



(d) Floating-point

Fig. 1: Retraining with quantization levels of 2-bit, 3-bit, 4-bit and original floating-point (no retraining).

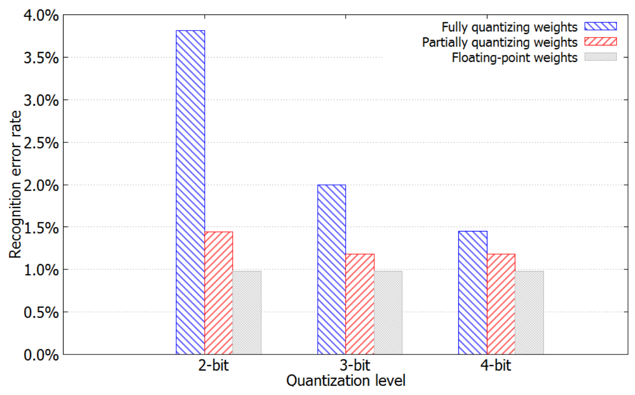


Fig. 2: Error rate comparisons between fully quantizing weights, partially quantizing weights and floating-point weights.

example and the architecture of the stochastic implementation are quite similar for other quantization levels.

3.1. *Stochastic Quantized Bit-streams*

In our stochastic implementation, multiplication for unipolar format bit-streams is achieved by ANDing two inputs. One of the inputs is quantized weight as a constant value. For 2-bit quantized weights with unipolar format, the quantized weights only have five values, 0, 0.25, 0.5, 0.75, and 1 in stochastic computing domain. Thus, the multiplication is changed to select 0%, 25%, 50%, 75% and 100% bits of the other input bit-stream and then pad un-selected bits with zeros. As a result, only selected bits contain useful information for future addition operations. Therefore, considering how to efficiently use the unselected bits, we propose a new type of bit-stream for the quantized weights.

The quantized weight bit-streams are similar to the time-based unary streams in previous works [18][19]. The difference is that our quantized weight bit-streams have contiguous '1's located in different phases. As seen in Fig. 3, it is an example for 2-bit quantized weights and the quantized bit-streams are generated from unary bit-streams by shifting

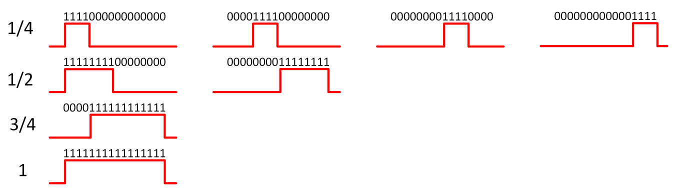


Fig. 3: Sequences of 2-bit quantized weights.

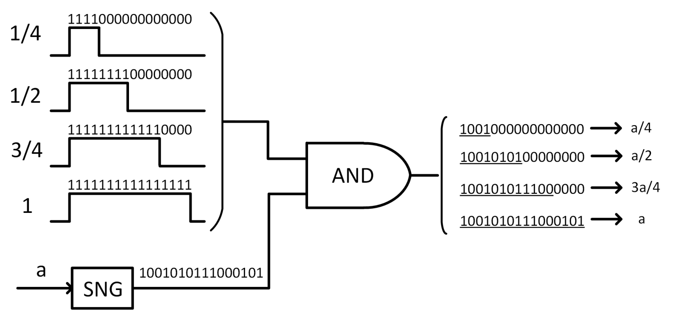


Fig. 4: An example of multiplication with 2-bit quantized weights.

different bits. By using the quantized weight bit-streams, the multiplication becomes to select a sequential part of a bit-stream as seen an example in Fig. 4. Finally, the output of quantized multiplication consists of a sequential part of one input and a sequence of zeros. The advantage of such encoding method is introduced in the following sections.

3.2. *Stochastic Quantized Addition*

As discussed above, the multiplication with 2-bit quantized weights truncates inputs with a number of sequential bits and pads them with zeros. Therefore, if the output bit-stream of the multiplication is padded with another product of two bit-streams instead of padding zeros, the output bit-stream will become the sum of two products (suppose two product bit-streams have enough number of padded zeros). For example, suppose an operation is to compute $a/4 + b/4 + c/4 + d/4$. As seen in Fig. 5, four bit-streams (a, b, c and d) generated by four stochastic number generators (SNGs) are ANDed with four shifted unary bit-streams of 1/4 whose phases of '1's are interleaved. That is, those four bit-streams of 1/4 select different part of a, b, c and d. Consequently, after going through OR gates, the output becomes $(a+b+c+d)/4$. Therefore, the four products are summed up by using only seven simple gates in this example. We call the adder as **S**hifted **U**nary **C**ode **Adder** (SUC-Adder), whose inputs are products of shifted unary bit-streams and conventional stochastic bit-streams.

Moreover, for different combinations of quantized weights, the SUC-Adders are implemented by different circuits. Fig. 6 lists four types of SUC-Adders for 2-bit quantized weights. For stochastic quantized neural network implementation, since 2-bit quantized weights are constant values and we know the total number of each quantized value, the number and types of SUC-Adders are determined after training neural networks.

Compared to traditional unipolar adder (only OR gate), as seen in Table I, our SUC-Adder has much better accuracy than the OR based adders. The analysis is given in Section 3.3.

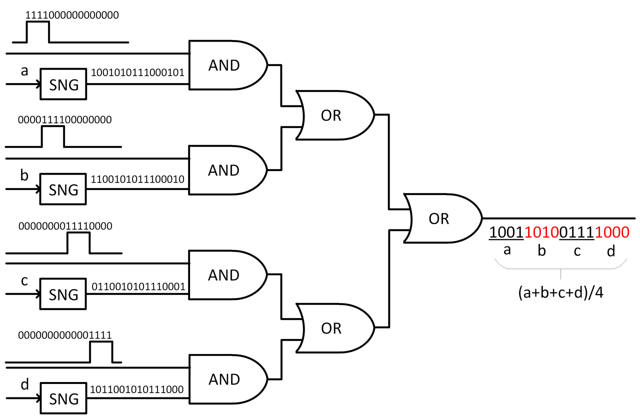


Fig. 5: An example of a four-input SUC-Adder with 2-bit quantized weights.

TABLE I: Mean absolute error (MAE) comparison between quantized weight generator and LFSRs

Bit-length	16	32	64	128
Operation	$(a+b+c+d)/4$			
(i) Quantized weight generator	7.74%	5.17%	3.61%	2.50%
(ii) Same LFSR	9.30%	6.12%	4.00%	2.75%
(iii) Different LFSRs	7.82%	5.33%	3.64%	2.53%
(iv) OR with same LFSR	26.9%	26.8%	26.7%	26.7%
(v) OR with different LFSRs	11.8%	10.2%	9.40%	8.97%
Operation	$(a+b+c+d+e+f+g+h)/8$			
(i) Quantized weight generator	8.16%	5.60%	3.88%	2.69%
(ii) Same LFSR	11.88%	7.80%	5.03%	3.28%
(iii) Different LFSRs	8.50%	5.63%	3.94%	2.75%
(iv) OR with same LFSR	37.6%	37.7%	37.4%	37.5%
(v) OR with different LFSRs	12.5%	11.0%	10.3%	9.91%

In addition, compared to previous stochastic neural network implementations [6][7][20], the SUC-Adder can reduce the number of inputs of parallel counters because it simply implements multi-input addition with several AND and OR gates. Therefore, the stochastic quantized neural networks with SUC-Adders will reduce hardware cost a lot compared to previous works.

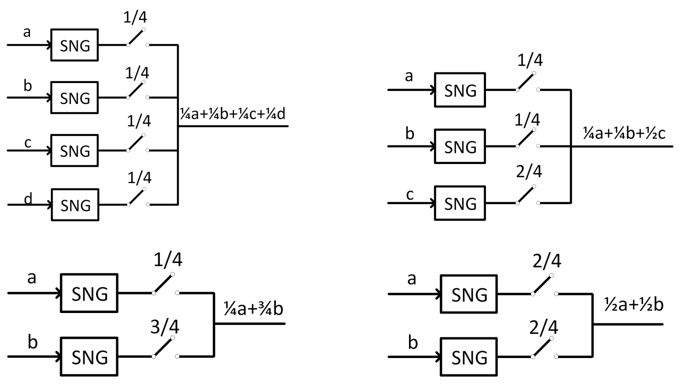


Fig. 6: SUC-Adders with different combinations of weights. All quantized weight bit-streams have different phases of '1's in each circuit.

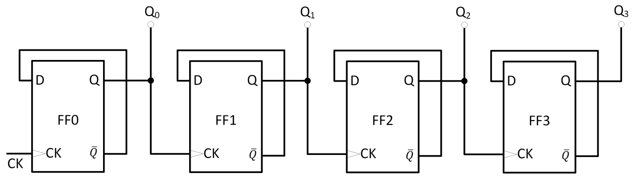


Fig. 7: An example of quantized weight generator for bit-streams with a period of 16 bits. A binary value and $D_3D_2D_1D_0$ go through a comparator to generate a bit-stream of the binary value.

3.3. *Stochastic Quantized Weight Generator*

In this section, we mainly focus on the stochastic quantized weights generator. As discussed in Section 3.1, quantized weights have 2^n non-zero values (n is the quantization level) in stochastic computing. Since each non-zero value needs to be encoded to interleaved bit-streams for SUC-Adders, for one non-zero value $i/2^n$ $(1 \leq i \leq 2^n)$, there are $\lfloor 2^n/i \rfloor$ types of shifted unary bit-streams. Therefore, it totally needs $\sum_{i=1}^{n} \lfloor 2^n/i \rfloor$ quantized weight generators used in the whole neural networks.

The design of the quantized weight generator is shown in Fig. 7, which is an example for producing bit-streams with a period of 16 bits. To generate bit-streams of different levels of quantized weights and different phases of '1's, we only need to initialize D Flip-flops by loading desired values.

To investigate accuracy of the shifted unary bit-streams, we use the operations of $(a+b+c+d)/4$ and $(a+b+c+d+e+f+g+h)/8$ with 3-bit quantized weights. Then, we compare our generator with linear-feedback shift registers (LFSR) by the mean absolute error (MAE). Moreover, we compare conventional OR adders with our design as well. In the implementation, all inputs are encoded by LFSRs and the coefficients (1/4 and 1/8) are produced by different methods. Five configurations are compared: (i) SUC-Adder: the same coefficient values are generated by the same quantized weight generator. (ii) Parallel adder: the same coefficient values are generated by the same LFSR. (iii) Parallel adder: the same coefficient values are generated by different LFSRs. (iv) OR adder: the same coefficient values are generated by the same LFSR. (v) OR adder: the same coefficient values are generated by different LFSRs.

As seen in Table I, because of correlation the OR adder with same LFSR gets the worst MAE results and the OR adder with different LFSRs are worse than our design. In addition, our quantized weight generator has even better MAE results with different bit lengths than the parallel counter implementations. Therefore, it proves that bit-streams generated by our quantized weight generators can obtain similar or even better accurate results compared to the conventional LFSR implementations. The details of hardware comparison of generators are shown in Section 4.2.

3.4. *Stochastic Neuron Implementation*

A single neuron is the basic unit in neural networks, which consists of a matrix multiplication and an activation function. For the structure of a stochastic neuron shown in Fig. 8, the inputs of neurons come from image bit-streams generated by stochastic number generators or outputs of previous layers. Since image pixel inputs and outputs of previous layers (outputs of sigmoid function) are always in the range of $[0, 1]$, the signs of products of inputs and quantized weights are determined by the quantized weights. Therefore, the number of positive and negative products are clearly known when the classifiers have been trained.

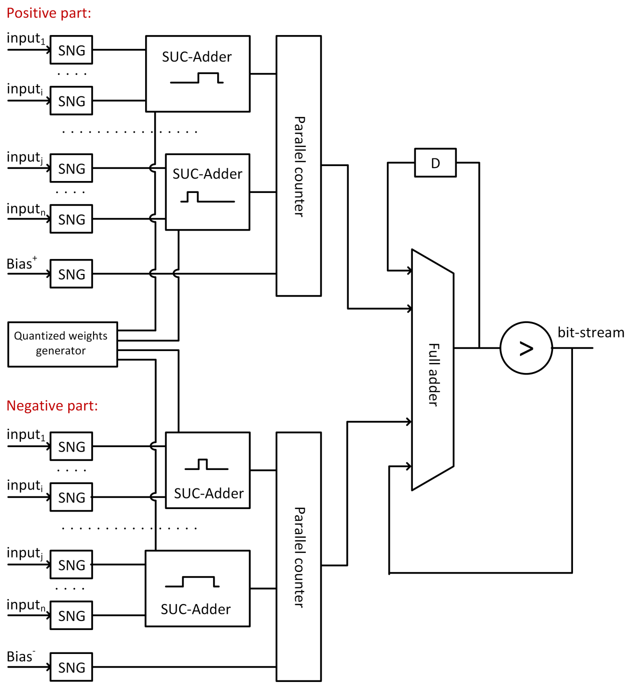


Fig. 8: The stochastic structure of a neuron with matrix multiplication.

In our proposed neuron, inputs are separated into positive and negative groups based on their quantized weights and are ANDed with the quantized weights to compute their products. Then, we group some of products going to the same SUC-Adder. Since number of quantized weights are determined, we can minimize the number of SUC-Adders by adding as many products as possible in one SUC-Adder. For example, we can group two products with weights of 1/4 and 3/4, or three products with weights of 1/4, 1/4 and 2/4. After that, partial addition results and bias are assigned to parallel counter to sum bit-streams up. $bias^+$ and $bias^-$ in the positive and negative parts are extra constant inputs, which are obtained from training process as constant values here. Finally, the full adder will compute the subtraction of positive and negative part and go through a comparator to generate a bit-stream which is the input of next layer. In addition, the activation function is sigmoid function in restricted Boltzmann machine and because the Taylor expansion of the sigmoid function is $sigmoid(x) = 1/2 + x/4 - x^3/48...$, we use the approximate the sigmoid function [21] of $\frac{x+2}{4}$ and it is automatically achieved in the proposed neuron.

TABLE II: Recognition error rates comparisons of RBM classifiers with different bit-lengths.

Quantization	Conventional Binary	Quantized Binary	Our method				Prior work [20]			
			16	32	64	128	16	32	64	128
2-bit	0.98%	1.44%	2.73%	2.71%	2.56%	2.37%	4.29%	2.86%	2.52%	1.92%
3-bit		1.18%	2.59%	2.42%	2.49%	1.82%	7.68%	2.80%	2.24%	2.03%
4-bit		0.99%	2.39%	2.05%	1.85%	1.79%	15.82%	3.36%	2.08%	1.80%

4. Experimental results of Neural Network Comparison

In this section, we use RBM classifier implemented by stochastic computing with a configuration of 784-500-1000-10. The MNIST handwritten digit image dataset [22] is used as input images, which consists of 70,000 data. Among the images, 60,000 images are training data and the rest 10,000 data are used for testing stochastic and conventional neural networks. All weights and coefficients of neural networks are first fully trained and then retrained.

4.1. *Recognition Error Rates*

First, we compared error rates of our stochastic implementation with previous stochastic methods and deterministic RBM implementation. In the stochastic implementation, we compared our work with the stochastic method in [20]. The bit-stream lengths are varied from 16 bits to 128 bits.

As seen in Table II, the quantized binary RBM implementation obtains recognition error rates quite similar to the conventional binary RBM. For the stochastic RBM comparisons, our method obtains slightly higher recognition error rates than the binary implementation. However, compared to the previous work, our implementation achieves much lower recognition error rates at shorter bit lengths.

4.2. *Hardware Cost*

In this section, we focus on hardware cost of different neural network implementations. We use the design compiler to synthesis neural networks with FreePDK 45nm library [23] for the binary and stochastic implementations. The binary implementation uses 8-bit fixed binary.

First, we compare hardware cost of the conventional LFSR and our proposed quantized weight generator. As seen in Table III, our proposed generator has a little lower area and power compared to the conventional LFSR for producing different lengths of bit-streams. Thus, our generator does not introduce any extra overhead in the hardware implementations.

There are mainly two types of implementations in current stochastic neural networks. One type of stochastic implementations [7][6] is based on bipolar format. They normally encode input images and weights as bipolar format bit-streams. Then, to achieve a neuron including matrix multiplication and activation function, the bit-streams go into parallel counters and then go to finite state machines. Finally, outputs of a neuron keep bipolar format and are matched to next layers. Another type of stochastic implementations [20] is unipolar based. Its structure is quite similar to ours. The differences between our method and other unipolar based implementations are stochastic matrix multiplication and quantized weight generator.

TABLE III: Hardware comparison between LFSR and quantized weight generator

Types of SNGs	Area (um^2)	Power (uW)
4-bit LFSR	81.7	15.2
5-bit LFSR	100.4	18.7
4-bit quantized weight generator	74.6	14.4
5-bit quantized weight generator	99	17.2

TABLE IV: Hardware comparison between stochastic RBM with 32-bit length and conventional RBM implementation with SNGs

Neural networks	Area (mm^2)	Power (mW)	$Energy$ (nJ)
Binary	188.6	14668	73.6
Stochastic Unipolar [20]	5.01	558.8	34.89
Stochastic Bipolar [6][7]	6.86	870.0	107.7
Our work	**2.72**	**123.2**	**7.44**

In hardware implementation, we compare our design with two previous stochastic methods and binary implementations. As seen in Table IV, compared to previous stochastic implementations, our architecture obtains about 2.18x, 5.94x, and 9.58x reduction in terms of area, power and energy, respectively. This is because our design significantly decreases the sizes of parallel counters and finally obtains reduction of hardware cost. Moreover, compared to the binary implementation, our approach with 32-bit length derives about 69x, 119x and 10x less area, power and energy, respectively. Furthermore, even we increase the bit-length to 128, our design still has about 2.5x less energy compared to the binary implementation.

5. Conclusion

In this paper, we propose a new stochastic architecture for quantized neural networks. The quantized neural networks are retrained by different quantization levels and obtain recognition error rates similar to floating point neural networks. Then, we propose a new adder called SUC-Adder, which is capable of summing several product bit streams without losing precision. The SUC-Adder can reduce the number of inputs of parallel counters and hence decrease hardware cost of the whole neural networks. In experimental results, in terms of area, power and energy, our approach with 32-bit length derives 2.2x, 6x and 10x reduction compared to previous stochastic implementations, and obtains 69x, 119x and 10x less hardware cost compared to the binary implementation while maintaining slightly higher recognition error rates than the binary implementation.

6. Acknowledgments

This work was supported in part by National Science Foundation grant no. CCF-1408123. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] A. R. Omondi and J. C. Rajapakse, *FPGA implementations of neural networks*. Springer, 2006, vol. 365.

[2] S. Jung and S. su Kim, "Hardware implementation of a real-time neural network controller with a dsp and an fpga for nonlinear systems," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 1, pp. 265–271, 2007.

[3] B. R. Gaines *et al.*, "Stochastic computing systems," *Advances in information systems science*, vol. 2, no. 2, pp. 37–172, 1969.

[4] B. Li, M. H. Najafi, and D. J. Lilja, "An fpga implementation of a restricted boltzmann machine classifier using stochastic bit streams," in *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2015, pp. 68–69.

[5] B. Li, M. H. Najafi, and D. J. Lilja, "Using stochastic computing to reduce the hardware requirements for a restricted boltzmann machine classifier," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 36–41.

[6] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 124.

[7] Z. Li, A. Ren, J. Li, Q. Qiu, Y. Wang, and B. Yuan, "Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks," in *Computer Design (ICCD), 2016 IEEE 34th International Conference on*. IEEE, 2016, pp. 678–681.

[8] J. Li, Z. Yuan, Z. Li, C. Ding, A. Ren, Q. Qiu, J. Draper, and Y. Wang, "Hardware-driven nonlinear activation for stochastic computing based deep convolutional neural networks," *arXiv preprint arXiv:1703.04135*, 2017.

[9] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*. IEEE, 2014, pp. 1–6.

[10] D. Yu, F. Seide, G. Li, and L. Deng, "Exploiting sparseness in deep neural networks for large vocabulary speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4409–4412.

[11] K. Kim, J. Lee, and K. Choi, "An energy-efficient random number generator for stochastic circuits," in *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*. IEEE, 2016, pp. 256–261.

[12] R. Venkatesan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan, "Spintastic: spin-based stochastic logic for energy-efficient computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*. IEEE, 2015, pp. 1575–1578.

[13] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Computation on stochastic bit streams digital image processing case studies," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, no. 3, pp. 449–462, 2014.

[14] W. Qian and M. D. Riedel, "Synthesizing logical computation on stochastic bit streams," *submitted to Communications of the ACM*, 2010.

[15] J. A. Dickson, R. D. McLeod, and H. Card, "Stochastic arithmetic implementations of neural networks with in situ learning," in *Neural Networks, 1993., IEEE International Conference on*. IEEE, 1993, pp. 711–716.

[16] P. Li, D. J. Lilja, W. Qian, M. D. Riedel, and K. Bazargan, "Logical computation on stochastic bit streams with linear finite state machines," *IEEE Transactions on Computers*, p. 1, 2012.

[17] R. Salakhutdinov and G. Hinton, "Deep boltzmann machines," in *Artificial Intelligence and Statistics*, 2009, pp. 448–455.

[18] M. H. Najafi, S. Jamali-Zavareh, D. J. Lilja, M. D. Riedel, K. Bazargan, and R. Harjani, "Time-encoded values for highly efficient stochastic circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1644–1657, 2017.

[19] D. Jenson and M. Riedel, "A deterministic approach to stochastic computation," in *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*. IEEE, 2016, pp. 1–8.

[20] V. T. Lee, A. Alaghi, J. P. Hayes, V. Sathe, and L. Ceze, "Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 13–18.

[21] B. Li, Y. Qin, B. Yuan, and D. J. Lilja, "Neural network classifiers using stochastic computing with a hardware-oriented approximate activation function," in *2017 IEEE 35th International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 97–104.

[22] Y. LeCun and C. Cortes, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2010.

[23] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh *et al.*, "Freepdk: An open-source variation-aware design kit," in *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*. IEEE, 2007, pp. 173–174.