A Hardware Implementation of a Radial Basis Function Neural Network Using Stochastic Logic

Yuan Ji, Feng Ran
Microelectronic Research and Development Center
Shanghai University
Shanghai, 200072, China
jiyuan@shu.edu.cn, ranfeng@shu.edu.cn

Cong Ma, David J. Lilja
Department of Electrical and Computer Engineering
University of Minnesota – Twin Cities
Minneapolis, MN, 55455, USA
maxxx376@umn.edu, lilja@umn.edu

***Abstract*—Hardware implementations of artificial neural networks typically require significant amounts of hardware resources. This paper proposes a novel radial basis function artificial neural network using stochastic computing elements, which greatly reduces the required hardware. The Gaussian function used for the radial basis function is implemented with a two-dimensional finite state machine. The norm between the input data and the center point is optimized using simple logic gates. Results from two pattern recognition case studies, the standard Iris flower and the MICR font benchmarks, show that the difference of the average mean squared error between the proposed stochastic network and the corresponding traditional deterministic network is only 1.3% when the stochastic stream length is 10kbits. The accuracy of the recognition rate varies depending on the stream length, which gives the designer tremendous flexibility to tradeoff speed, power, and accuracy. From the FPGA implementation results, the hardware resource requirement of the proposed stochastic hidden neuron is only a few percent of the hardware requirement of the corresponding deterministic hidden neuron. The proposed stochastic network can be expanded to larger scale networks for complex tasks with simple hardware architectures.**

Keywords—stochastic computing, RBF (radial basis function), ANN (artificial neural network), pattern recognition, Gaussian function, 2D-FSM (two-dimensional finite state machine).

I. INTRODUCTION

Stochastic computing uses a unique data representation that performs mathematical operations with probabilistic streams of bits rather than traditional deterministic values. It uses the ratio of the number "1" bits in a stochastic bit stream to the length of that stream to represent a probabilistic value. For example, the bit stream "01001011" represents 0.5, and the bit stream "0110110101101110" represents 0.625. The ratio will converge to the actual probabilistic value by increasing the stream length [1, 2]. The main advantage of stochastic computing is its great simplicity of hardware arithmetic units. For example, multiplication can be implemented with only a single AND or XNOR gate. It is potentially useful in some arithmetic intensive systems to substitute a stochastic implementation for traditional deterministic hardware units. Additionally, the stochastic implementation can be quite robust to small data fluctuations which can be caused by the manufacturing process or the operating environment in deep submicron devices. Another useful feature is its adjustable data precision provided by using different stream lengths with the same hardware, which provides more flexibility for system design tradeoffs than in conventional architectures [3, 4].

One of the early applications of stochastic computing was implementing ANNs (artificial neural networks) [5, 6] where numerous mathematical operations are required. Brown and Card [7] proposed a number of stochastic computational units that significantly boosted performance, including a linear finite state machine for exponentiation operations. They also applied these stochastic units in an ANN-based SCL (soft competitive learning) circuit for optical character recognition [8]. Zhang and Li [9] further extended the stochastic neural networks applications in the industrial controller of a wind turbine system. However, the hardware implementation of the activation functions in these stochastic neural networks was restricted, and these systems also tended to lose precision due to the scaled addition problem.

In this work, we exploit the advantages of stochastic computing in the RBF (radial basis function) neural network. We extends previous work [10] where a 2D (two-dimensional) finite state machine (FSM) based on a Markov chain is used to synthesize the Gaussian function. With the 2D-FSM, the shape of the Gaussian function can be easily adjusted by varying the input modulation streams. Only eight states in a 2D-FSM are necessary for most tasks. Furthermore, there is no addition operation in the first layer of the network, so the stream length can be relatively short. Inter-stream correlations, which can cause problems in most stochastic systems [3], are actually exploited in this work to perform an absolute subtraction operation. The RBF network is evaluated using two standard pattern recognition case studies – the Iris flower and the MICR optical character recognition. The results show that the recognition rate and MSE (mean squared error) of the proposed stochastic RBF network are very close to the output of the corresponding deterministic RBF network, but the hardware architecture is greatly simplified. The cost-performance tradeoffs are also compared between the proposed stochastic RBF network and three deterministic networks.

This work was supported in part by China National Natural Science Fund under Grant Number 61376028, and "2013 special program of intelligent manufacture and advanced materials'' of Shanghai Science and Technology Commission under Grant Number 13111104600.

This work was supported in part by National Science Foundation grants CCF-1241987 and 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.

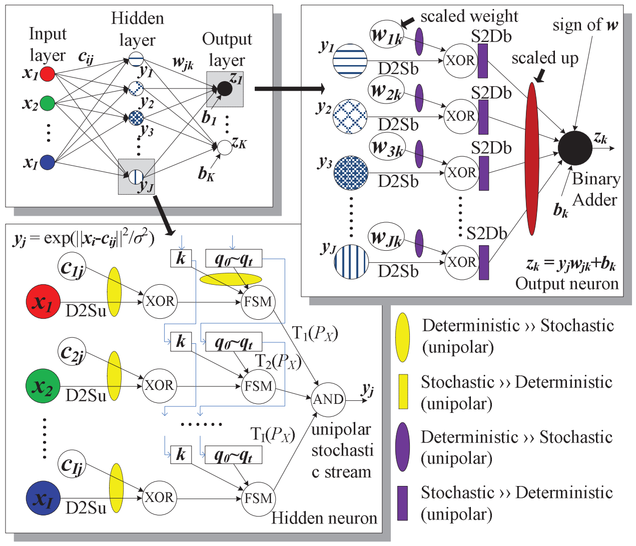
Input layer
Hidden layer
Output layer
scaled weight
sign of w
scaled up
D2Sb
S2Db
XOR
Binary Adder
Output neuron
Hidden neuron
FSM
AND
unipolar stochastic stream
D2Su
Deterministic ›› Stochastic (unipolar)
Stochastic ›› Deterministic (unipolar)
Deterministic ›› Stochastic (unipolar)
Stochastic ›› Deterministic (unipolar)

Fig. 1. Architecture of the proposed stochastic radial basis function network

II. Stochastic RBF Network

The proposed stochastic RBF network is shown in Fig. 1. The hidden layer and the output layer are implemented with stochastic computing elements. Network input $\boldsymbol{x}_i$ ($i = 1, 2, \cdots, I$) is converted into stochastic bit streams by D2Su. The hidden layer output $\boldsymbol{y}_j$ ($j = 1, 2, \cdots, J$) is implemented with a Gaussian function and calculated in (1):

$$\boldsymbol{y}_j = exp(-\| \boldsymbol{x}_i - \boldsymbol{c}_{ij} \|^2/\sigma^2) \quad (1),$$

where $\boldsymbol{c}_{ij}$ is the center point and σ controls the shape of $\boldsymbol{y}_j$. We expand (1) using the Euclidean distance as follows:

$$\boldsymbol{y}_j = exp[-(\boldsymbol{x}_1 - \boldsymbol{c}_{1j})^2/\sigma^2] \cdot exp[-(\boldsymbol{x}_2 - \boldsymbol{c}_{2j})^2/\sigma^2] \cdot \ldots \cdot exp[-(\boldsymbol{x}_I - \boldsymbol{c}_{Ij})^2/\sigma^2] \quad (2),$$

Therefore, $\boldsymbol{y}_j$ can be calculated in three steps: (1) subtraction between $\boldsymbol{x}_i$ and $\boldsymbol{c}_{ij}$; (2) calculation of the Gaussian function with zero mean and standard deviation σ; (3) multiplication of all these Gaussian functions. The subtraction in step (1) is implemented by XOR gates with fully correlated input streams (e.g. generated with the same randomizer). The output bit stream of the XOR gate will be set to "0" when the two input bits are different, otherwise it is set to "1". The multiplication in step (3) is implemented by an AND gate using the unipolar coding format [1]. The Gaussian function in step (2) is approximated by the 2D-FSM as shown in Fig. 2, where states can transition within the two-dimensional state array. There are M×N states ($\boldsymbol{S}_0$ to $\boldsymbol{S}_{MN-1}$) in this configuration. The state transition depends on the input stream $\boldsymbol{x}$ and the modulation stream $\boldsymbol{k}$. It can be modeled as a time-homogeneous Markov chain which is irreducible, aperiodic and ergodic. If the probability of the states is denoted by P_{st} (st is the state number, $st = 0, 1, \cdots, MN-1$) and the probability of the input stream and the modulation stream are denoted by P_X and P_K, respectively, then P_{st} can be calculated by (3) [10]:

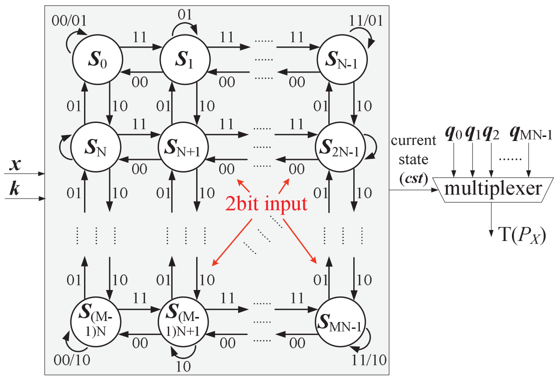

Fig. 2. State transitions of the 2D finite state machine

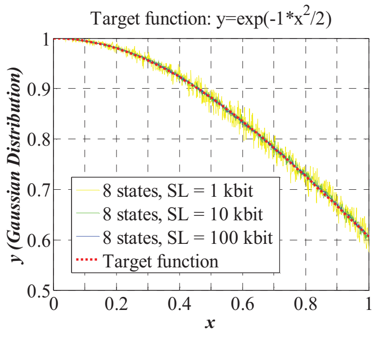

Fig. 3. Gaussian function approximation examples using the 2D-FSM

$$P_{st} = P_{i\times N+j} = t_x^{\,i} \cdot t_y^{\,i} / (\Sigma_u \Sigma_v t_x^{\,u} \cdot t_y^{\,v}) \quad (3)$$

where i and j are the horizontal and vertical state numbers, u is from 0 to M-1, v is from 0 to N-1, $t_x = (P_X \cdot P_K) / [(1-P_X)\cdot(1-P_K)]$, and $t_y = [P_X(1-P_K)] / [P_K(1-P_X)]$. The current state of the 2D-FSM (cst) is connected to the selection port of the multiplexer, which selects the predefined bit streams $\boldsymbol{q}$ as the output T(P_X). T(P_X) can be approximated by minimizing the error □ between the target function and the actual output. Fig. 3 shows examples of the Gaussian function synthesized with an 8-state 2D-FSM. Note that the variance decreases with longer stream lengths.

Vector $\boldsymbol{z}_k$ ($k = 1, 2, \cdots, K$) is the network output. It is a linear combination of the hidden layer and calculated in (4):

$$\boldsymbol{z}_k = \boldsymbol{y}_j \cdot \boldsymbol{w}_{jk} + \boldsymbol{b}_k \quad (4)$$

where $\boldsymbol{w}_{jk}$ is the weight matrix of the linear output layer and vector $\boldsymbol{b}_k$ is the bias. $\boldsymbol{z}_k$ is implemented using either stochastic or deterministic logic. For deterministic logic, pipelined multipliers are used. For stochastic logic, a deterministic-to-stochastic converter is used to scale down the value of $\boldsymbol{w}$ to be within the range [-1, 1]. $\boldsymbol{y}$ and $\boldsymbol{w}$ are multiplied using XOR gates. The multiplication results are converted to deterministic values by binary counters (S2Db), and need to be scaled up proportionally. The sign of $\boldsymbol{w}$ is processed in the binary adder. The scaling down and up process introduces additional variance, but it avoids using the stochastic scaled adder.

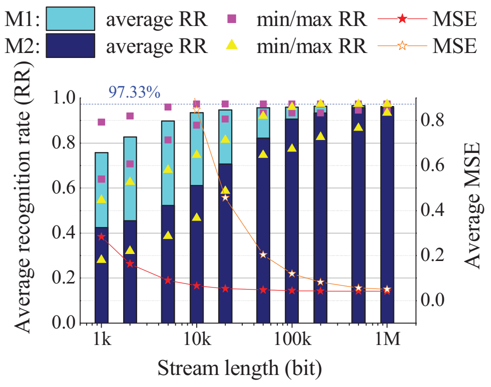

Fig. 4. Iris flower recognition rate and MSE

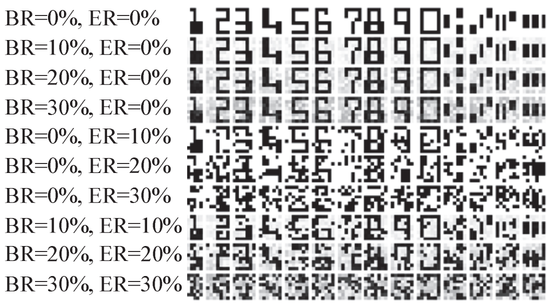

Fig. 5. E-13B MICR font with noise (BR: blur rate, ER: error rate)

III. Experimental Results and Discussion

A. *Iris flower recognition*

Fisher's Iris flower data set [11] consists of 50 samples of three species of Iris. Each sample has four features: length and width of the sepals and petals. We used this data set for pattern recognition to compare the deterministic and stochastic RBF networks. A total of 75 samples were randomly selected for network training and the remaining 75 were used for testing. The orthogonal least squares training algorithm was used. A RBF network with 4 input neurons, 8 hidden neurons and 3 output neurons was simulated. For the deterministic network, 2 out of the 75 samples were not recognized. For the stochastic network, the Gaussian function was approximated using a 2D-FSM with eight states. Two variations, M1 and M2, were tested. M1 used stochastic logic only in the hidden layer. M2 implemented the full stochastic logic. The test results are shown in Fig. 4. Each result was repeated at least ten thousand times to reduce randomization errors. The MSE is calculated by comparing the stochastic output with the corresponding deterministic output. In M1, the average recognition rate reaches 93.4% with a stream length of 10kbits, and converges to 96.7% when the stream length exceeds 500kbits. In M2, the average recognition rate converges to 95.6% at a stream length of 1Mbits. The MSE converges to 0.051, which is nearly 20% worse than M1. This is because M2 uses stochastic logic in the output layer, which leads to a larger variance. The variance can be reduced by using longer bit streams, resulting in a trade-off between the system speed and accuracy.

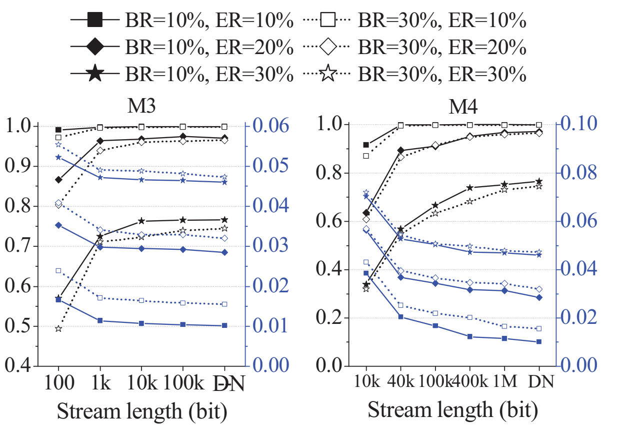

Fig. 6. MICR font recognition rate and MSE (DN: determinisitc network). Black font: Average recognition rate. Blue font: Average MSE.

B. *Optical character recognition*

The characters of the E-13B MICR (Magnetic Ink Character Recognition) font are used mainly in the banking industry for magnetic scanning and optical recognition. In this test, random noise at two levels was added to the MICR font: pixel blur (modify the pixel gray level), and pixel error (reverse the pixel color), as shown in Fig. 5. A 7×8 pixel array for each character was used. The pixel gray levels are normalized. A RBF network with 56 input neurons, 15 hidden neurons and 14 output neurons was set up with the orthogonal least squares training algorithm. Two variations, M3 and M4, were tested. M3 used stochastic logic only in the hidden layer. M4 used full stochastic logic. The test results are shown in Fig. 6. Each sample was repeated at least two thousand times. The recognition rate and the MSE gradually converge to the corresponding deterministic network outputs. Compared with the Iris flower test, a much shorter stream length is needed to obtain the same recognition rate level, mainly because there are more inputs and fewer outputs in the MICR font test. For M4, When ER is below 10% and the stream length is greater than 40kbits, the recognition rate is almost 100%. When ER is 20%, the recognition rate is around 90% at a stream length of 40kbits, and is 96.7% at a stream length of 1Mbit. The MSE gets closer to the deterministic network result when the stream length increases. When the bit stream length is 10kbits, the recognition difference between the average stochastic network and the corresponding deterministic network is only 1.3%.

C. *Hardware comparisons*

The proposed stochastic RBF network was realized on an Altera Cyclone III FPGA EP3C80F780C8 with 81264 logic elements (LEs). The synthesis results are shown in Table I. For M1 and M3, the output neurons are implemented with binary adders and multipliers, so their areas are larger than M2 and M4, which use only stochastic components. The data width refers to the bit stream length in the stochastic network. The network parameter (***c***, ***k***, ***q*** and ***w***) are stored in an off-network memory. They are transferred to the stochastic network by shift registers. As the data width increases, the network circuit area increases linearly, but the area of the hidden neurons does not change given the same network architecture.

TABLE I. FPGA IMPLEMENTATION OF THE STOCHASTIC RBF NETWORK

Case	Control Logic (LE)	Hidden Neuron (LE)	Output (LE)	LFSR (LE)	Total (LE)
Neural network (Input × Hidden × Output) 4×8×3					
M1 (10bit)	720	712 / 89	294	60	2455
M1 (20bit)	1440	712 / 89	986	120	5389
M2 (10bit)	720	712 / 89	132	70	1898
M2 (20bit)	1440	712 / 89	254	140	3053
Neural network (Input × Hidden × Output) 56×15×14					
M3 (10bit)	11290	18765 / 1251	4990	590	35635
M3 (20bit)	22580	18765 / 1251	15609	1180	58134
M4 (10bit)	11290	18765 / 1251	3668	590	34313
M4 (20bit)	22580	18765 / 1251	8092	1180	50617

TABLE II. FEATURES OF DIFFERENT HIDDEN NEURONS (PER INPUT)

Hardware Architecture	A: Area (LE)	C: Clock /OPT	O: OPT/ Sec	R0= 1/AC	R1= 1/A^{2}C	R2= O/A
10bit 2D-FSM	22	1024	244	1	1	1
12bit 2D-FSM	22	4096	61	0.20	0.25	0.23
14bit 2D-FSM	22	16384	15	0	0.06	0.04
12bit look-up table	33898	1	10000	0.64	0	0
12bit look-up table with interpolation	1853	15	3333	0.80	0.01	0.11
12bit CORDIC	1079	22	5455	0.95	0.02	0.28

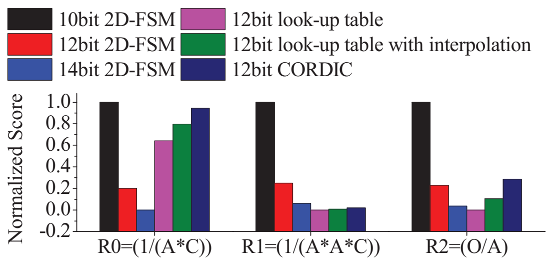

Fig. 7. Comparison between different architectures for the hidden neurons. (A: circuit area, C: clock cycles per operation, O: number of operations per second). A higher normalized score indicates better overall performance.

Table II compares the stochastic 2D-FSM and three other deterministic implementations on the circuit area of the hidden neuron amortized by each input (denoted by A, measured by LEs), the number of clock cycles required for each operation (denoted by C), and the maximum number of operations per second (denoted by O, and determined by the maximum clock frequency). The 2D-FSM always takes 22 LEs regardless of the data width. The full look-up table architecture needs only one clock cycle for each operation, but it has an extremely large area and slow clock frequency. The look-up table with the linear interpolation uses a smaller table but requires more cycles for data interpolation. Table II also shows three cost-performance normalizations for these architectures : R0 = 1 / (area * number of clock cycles), R1 = 1 / (area-squared * number of clock cycles), R2 = (operations per unit area). These metrics are compared in Fig. 7. The 10-bit 2D-FSM architecture has the best area-clock product. This product becomes smaller when the data width increases. R0 also measures the circuit power which is a factor of area and speed. As the circuit area is highlighted by R1, the 2D-FSM architectures show better area-clock product even if the data width exceeds 14 bits. For area efficiency, which is measured by R2, the 12-bit 2D-FSM architecture is comparable with the CORDIC (Coordinate rotation digital computer) architecture. It can be seen that the 2D-FSM has great advantages in terms of circuit area, but its performance (speed) must be balanced with the accuracy, which is a function of the bit stream length.

IV. CONCLUSION

An RBF neural network with two stochastic logic levels was designed and evaluated. The results show that this network can perform recognition tasks with an acceptably low error rate using very simple hardware consisting of only basic logic gates (XOR, AND) and simple FSMs. The output of the stochastic network is a probabilistic value that fluctuates around an average value. The difference of this average value compared to the output of a deterministic network is only 1.3% when the stochastic bit stream length is 10kbits. As the stream length increases, the variance of the output reduces, but the operation speed is reduced appreciably. The deterministic network can produce greater speed with lower power than the stochastic implementation, but the circuit area cost of the stochastic approach is substantially lower and its accuracy can be adjusted by varying the stream length without changing the hardware. This feature gives the designer more flexibility to balance the accuracy, speed and power. The stochastic network also is able to continue working even with significant input data errors or when some logic units fail. These specific error tolerance characteristics are the focus of our future research.

REFERENCES

[1] B. R. Gaines, "Stochastic Computing Systems," *Advances in Information Systems Science*, J. F. Tou, ed., vol. 2, no. 2, pp. 37-172, New York: Plenum, 1969.

[2] W. K. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An Architecture for Fault-Tolerant Computation with Stochastic Logic," *IEEE Transactions on Computers*, vol. 60, pp. 93-105, 2011.

[3] A. Alaghi and J. P. Hayes, "Survey of Stochastic Computing," *ACM Trans. Embed. Comput. Syst.*, vol. 12, pp. 1-19, 2013.

[4] L. Peng, D. J. Lilja, Q. Weikang, K. Bazargan, and M. D. Riedel, "Computation on Stochastic Bit Streams Digital Image Processing Case Studies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, pp. 449-462, 2014.

[5] M. van Daalen, P. Jeavons, and J. Shawe-Taylor, "A stochastic neural architecture that exploits dynamically reconfigurable FPGAs," *in Proc. IEEE Workshop on FPGAs for Custom Computing Machines,* pp. 202-211, 1993.

[6] S. L. Bade and B. L. Hutchings, "FPGA-based stochastic neural networks-implementation," *in Proc. IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 189-198, 1994.

[7] B. D. Brown and H. C. Card, "Stochastic neural computation. I. Computational elements," *IEEE Transactions on Computers*, vol. 50, pp. 891-905, 2001.

[8] B. D. Brown and H. C. Card, "Stochastic neural computation. II. Soft competitive learning," *IEEE Transactions on Computers*, vol. 50, pp. 906-920, 2001.

[9] L. Hui, Z. Da, and S. Y. Foo, "A Stochastic Digital Implementation of a Neural Network Controller for Small Wind Turbine Systems," *IEEE Transactions on Power Electronics*, vol. 21, pp. 1502-1507, 2006.

[10] L. Peng, D. J. Lilja, Q. Weikang, K. Bazargan, and M. Riedel, "The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic," *in Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 480-487, 2012.

[11] http://archive.ics.uci.edu/ml/datasets/Iris.