An FPGA Implementation of a Restricted Boltzmann Machine Classifier Using Stochastic Bit Streams

Bingzhe Li, M. Hassan Najafi and David J. Lilja
Department of Electrical and Computer Engineering, University of Minnesota–Twin Cities
Minneapolis, MN, USA, 55455
{lixx1743, najaf011, lilja}@umn.edu

***Abstract*—Artificial neural networks (ANNs) usually require a very large number of computation nodes and can be implemented either in software or directly in hardware, such as FPGAs. Software-based approaches are offline and not suitable for real-time applications, but they support a large number of nodes. FPGA-based implementations, in contrast, can greatly speedup the computation time. However, resource limitations in an FPGA restrict the maximum number of computation nodes in hardware-based approaches. This work exploits stochastic bit streams to implement the Restricted Boltzmann Machine (RBM) handwritten digit recognition application completely on an FPGA. Exploiting this approach saves a large number of hardware resources making the FPGA-based implementation of large ANNs feasible.**

I. Introduction

Software-based implementations running on general-purpose processors are the main approach for implementing large Artificial neural networks (ANNS). With growing interest in neural networks, researchers look for new methods to achieve low-power back-end applications or high-performance neural networks rather than being restricted to the slow, non-portable, software-based approaches. A solution to the limitations of the software-based approaches is an FPGA-based implementation. However, this approach is limited by its high power consumption and substantial resource requirements.

The Restricted Boltzmann Machine (RBM) is a type of artificial neural network that is capable of solving difficult problems. Like other machine learning models, RBM has two types of processes – learning and testing. During learning, the system is presented with a large number of input examples and desired outputs to generate a suitable RBM structure that learns a general rule to map inputs to outputs. In the testing process, the RBM produces outputs for new inputs following the general rule obtained in the learning process.

Implementing a fully parallel specially designed hardware version of a neural network in a single FPGA is expensive, and in most cases is not even possible using today's FPGAs [1]. Recently, a fully pipelined FPGA architecture of a factored RBM has been implemented [2]. Although the virtualized architecture proposed in this work could implement a neural network with a maximum of 4096 nodes using time multiplex sharing, the largest achievable RBM without virtualization was on the order of 256 nodes.

Stochastic computing (SC) exploits the unique characteristics of computations using random streams of bits. This work shows that SC can be used to overcome FPGA hardware limitations. Stochastic bit streams can be encoded using either a unipolar or a bipolar format. The main motivation of using SC is the simplicity of the computational elements involved. For example, multiplication or scaled addition can be performed using only a single AND or MUX gate [3]. In this paper, we exploit SC to implement the testing process of an RBM with a massive number of computation nodes based on the Hinton et al [4] learning codes.

The stochastic architecture proposed in this work is used to implement a two-layer RBM classifier, which classifies the most well-known handwriting digit image recognition data set, MNIST, completely on a single FPGA. We develop a specific stochastic adder module, *Uni_pos_neg*, which can automatically handle the format conversions required in implementing stochastic matrix multiplications. Equations (1), (2) and (3) show the functions of different RBM layers [5]:

$$\mathbf{w1p} = 1/(1 + \exp(\mathbf{data} * \mathbf{w1\_vishid} + \mathbf{coef})) \quad (1)$$

$$\mathbf{w2p} = 1/(1 + \exp(-\mathbf{w1p} * \mathbf{w2} - \mathbf{bias\_pen})) \quad (2)$$

$$\mathbf{out} = \mathbf{w2p} * \mathbf{w\_class} + \mathbf{bias\_top} \quad (3)$$

where $\mathbf{data}$ (1×784) matrix is a 28×28 handwritten input image from MNIST data set, $\mathbf{coef}$ $(1 \times n)$, $\mathbf{w1\_vishid}$ $(784 \times n)$, $\mathbf{w2}$ $(n \times m)$, $\mathbf{bias\_pen}$ $(1 \times m)$, $\mathbf{w\_class}$ $(m \times 10)$, and $\mathbf{bias\_top}$ (1×10) are the RBM constants computed by the learning process. $\mathbf{w1p}$ and $\mathbf{w2p}$ are probabilities of the first and second hidden layers.

II. Stochastic Design of RBM Handwriting Classification

A. RBM Data Flow:

As shown in (1), (2) and (3), both the first and second RBM layers use similar operations, including multiplications and additions, in the same functional format, $1/(1 + exp(x))$. However, the stochastic computation elements used in the proposed architecture require different encoding formats in their inputs while not always producing their outputs in the same format. Fig.1 shows the implementation dataflow of the proposed stochastic RBM. For the details of the FSM-based stochastic $\tanh$ function and the stochastic comparator, refer to [6] and [7].

B. Uni_pos_neg adder:

The circuits used in the proposed stochastic RBM need to work with both positive and negative values. However, unipolar stochastic streams are suitable for representing only positive or only negative values. Since the stochastic $\tanh$ function [6] accepts the input streams in only bipolar format, the output streams generated from the stochastic multiplication of matrices also need to be in the same format. Thus, we develop a special *Uni_pos_neg* adder module that can automatically handle the required format conversions for the stochastic matrix multiplication using only AND, OR, NOT, and MUX units, as shown in Fig.2.

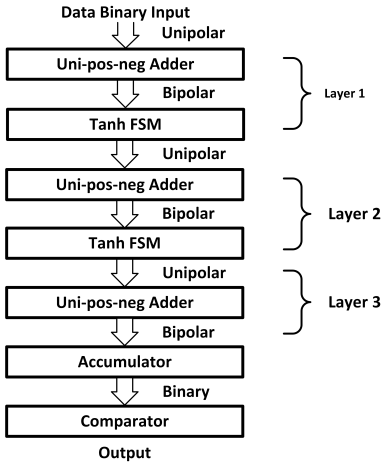

Fig. 1. The RBM bit stream implementation data flow

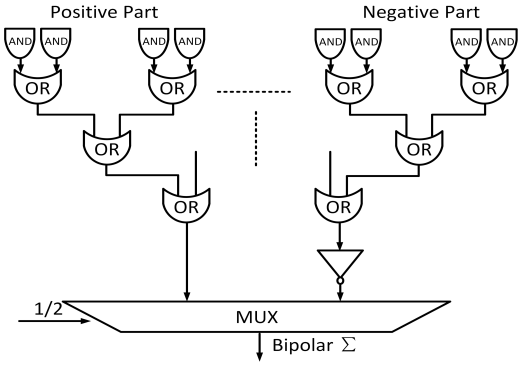

Fig. 2. Uni-pos-neg-Adder: Large matrix multiplication hardware structure with positive and negative inputs

C. Sigmoid Function:

The sigmoid operation is an important function required in the stochastic RBM. We implement this function using transformations on (4) to obtain (5). Based on (5), the sigmoid function could be considered as the bipolar encoding of the $\tanh$ function. Fig. 3 shows that both the input and the output of the $\tanh$ function are in bipolar format by default. However, to match with the restrictions of the proposed RBM structure, we obtain the sigmoid function with bipolar inputs and a unipolar output based on (5).

$$tanh(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x}) \tag{4}$$

$$1/(1 + e^{-Nx}) = (1 + tanh(Nx/2))/2 \tag{5}$$

III. EXPERIMENTAL RESULTS

The proposed architecture with different layer sizes was implemented using Verilog HDL, then synthesized, placed and routed using the Xilinx ISE Design Suite 14.7 on a Virtex7 xc7v2000t FPGA. Functional verification of the proposed architecture has been done in Matlab where we implemented both the conventional approach [4] and also the proposed stochastic architecture to work on 10,000 MNIST handwritten test images. Table I shows the hardware resource requirements, the latency for testing each input image, and also the average output rate of classifying all input images when using the implemented stochastic RBMs with 1024-bit stochastic streams. As can be seen in this table, by shrinking the layer sizes, the number of LUTs is reduced dramatically whereas the latency

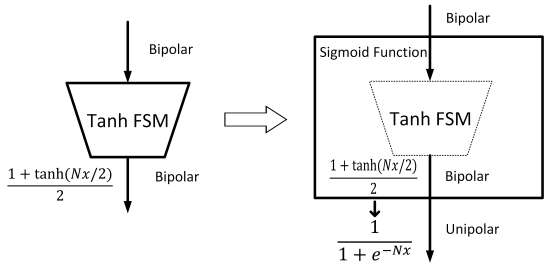

Fig. 3. Sigmoid function implementation with bipolar inputs and a unipolar output from the $\tanh$ function

TABLE I. AREA, LATENCY AND ERROR RATE OF THE IMPLEMENTED RBMS

Size	Area(# of LUTs)	Latency(s)	Output error rate (%)
$100 \times 200 \times 10$	144,450	8.561^{-6}	5.72%
$300 \times 600 \times 10$	603,750	9.797^{-6}	2.92%
$500 \times 1000 \times 10$	1,292,310	1.077^{-5}	2.32%

of the handwritten digit recognition does not experience much change. However, the error rate increases with a very low slope when reducing the layer size or the number of computation nodes. The important point is that even with the largest selected layer size, the proposed stochastic RBM could be completely placed and routed in the current available FPGAs.

IV. CONCLUSION

In this paper, we proposed an FPGA implementation of a restricted Boltzmann Machine classifier using stochastic logic. In a large RBM network, the matrix multiplications and the sigmoid functions are the most expensive operations to be implemented. We proposed the *Uni_pos_neg* adder to implement the stochastic matrix multiplication and used a modified FSM $\tanh$ function to achieve the sigmoid function.

ACKNOWLEDGMENT

This work was supported in part by National Science Foundation grants no. 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.

REFERENCES

[1] J. Zhu and P. Sutton, "Fpga implementations of neural networks–a survey of a decade of progress," in *Field Programmable Logic and Application*. Springer, 2003, pp. 1062–1066.

[2] L.-W. Kim, S. Asaad, and R. Linsker, "A fully pipelined fpga architecture of a factored restricted boltzmann machine artificial neural network," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 1, p. 5, 2014.

[3] M. Najafi and M. Salehi, "A fast fault-tolerant architecture for sauvola local image thresholding algorithm using stochastic computing," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[4] G. E. Hinton and R. Salakhutdinov, "A better way to pretrain deep boltzmann machines," in *Advances in Neural Information Processing Systems*, 2012, pp. 2447–2455.

[5] R. Salakhutdinov and G. Hinton, "An efficient learning procedure for deep boltzmann machines," *Neural computation*, vol. 24, no. 8, pp. 1967–2006, 2012.

[6] 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.

[7] 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.